Blockchain Superoptimizer

Julian Nagele **Maria A. Schett**

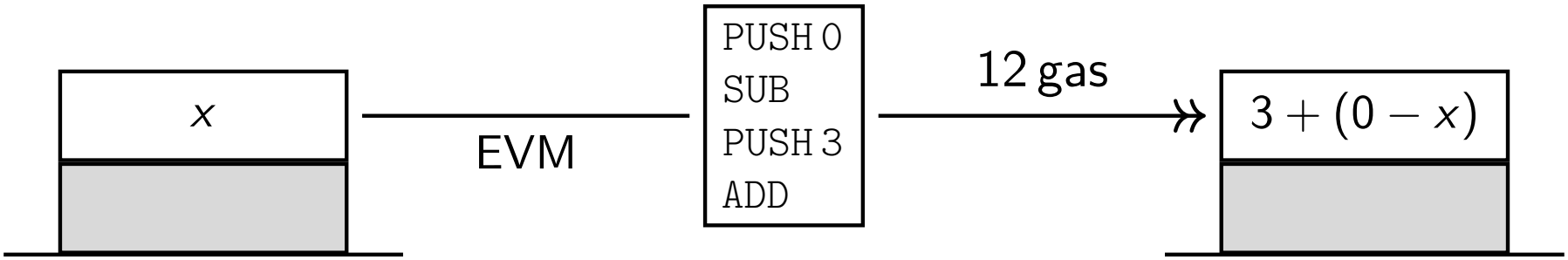
April 10 2019 @ IMDEA

Overview

```
PUSH 0
SUB
PUSH 3
ADD
```

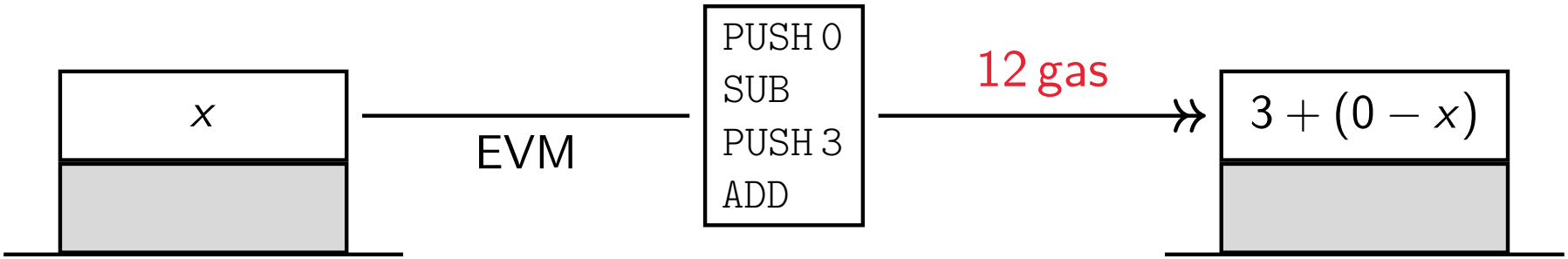
- Ethereum smart contracts are executed as bytecode on the Ethereum Virtual Machine (EVM)

Overview

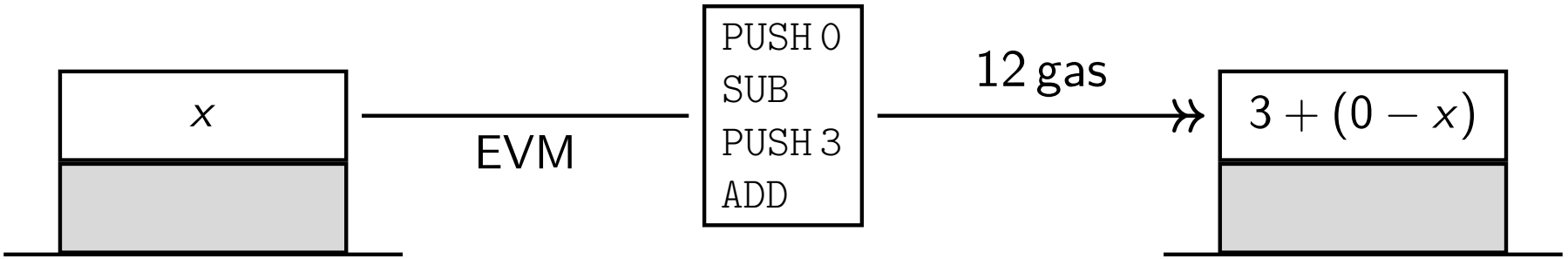


- Ethereum smart contracts are executed as bytecode on the Ethereum Virtual Machine (EVM)

