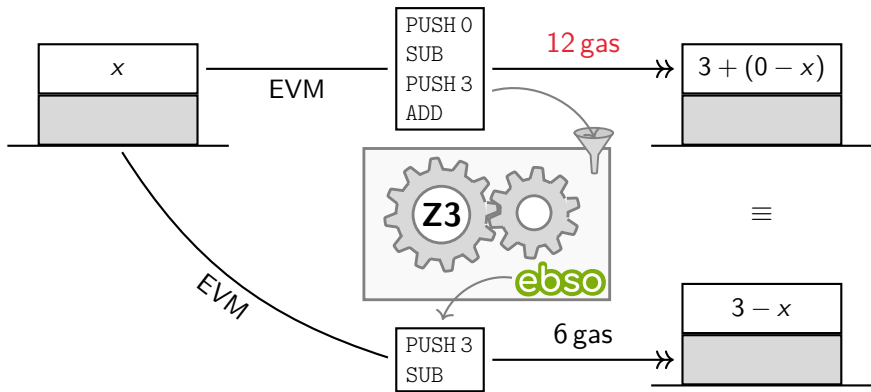


Blockchain Superoptimizer

Julian Nagele **Maria A Schett**

Overview

- ▶ Ethereum smart contracts are executed as bytecode on the Ethereum Virtual Machine (EVM)



Goals

- ▶ our tool **ebso** superoptimizes EVM bytecode

Ethereum

- ▶ EVM gas provides **clear cost model**
- ▶ formal semantics [Yellow Paper, 2018]
- ▶ \exists data sets for evaluation

Today

- ▶ feedback on ideas for future work (post-proceedings)

Superoptimization

Superoptimization

- ▶ **given:** source program s & cost function C
- ▶ **find:** target program t that
 1. has **minimal** cost $C(t)$
 2. correctly implements s
- ▶ **using:** a constraint solver

program $l_1 l_2 \cdots l_n$
Instr $l \in \{\text{ADD, SUB, SLT, PUSH } w, \text{DUP, SWAP, } \dots\}$
 $w \in \text{Word} = \{0, 1\}^{256}$
Pos PUSH₀₁ SUB₂ PUSH₃₃ ADD₄

Satisfiability Modulo Theories (SMT) Solver

- ▶ first-order logic with background theories
 - ★ bit vectors, integers, uninterpreted functions

Uninterpreted Functions

functions a, f, f' s.t.

- ▶ $a(1) = 3$ $\lambda x.3$
- ▶ $\forall \ell < 5. f(\ell) = f'(\ell)$ $\lambda x.3$
- ▶ $f(1) = a(1) \wedge f'(1) = 42$ UNSAT

Superoptimization [Massalin 1987]

- ▶ enumerate all possible candidate programs t in increasing cost

$\exists \vec{x}. \text{ to distinguish } s \ \& \ t?$

Templates [Gulwani et al. 2011]

- ▶ PUSH w where w is a 256 bit word $\implies 2^{256}$ candidates

$\exists (a : \text{Pos} \rightarrow \text{Word}) \forall \vec{x}. t \text{ implements } s?$

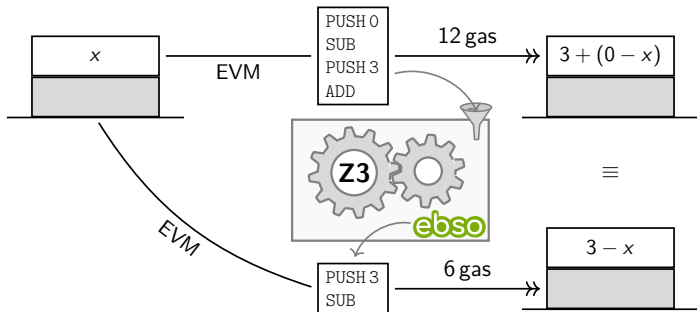
Unbounded Superoptimization [Jangda&Yorsh 2017]

