Accurate Smart Contract Verification through Direct Modelling

Matteo Marescotti[1], **Rodrigo Otoni**[1], Leonardo Alt[2],
Patrick Eugster[1], Antti E. J. Hyvärinen[1], and Natasha Sharygina[1]

[1] Università della Svizzera italiana, Lugano, Switzerland
[2] Ethereum Foundation, Zug, Switzerland

2nd June 2021

Foreword

Industrial application of our work

- Component of the SMTChecker module of the Solidity compiler
  - SMTChecker's constrained Horn clauses model checking engine

Availability of our tool

- `github.com/ethereum/solidity`
- Add `pragma experimental SMTChecker;` to the source file

Formal Verification of Smart Contracts

Essential for safety

Formal Verification of Smart Contracts

Essential for safety

Many works published

Formal Verification of Smart Contracts

Essential for safety

Many works published

- Symbolic execution, e.g.:
  - Oyente [Luu et al., CCS'16]
  - Maian [Nikolić et al., ACSAC'18]
  - Manticore [Mossberg et al., ASE'19]
  - VerX [Permenev et al., S&P'20]
- Model checking, e.g.:
  - Zeus [Kalra et al., NDSS'18]
  - SAFEVM [Albert et al., ISSTA'19]
  - VeriSol [Wang et al., VSTTE'20]
- Others, e.g.:
  - F* [Bhargavan et al., PLAS'16]
  - KEVM [Hildenbrandt et al., CSF'18]
  - Securify [Tsankov et al., CCS'18]
  - Why3 [Nehaï and Bobot, FM'19]
  - Solc-Verify [Hajdu and Jovanovic, VSTTE'19, ESOP'20]

Formal Verification of Smart Contracts

Essential for safety

Many works published

Limitations to be addressed

- Symbolic execution, e.g.:
  - Oyente [Luu et al., CCS'16]
  - Maian [Nikolić et al., ACSAC'18]
  - Manticore [Mossberg et al., ASE'19]
  - VerX [Permenev et al., S&P'20]
- Model checking, e.g.:
  - Zeus [Kalra et al., NDSS'18]
  - SAFEVM [Albert et al., ISSTA'19]
  - VeriSol [Wang et al., VSTTE'20]
- Others, e.g.:
  - F* [Bhargavan et al., PLAS'16]
  - KEVM [Hildenbrandt et al., CSF'18]
  - Securify [Tsankov et al., CCS'18]
  - Why3 [Nehaï and Bobot, FM'19]
  - Solc-Verify [Hajdu and Jovanovic, VSTTE'19, ESOP'20]

Common Feature: Domain Translation

Translation for reuse

- Established off-the-shelf frameworks, e.g.:
  - Boogie
  - LLVM
- Programming languages ecosystems, e.g.:
  - C/C++
  - WhyML

Common Feature: Domain Translation

Translation for reuse

- Established off-the-shelf frameworks, e.g.:
  - Boogie
  - LLVM
- Programming languages ecosystems, e.g.:
  - C/C++
  - WhyML

Translation layer drawbacks

- Error prone to develop
- Requires correctness proofs
- Adverse effect on precision and efficiency

Common Feature: Domain Translation

Translation for reuse

- Established off-the-shelf frameworks, e.g.:
  - Boogie
  - LLVM
- Programming languages ecosystems, e.g.:
  - C/C++
  - WhyML

Translation layer drawbacks

- Error prone to develop
- Requires correctness proofs
- Adverse effect on precision and efficiency

Our approach

- Direct modelling into the target formalism
  - Focus on Solidity contracts
  - Modelling in constrained Horn clauses (CHCs)

