Blockchain Superoptimizer

Julian Nagele Maria A. Schett

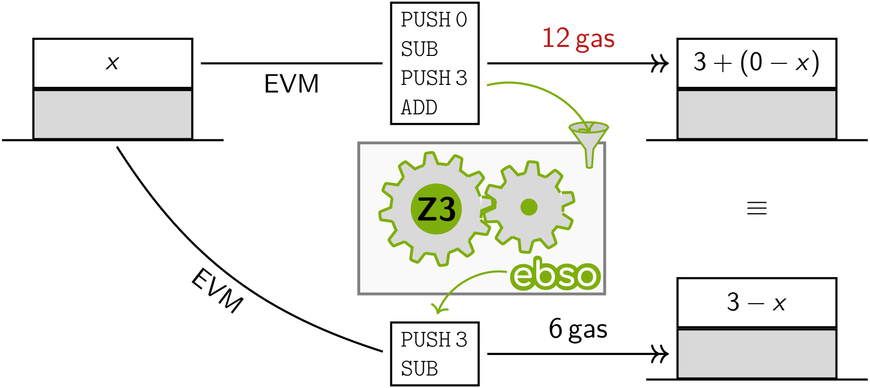
QMUL

UCL

February 8 2019

S-REPLS 11

Overview



- Ethereum smart contracts are executed as bytecode on the Ethereum Virtual Machine (EVM)
- every instruction executed on the EVM has a cost: gas
- our tool **ebso** superoptimizes EVM bytecode

Demo

```
$ ./ebso -translation-validation 256 -direct "600003600301"
Optimized PUSH 0 SUB PUSH 3 ADD to
PUSH 3 SUB
Saved 6 gas, translation validation successful,
this instruction sequence is optimal.
```

Superoptimization

given source program p_s and cost function C find target program p_t that

- correctly implements p_s
- has minimal cost

Ethereum

- formal semantics available
- EVM gas provides clear cost model
- large data set readily available for evaluation
- programs are deployed once $\Rightarrow$ long compilation time acceptable

ebso

- does binary recompilation of EVM bytecode
- in two flavors: BASIC and UNBOUNDED superoptimization

Basic Superoptimization

function BasicSo(p_s, C)
  $n \leftarrow 0$
  while true **do**
    for all $p_t \in \{p \mid C(p) = n \wedge \text{checks}(p)\}$ **do**
      $\chi \leftarrow$ EncBso(p_s, p_t)
      if Satisfiable(χ) **then**
        $m \leftarrow$ GetModel(χ)
        $p_t \leftarrow$ DecBso(m)
        return p_t
    $n \leftarrow n + 1$

- search through candidate instruction sequences and call solver to check correctness

Unbounded Superoptimization

function $\textsc{UnboundedSo}(p_s, C)$
    $p_t \leftarrow p_s$
    $\chi \leftarrow \textsc{EncUso}(p_t) \land \textsc{Bound}(p_t, C)$
    while $\textsc{Satisfiable}(\chi)$ **do**
        $m \leftarrow \textsc{GetModel}(\chi)$
        $p_t \leftarrow \textsc{DecUso}(m)$
        $\chi \leftarrow \chi \land \textsc{Bound}(p_t, C)$
    return p_t

- shifts the search into the solver
- can stop early with possibly non-optimal solution

Encoding

Satisfiability Modulo Theories

- first-order logic with background theories (bit vectors, integers, uninterpreted functions, arrays, ...)
- powerful off-the-shelf solvers available

Ingredients

- state, i.e., stack, used gas, ...
- $\twoheadrightarrow^{p}_{\mathsf{EVM}}$, i.e., operational semantics of EVM
- $\equiv$, i.e, equality on states
- for UnboundedSo: encoding of search space

State

state $\sigma = \langle \mathrm{st}, \mathrm{c}, \mathrm{g} \rangle$ consists of

- function $\mathrm{st}(\vec{x}, j, n)$: n-th word on stack after j instructions on input $\vec{x}$
- function $\mathrm{c}(j)$: number of words on stack after j instructions
- function $\mathrm{g}(j)$: amount of gas consumed by first j instructions.

Example

symbolically executing `PUSH 0 SUB PUSH 3 ADD` yields

$$\mathrm{g}(0) = 0 \qquad \mathrm{c}(0) = 1 \qquad \mathrm{st}(x, 0, 0) = x$$

Equality

equality of states after j_1 and j_2 $\rightarrow_{\mathrm{EVM}}$ steps

$$\begin{aligned}\epsilon(\vec{x}, \sigma_1, \sigma_2, j_1, j_2) \equiv \mathrm{c}_{\sigma_1}(j_1) &= \mathrm{c}_{\sigma_2}(j_2) \\ \wedge \forall n < \mathrm{c}_{\sigma_1}(j_1).\, \mathrm{st}_{\sigma_1}(\vec{x}, j_1, n) &= \mathrm{st}_{\sigma_2}(\vec{x}, j_2, n)\end{aligned}$$