- ▶ shift search in solver

$\exists (t : \text{Pos} \rightarrow \text{Instr}) \forall \vec{x}. t \text{ implements } s \ \& \ C(t) < C(s)?$

SMT Encoding

Ingredients



- state** stack, used gas, stack counter ...
- \twoheadrightarrow operational semantics of EVM
- \equiv equality on states

State & $\rightarrow\rightarrow$

state $\sigma = \langle \text{stk}, c, g \rangle$ consists of

- ▶ $\text{stk}(j, \ell)$: ℓ -th word on **stack** after j instructions (on input \vec{x})
- ▶ $g(j)$: **gas** after j instructions
- ▶ **stack counter** $\sim \top_j$ for top of stack after j instructions

$\sigma_j \xrightarrow{\iota} \sigma_{j+1}$ semantics of $\iota \in \text{Instr}$

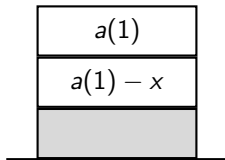
- ▶ $\text{stk}(j+1, \top_{j+1}) = \text{stk}(j, \top_j) +_{bv} \text{stk}(j, \top_j - 1)$ for $\iota = \text{ADD}$
& “preserverance of stack” \wedge
- ▶ $g(j+1) = g(j) + C(\iota) \wedge$
- ▶ $\top_{j+1} = \top_j + \alpha(\iota) - \delta(\iota)$

for program $p = \iota_0, \dots, \iota_n$, define $\sigma_0 \xrightarrow{p} \sigma_{|p|}$ as $\bigwedge_{0 \leq j \leq n} \sigma_j \xrightarrow{\iota_j} \sigma_{j+1}$

Example

PUSH $a(1)$ ₁ SUB₂

$j = 0 :$	$\text{stk}(j, 1) = x$		$g(j) = 0$	$\top_j = 1$
$j = 1 :$	$\text{stk}(j, 1) = x$	$\text{stk}(j, 2) = a(j)$	$g(j) = 3$	$\top_j = 2$
$j = 2 :$	$\text{stk}(j, 1) = a(1) - x$		$g(j) = 6$	$\top_j = 1$



Equality

$\sigma_j \equiv \sigma'_{j'}$ states σ and σ' after j and j' instructions

▶ $\sigma.\top_j = \sigma'.\top_{j'} \wedge$

$$\forall l < \top_j. \sigma.\text{stk}(j, l) = \sigma'.\text{stk}(j', l)$$

▶ not equal **gas**



Basic Superoptimization

$\exists (a : \text{Pos} \rightarrow \text{Word}) \forall \vec{x} \text{ s.t. } t \text{ implements } s?$

$$\exists a \forall \vec{x}. \sigma_0 \xrightarrow{s} \sigma \wedge \sigma_0 \xrightarrow{t} \sigma' \wedge \sigma_{|s|} \equiv \sigma'_{|t|}$$

Unbounded Superoptimization

$\exists (t : \text{Pos} \rightarrow \text{Instr}) \forall \vec{x} \text{ s.t. } t \text{ implements } s \ \& \ C(t) < C(s)?$

$$\begin{aligned} \exists n \exists t \forall \vec{x}. \sigma_0 \xrightarrow{s} \sigma \wedge \sigma_0 \equiv \sigma'_0 \wedge \sigma_{|s|} \equiv \sigma'_n \wedge \sigma.g(|s|) > \sigma'.g(n) \wedge \\ \forall j < n. \bigwedge_{\iota \in \text{Instr}} t(j) = \iota \implies \sigma'_j \xrightarrow{\iota_j} \sigma'_{j+1} \wedge \bigvee_{\iota \in \text{Instr}} t(j) = \iota \wedge \end{aligned}$$