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}. (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}. P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}') \leftarrow \mathcal{P}^v_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}) \wedge \text{SSA}_{\lambda_v}(\boldsymbol{l}, \boldsymbol{l}') \wedge \text{SSA}_{\mu_e}(\boldsymbol{l}')$$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}') \leftarrow \mathcal{P}^v_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}) \wedge \text{SSA}_{\lambda_v}(\boldsymbol{l}, \boldsymbol{l}') \wedge \text{SSA}_{\mu_e}(\boldsymbol{l}')$$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}') \leftarrow \mathcal{P}^v_f(\underline{\boldsymbol{s}}, \boldsymbol{a}, \boldsymbol{l}) \wedge \text{SSA}_{\lambda_v}(\boldsymbol{l}, \boldsymbol{l}') \wedge \text{SSA}_{\mu_e}(\boldsymbol{l}')$$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}') \leftarrow \mathcal{P}^v_f(\boldsymbol{s}, \underline{\boldsymbol{a}}, \boldsymbol{l}) \wedge \text{SSA}_{\lambda_v}(\boldsymbol{l}, \boldsymbol{l}') \wedge \text{SSA}_{\mu_e}(\boldsymbol{l}')$$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}') \leftarrow \mathcal{P}^v_f(\boldsymbol{s}, \boldsymbol{a}, \underline{\boldsymbol{l}}) \wedge \text{SSA}_{\lambda_v}(\boldsymbol{l}, \boldsymbol{l}') \wedge \text{SSA}_{\mu_e}(\boldsymbol{l}')$$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}') \leftarrow \mathcal{P}^v_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \text{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I}') \wedge \text{SSA}_{\mu_e}(\boldsymbol{I}')$$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}') \leftarrow \mathcal{P}^v_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \underline{\text{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I}')} \wedge \text{SSA}_{\mu_e}(\boldsymbol{I}')$$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}') \leftarrow \mathcal{P}^v_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \text{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I}') \wedge \underline{\text{SSA}_{\mu_e}(\boldsymbol{I}')}$$

Smart Contracts Verification through Direct Modelling

- Rule-based encoding for assertion checking, starting with the contracts' control-flow graphs

CHC rule format

$$\forall \mathcal{V}.\ (\underbrace{H(\boldsymbol{x})}_{\textit{head}} \leftarrow \underbrace{\exists \boldsymbol{y}.\ P_1(\boldsymbol{y}) \wedge \ldots \wedge P_n(\boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})}_{\textit{body}})$$

with $\mathcal{V}$ a set of variables over a background theory $\mathcal{T}$, $\boldsymbol{x} \cup \boldsymbol{y} \subseteq \mathcal{V}$, $n \geq 0$

Rule $\text{Jump}_{f,e}$

$$\mathcal{P}^u_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}') \leftarrow \mathcal{P}^v_f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{l}) \wedge \text{SSA}_{\lambda_v}(\boldsymbol{l}, \boldsymbol{l}') \wedge \text{SSA}_{\mu_e}(\boldsymbol{l}')$$