Instructions

semantics of instruction i given by

$$
\begin{aligned}
\tau(i,\vec{x},\sigma,j) &\equiv \tau_{\mathsf{g}}(i,\sigma,j) \wedge \tau_{\mathsf{c}}(i,\sigma,j) \wedge \tau_{\mathsf{pres}}(i,\vec{x},\sigma,j) \wedge \tau_{\mathsf{st}}(i,\vec{x},\sigma,j) \\
\tau_{\mathsf{g}}(i,\sigma,j) &\equiv \mathsf{g}_\sigma(j+1) = \mathsf{g}_\sigma(j) + C(i) \\
\tau_{\mathsf{c}}(i,\sigma,j) &\equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(i) - \delta(i) \\
\tau_{\mathsf{pres}}(i,\vec{x},\sigma,j) &\equiv \forall n < \mathsf{c}_\sigma(j) - \delta(i).\, \mathsf{st}_\sigma(\vec{x},j,n) = \mathsf{st}_\sigma(\vec{x},j+1,n) \\
\tau_{\mathsf{st}}(\mathtt{ADD},\vec{x},\sigma,j) &\equiv \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j+1)-1) \\
&= \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j)-1) +_{bv} \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j)-2) \\
\tau_{\mathsf{st}}(\mathtt{SWAP2},\vec{x},\sigma,j) &\equiv \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j+1)-1) = \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j)-3) \\
&\wedge \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j+1)-2) = \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j)-2) \\
&\wedge \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j+1)-3) = \mathsf{st}_\sigma(\vec{x},j,\mathsf{c}_\sigma(j)-1)
\end{aligned}
$$

for program $p = i_0, \ldots, i_n$ we define

$$
\tau(p,\vec{x},\sigma) \equiv \bigwedge_{0 \leqslant j \leqslant n} \tau(i_j,\vec{x},\sigma,j)
$$

Superoptimization

Basic

$$\textsc{EncBso}(p_s, p_t) \equiv \forall \vec{x}.\, \tau(p_s, \vec{x}, \sigma) \wedge \tau(p_t, \vec{x}, \sigma') \wedge \epsilon(\vec{x}, \sigma, \sigma', 0, 0) \wedge \epsilon(\vec{x}, \sigma, \sigma', |p_s|, |p_t|)$$

Unbounded

$$\textsc{EncUso}(p) = \forall \vec{x}.\, \tau(p, \vec{x}, \sigma) \wedge \epsilon(\vec{x}, \sigma, \sigma', 0, 0) \wedge \epsilon(\vec{x}, \sigma, \sigma', |p|, n) \wedge \forall j < n.\, \bigwedge_{i \in \mathcal{I}} \mathsf{instr}(j) = i \longrightarrow \tau(i, \vec{x}, \sigma', j) \wedge \bigvee_{i \in \mathcal{I}} \mathsf{instr}(j) = i$$

Templates

- represent subsets of instructions using uninterpreted functions
- for immediate arguments of PUSH use function a(j) that maps a program location j to word
- reconstruct actual value from model found by solver

Implementation

- available at `github.com/juliannagele/ebso`
- implemented in OCaml, using Z3 as SMT solver
- ~1.6 kloc

Translation Validation

- large word size of EVM (256 bit) led to scalability problems
- solution: find model for small word-size and validate for full size:

$$\begin{aligned}\textsc{TransVal}(p_s, p_t) = \exists \vec{x}.\, \tau(p_s, \vec{x}, \sigma) \wedge \tau(p_t, \vec{x}, \sigma') \\ \wedge\, \epsilon(\vec{x}, \sigma, \sigma', 0, 0) \wedge \neg\epsilon(\vec{x}, \sigma, \sigma', |p_s|, |p_t|)\end{aligned}$$

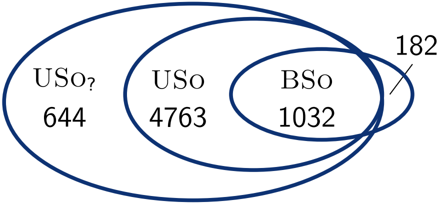
BasicSo vs. UnboundedSo

- 11 467 Solidity (PL for smart contracts) files from EtherScan platform resulting in 51 146 contracts and 89 004 encodable sequences of instructions
- 15 min timeout on single core at 2.40 GHz with 1 GiB RAM

	BasicSo	UnboundedSo
optimized	1214	5407
proved optimal	2135	17 946
gas saved	6451	29 973
weighted gas saved	927 736	2 201 784
transl. val. failed	n/a	4205

Overlap of BasicSo and UnboundedSo

- overlap between bounded (BSo) and unbounded (USo) superoptimization
- ? indicates USo stopped prematurely



ebso vs. `solc`

- **ebso** on code generated by Solidity compiler with `--optimize` finds

optimized	gas saved	weighted gas saved
2609	9325	2 259 871

⇒ optimization potential due to immature compiler

Conclusion

Summary

- **ebso** optimizes EVM bytecode
- unbounded superoptimization shifts search into solver
- relying on search heuristics of solver allows low effort implementation

Future Work

- extend encoding with more EVM features, e.g. storage
- go beyond straight-line code with control flow analysis
- generalize optimization patterns and build into rewrite engine
- extract SMT benchmarks
- develop tactics and strategies to guide SMT solver