Implementation

Implementation

- ▶ available at

`github.com/juliannagele/ebso`

- ▶ implemented in OCaml
- ▶ ~1.6 kloc (encoding 1 kloc), 635 tests



- ▶ using Z3 as SMT solver

Z3

Interface

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


Translation Validation

- ▶ **given:** large word size of EVM 256 bit \implies scalability problems
- ▶ **solution:** find t for small word size & validate for 256 bit

$\exists \vec{x}$ s.t. t does not implement s ?

$$\exists \vec{x}. \sigma_0 \xrightarrow{s} \sigma \wedge \sigma_0 \xrightarrow{t} \sigma' \wedge \neg(\sigma|_s \equiv \sigma'|_t)$$

- ▶ for word size 2 bit
 - ★ PUSH 0 SUB PUSH 3 ADD optimizes to NOT
 - ★ because the binary representation of 3 is 11.

Evaluation

1. “Optimize the Optimized”

- ★ Gas Golfing Contest: 199 Solidity contracts
- ★ \implies 2743 ebso blocks

2. “Basic vs. Unbounded”

- ★ bytecode of 2500 most called contracts from Ethereum Blockchain
 - ★ \implies 61217 ebso blocks
- ▶ 60 min/15 min time-out on 1 core at 2.40 GHz with 1 GiB RAM
 - ▶ validation with pseudo-random input on go-ethereum EVM

Optimize The Optimized (1)

$s = \text{CALLVALUE DUP ISZERO PUSH 81}$

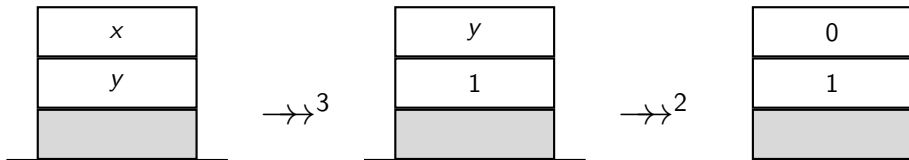
to $t = \text{CALLVALUE CALLVALUE ISZERO PUSH 81}$

- ▶ $C(\text{DUP}) = 3 \text{ gas} > C(\text{CALLVALUE}) = 2 \text{ gas}$

Optimize The Optimized (2)

$s = \text{POP PUSH 1 SWAP POP PUSH 0}$

to $t = \text{SLT DUP EQ PUSH 0}$



Basic vs. Unbounded (1)

- ▶ in our setting

Unbounded > Basic

Basic vs. Unbounded (2)

	Unbounded	
	#	%
optimized ¹	943	1.54 %
time-out	56 392	92.12 %
proved optimal	3882	6.34 %

¹shown optimal: 393

Conclusion

Future Work (1)

- ▶ performance/time-outs

Feedback

1. tune encoding
 - ★ not use theory of integers *and* bit vectors, remove storage constraints, ...
2. use domain knowledge
 - ★ exclude rare instructions from Instr, ...
3. tune solver
 - ★ strategy, different solvers

Future Work (2)

- ▶ generate peephole optimization rules
- ▶ SuperOptimization-based Rule Generator **sorg**²



- ▶ $s = \text{CALLVALUE DUP ISZERO PUSH 81}$
to $t = \text{CALLVALUE CALLVALUE ISZERO PUSH 81}$
- ▶ **rule:** $\text{CALLVALUE DUP} \rightarrow \text{CALLVALUE CALLVALUE}$
- ▶ found **397** distinct rules from 943 optimized ebso blocks

² available at github.com/mariaschett/sorg

Thank you & questions?



available at github.com/juliannagele/ebso

[mail@]jnagele.net

[mail@]maria-a-schett.net

Bibliography



H. Massalin

Superoptimizer: A Look at the Smallest Program

ASPLOS II, 1987



S. Gulwani, S. Jha, A. Tiwari, and R. Venkatesan

Synthesis of Loop-free Programs

Proc. PLDI 2011



A. Jangda and G. Yorsh

Unbounded Superoptimization

Proc. Onward! 2017



Ethereum: A Secure Decentralised Generalised Transaction Ledger

Technical Report Byzantium Version e94ebda