Example: an Auction in Solidity

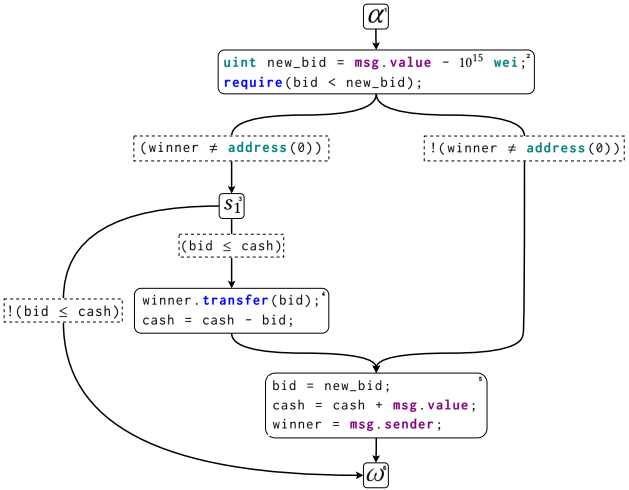
```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             //assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```



Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

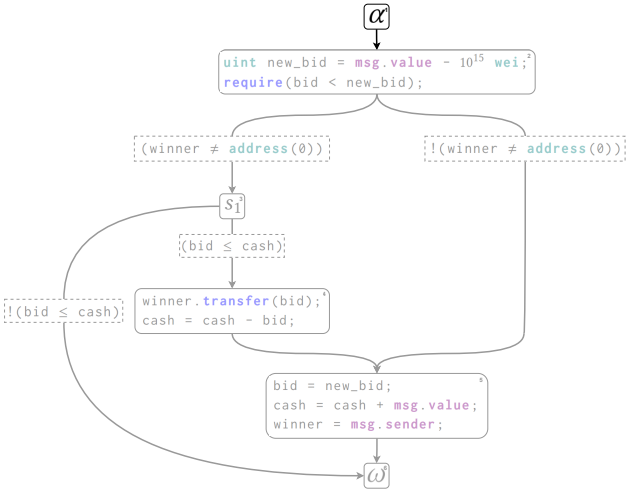
$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathsf{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathsf{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\begin{aligned}
\mathcal{P}_o^2 &\leftarrow \mathcal{P}_o^1 \\
\mathcal{P}_o^3 &\leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0 \\
\mathcal{P}_o^5 &\leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0) \\
\mathcal{P}_o^4 &\leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c \\
\mathcal{P}_o^6 &\leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3 \\
\mathcal{P}_o^5 &\leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b \\
\mathcal{P}_o^6 &\leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s
\end{aligned}$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```



Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathsf{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathsf{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$
$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$
$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}') \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathsf{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I}') \wedge \mathsf{SSA}_{\mu_e}(\boldsymbol{I}')$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$

$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$

$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$

$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$

$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$

$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$

$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathsf{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathsf{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$
$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$
$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathsf{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathsf{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$
$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$
$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

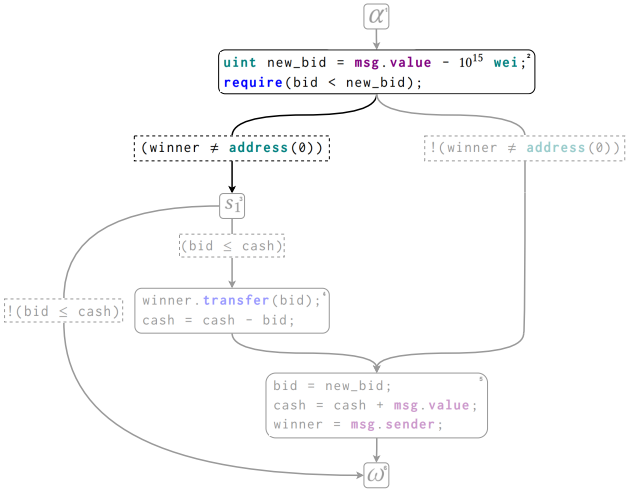
$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathsf{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathsf{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$
$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$
$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```



Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathrm{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathrm{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$

$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$

$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$

$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$

$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$

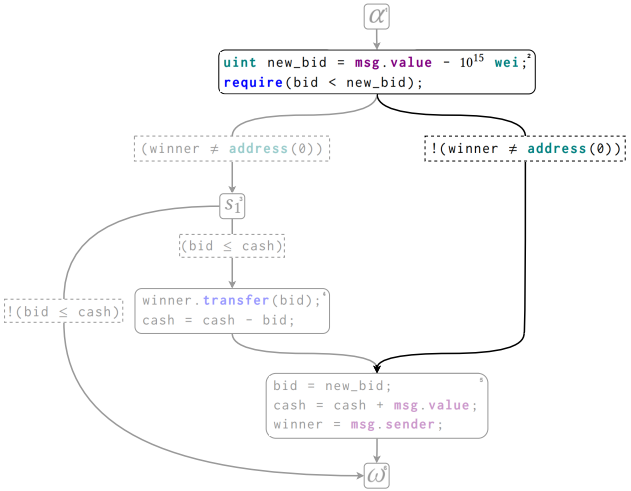
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$

$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```



Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathrm{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathrm{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$

$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$

$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$

$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$

$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$

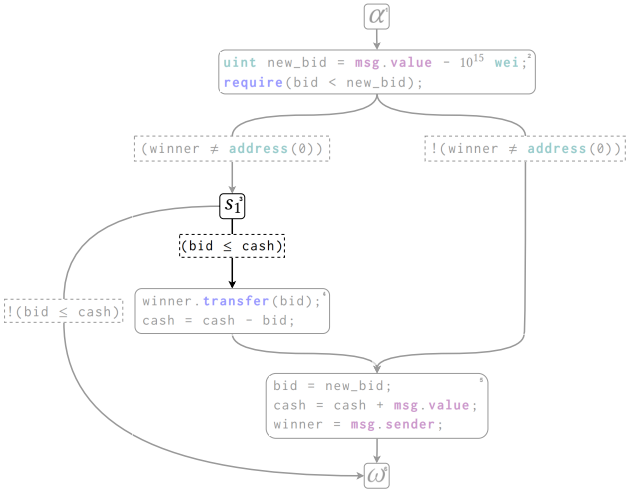
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$

$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```



Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

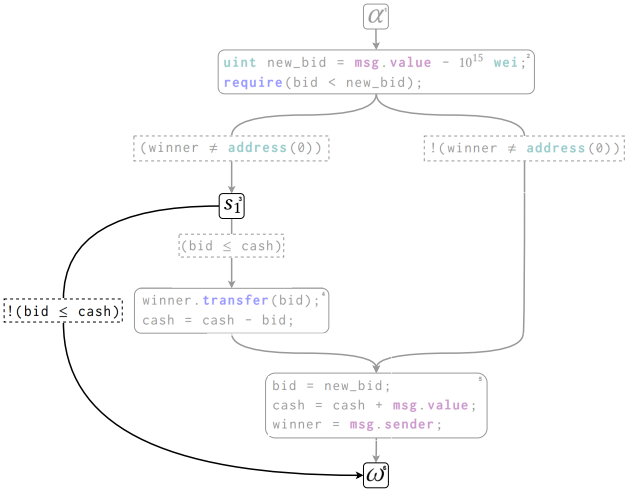
$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathrm{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathrm{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$
$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$
$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```



Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

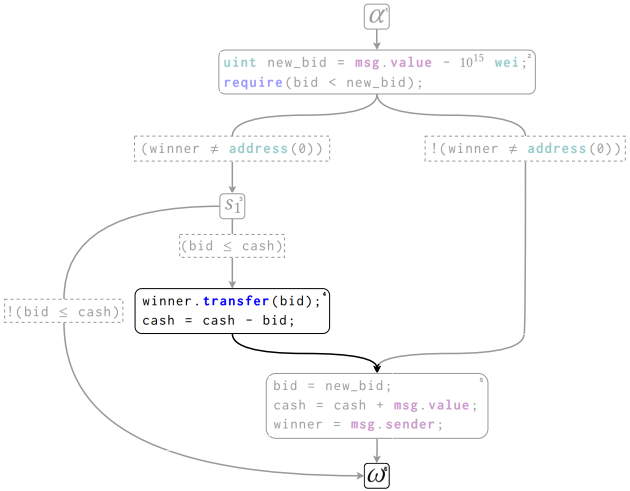
$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathrm{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathrm{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$
$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$
$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```



Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathrm{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathrm{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$

$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$

$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$

$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$

$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$

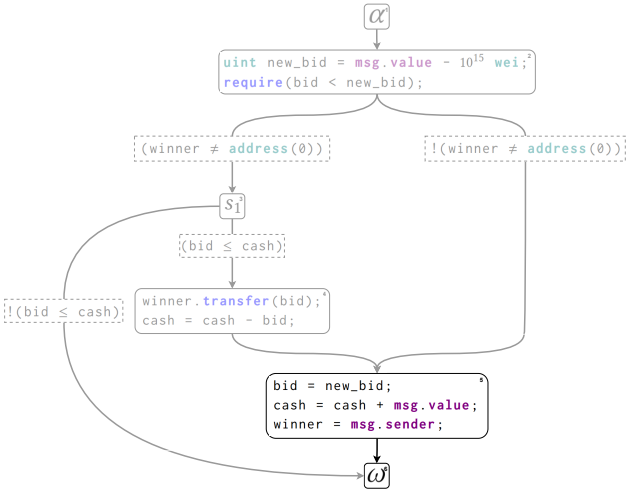
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$

$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Example: an Auction in Solidity

```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```



Example: an Auction in Solidity

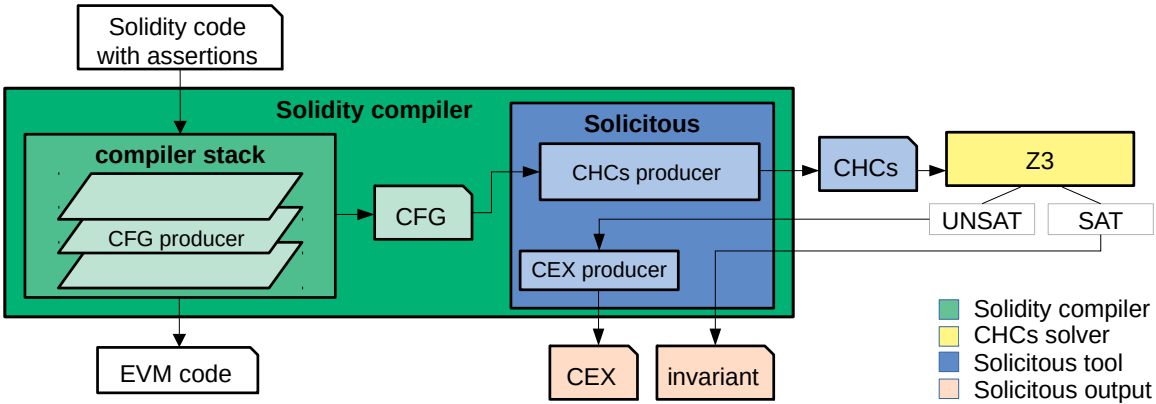
```
1  contract Auction {
2      uint bid = 0;
3      uint cash = 0;
4      address payable winner = address(0);
5
6      function offer() public payable {
7          uint new_bid = msg.value − 10^15 wei;
8          require(bid < new_bid);
9          if (winner ≠ address(0)) {
10             assert(bid ≤ cash);
11             winner.transfer(bid);
12             cash = cash − bid;
13         }
14         bid = new_bid;
15         cash = cash + msg.value;
16         winner = msg.sender;
17     }
18 }
```

$$\mathcal{P}_f^u(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I'}) \leftarrow \mathcal{P}_f^v(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{I}) \wedge \mathrm{SSA}_{\lambda_v}(\boldsymbol{I}, \boldsymbol{I'}) \wedge \mathrm{SSA}_{\mu_e}(\boldsymbol{I'})$$

$$\mathcal{P}_o^2 \leftarrow \mathcal{P}_o^1$$
$$\mathcal{P}_o^3 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge I_w \neq 0$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^2 \wedge I_{nb} = I_v - 10^{15} \wedge I_b < I_{nb} \wedge \neg(I_w \neq 0)$$
$$\mathcal{P}_o^4 \leftarrow \mathcal{P}_o^3 \wedge I_b \leq I_c$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^3 \wedge \neg(I_b \leq I_c) \wedge I_{\tilde{r}} = 3$$
$$\mathcal{P}_o^5 \leftarrow \mathcal{P}_o^4 \wedge I'_c = I_c - I_b$$
$$\mathcal{P}_o^6 \leftarrow \mathcal{P}_o^5 \wedge I'_b = I_{nb} \wedge I'_c = I_c + I_v \wedge I'_w = I_s$$

$$\mathcal{P}_o^n(b, c, w, s, v, I_b, I_c, I_w, I_s, I_v, I_{nb}, I_{\tilde{r}}),\ n \in \{1..6\}$$

Modelling Framework



Solicitous (Solidity Contract verification using constrained Horn clauses)

- Assertion failures lead to reverts, which are modelled as error predicates
- Queries are made to the solver for the reachability of error predicates

Evaluation

- Evaluation with 22446 real-world contracts
  - Solidity versions 0.5 to 0.8

	SOLICITOUS	SOLC-VERIFY	VERISOL	MYTHRIL
Instances	**22446**	13310	12402	**22446**
Safe	**6651**	103	201	0
Unsafe	8	0	9	**21**
Unknown	2970	5601	230	728
Timeout	9058	3163	4032	19511
Error	3759	4443	7930	2186
Verified	∼**29%**	∼0.007%	∼0.01%	∼0.0009%

Evaluation

- Evaluation with 22446 real-world contracts
  - Solidity versions 0.5 to 0.8

	SOLICITOUS	SOLC-VERIFY	VERISOL	MYTHRIL
Instances	**22446**	13310	12402	**22446**
Safe	**6651**	103	201	0
Unsafe	8	0	9	**21**
Unknown	2970	5601	230	728
Timeout	9058	3163	4032	19511
Error	3759	4443	7930	2186
Verified	∼**29%**	∼0.007%	∼0.01%	∼0.0009%

Evaluation

- Evaluation with 22446 real-world contracts
  - Solidity versions 0.5 to 0.8

	SOLICITOUS	SOLC-VERIFY	VERISOL	MYTHRIL
Instances	**22446**	13310	12402	**22446**
Safe	**6651**	103	201	0
Unsafe	8	0	9	**21**
Unknown	2970	5601	230	728
Timeout	9058	3163	4032	19511
Error	3759	4443	7930	2186
Verified	∼**29%**	∼0.007%	∼0.01%	∼0.0009%

Evaluation

- Evaluation with 22446 real-world contracts
  - Solidity versions 0.5 to 0.8

	SOLICITOUS	SOLC-VERIFY	VERISOL	MYTHRIL
Instances	**22446**	13310	12402	**22446**
Safe	**6651**	103	201	0
Unsafe	8	0	9	**21**
Unknown	2970	5601	230	728
Timeout	9058	3163	4032	19511
Error	3759	4443	7930	2186
Verified	∼**29%**	∼0.007%	∼0.01%	∼0.0009%

Evaluation

- Evaluation with 22446 real-world contracts
  - Solidity versions 0.5 to 0.8

	SOLICITOUS	SOLC-VERIFY	VERISOL	MYTHRIL
Instances	**22446**	13310	12402	**22446**
Safe	**6651**	103	201	0
Unsafe	8	0	9	**21**
Unknown	2970	5601	230	728
Timeout	9058	3163	4032	19511
Error	3759	4443	7930	2186
Verified	∼**29%**	∼0.007%	∼0.01%	∼0.0009%

Evaluation

- Evaluation with 22446 real-world contracts
  - Solidity versions 0.5 to 0.8

	SOLICITOUS	SOLC-VERIFY	VERISOL	MYTHRIL
Instances	**22446**	13310	12402	**22446**
Safe	**6651**	103	201	0
Unsafe	8	0	9	**21**
Unknown	2970	5601	230	728
Timeout	9058	3163	4032	19511
Error	3759	4443	7930	2186
Verified	∼**29%**	∼0.007%	∼0.01%	∼0.0009%

Evaluation

- Evaluation with 22446 real-world contracts
  - Solidity versions 0.5 to 0.8

	SOLICITOUS	SOLC-VERIFY	VERISOL	MYTHRIL
Instances	**22446**	13310	12402	**22446**
Safe	**6651**	103	201	0
Unsafe	8	0	9	**21**
Unknown	2970	5601	230	728
Timeout	9058	3163	4032	19511
Error	3759	4443	7930	2186
Verified	∼**29%**	∼0.007%	∼0.01%	∼0.0009%

Evaluation

- Evaluation with 22446 real-world contracts
  - Solidity versions 0.5 to 0.8

- Verified $= \dfrac{\text{safe} + \text{unsafe}}{\text{num. of instances}}$

	SOLICITOUS	SOLC-VERIFY	VERISOL	MYTHRIL
Instances	**22446**	13310	12402	**22446**
Safe	**6651**	103	201	0
Unsafe	8	0	9	**21**
Unknown	2970	5601	230	728
Timeout	9058	3163	4032	19511
Error	3759	4443	7930	2186
Verified	∼**29%**	∼0.007%	∼0.01%	∼0.0009%

Summary

- Direct modelling of smart contracts into CHCs
- Solicitous implemented inside the Solidity compiler

Summary

- Direct modelling of smart contracts into CHCs
- Solicitous implemented inside the Solidity compiler

Future Work

- Smart contract verification
  - Enhancement of the modelling to account for gas consumption
  - Instantiation of the approach to languages other than Solidity
  - Production of correctness certificates as witnesses of safe results
- Tool improvements
  - Support for additional Solidity features
  - Integration with different back-end solvers

Summary

- Direct modelling of smart contracts into CHCs
- Solicitous implemented inside the Solidity compiler

Future Work

- Smart contract verification
  - Enhancement of the modelling to account for gas consumption
  - Instantiation of the approach to languages other than Solidity
  - Production of correctness certificates as witnesses of safe results
- Tool improvements
  - Support for additional Solidity features
  - Integration with different back-end solvers

`verify.inf.usi.ch/research/fvsc`