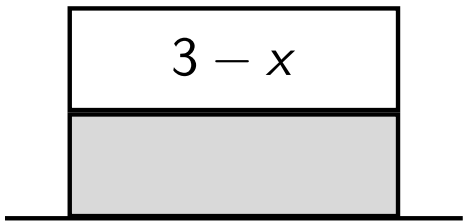
Overview



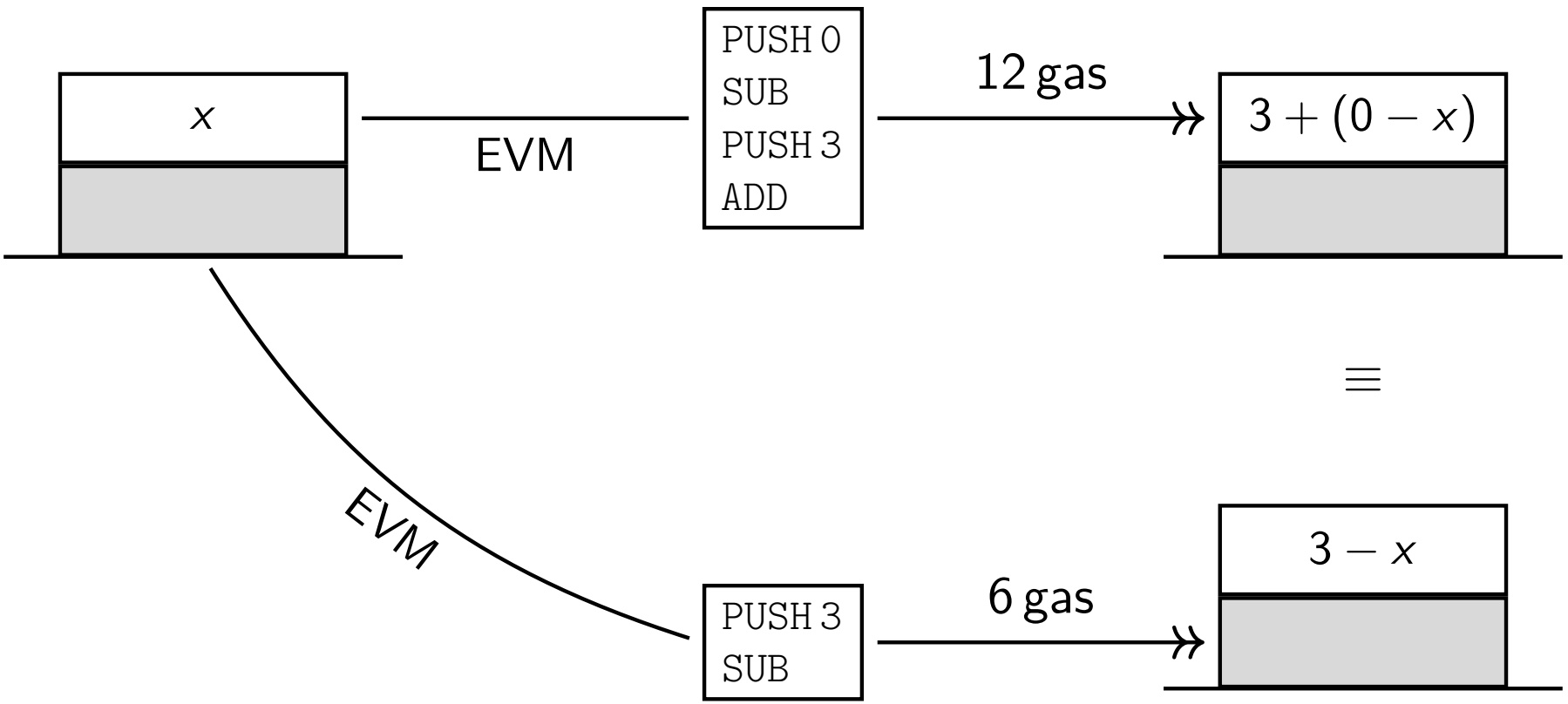
Overview



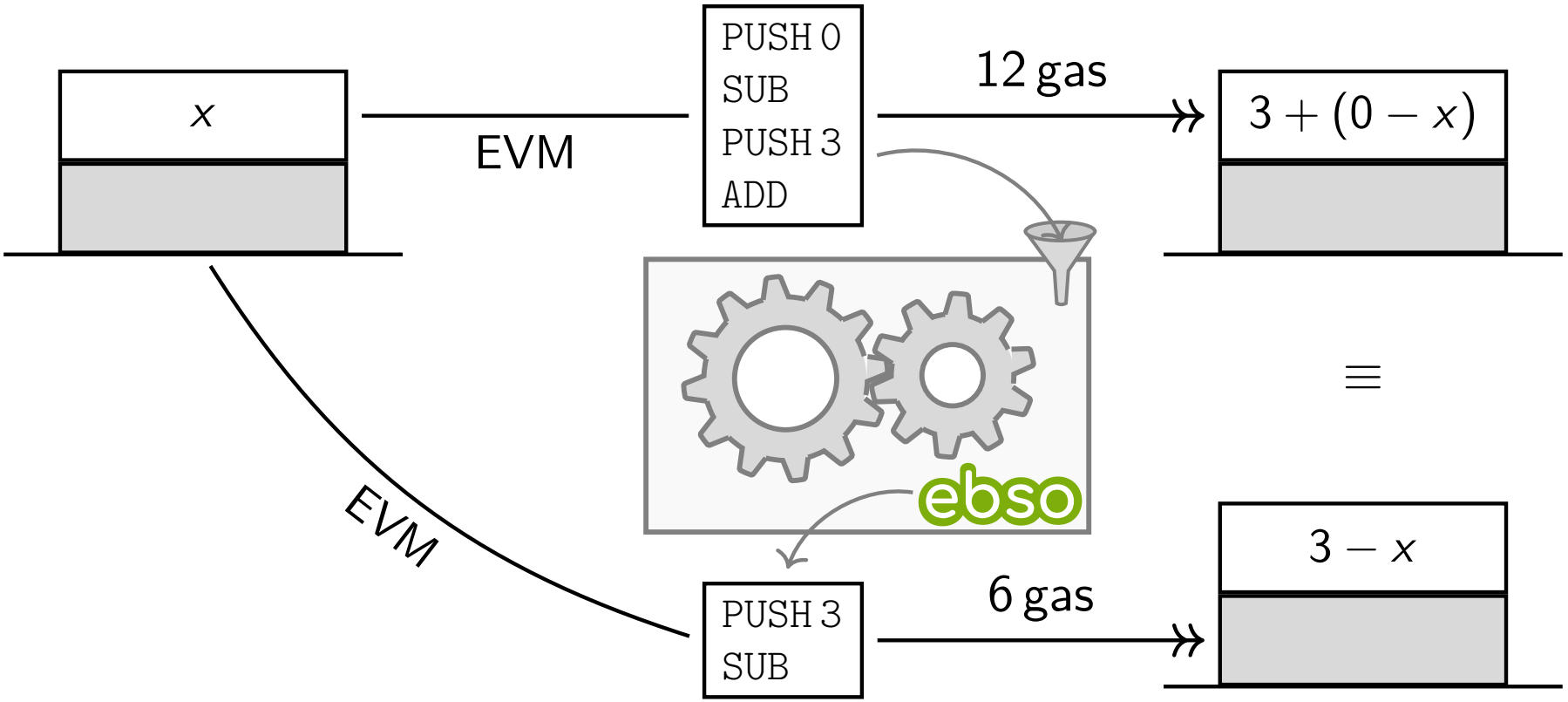
$\equiv$



Overview

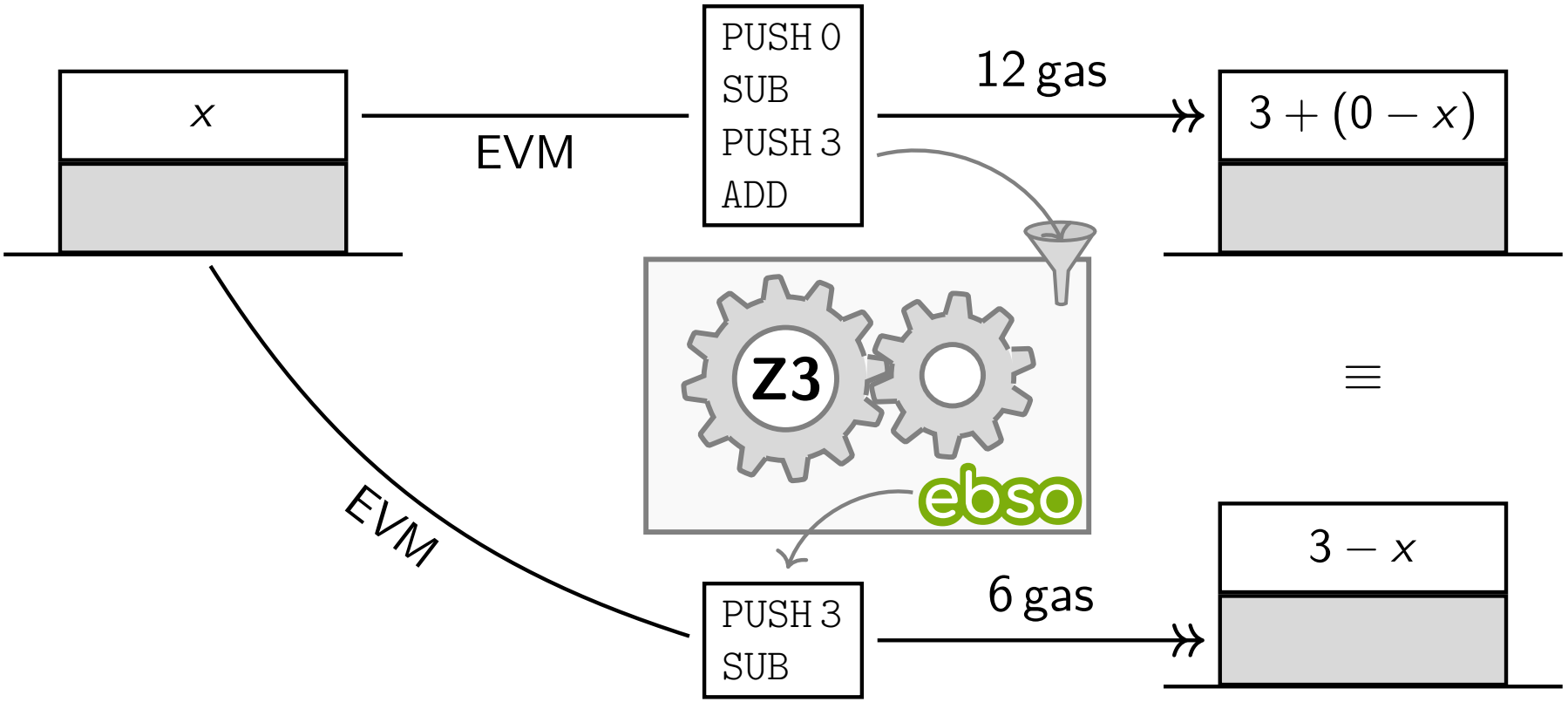

x
EVM
PUSH 0
SUB
PUSH 3
ADD
12 gas
$3 + (0 - x)$
$\equiv$
EVM
PUSH 3
SUB
6 gas
$3 - x$

Overview



- our tool **ebso** superoptimizes EVM bytecode

Overview


x
EVM
PUSH 0
SUB
PUSH 3
ADD
12 gas
3 + (0 − x)
Z3
ebso
≡
EVM
PUSH 3
SUB
6 gas
3 − x

Interface

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

Superoptimization

Superoptimization [Massalin 1987]

- **given:** source program p_s & cost function C
- **find:** target program p_t that
  1. correctly implements p_s
  2. has minimal cost
- **flavors:** basic and unbounded superoptimization

Superoptimization [Massalin 1987]

- **given:** source program p_s & cost function C
- **find:** target program p_t that
  1. correctly implements p_s
  2. has minimal cost
- **flavors:** basic and unbounded superoptimization

Ethereum

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

Basic Superoptimization (1)

```
1: function BasicSo(p_s, C)
2:   n ← 0
3:   while true do
4:     for all p_t ∈ {p | C(p) = n} do
5:       χ ← EncBso(p_s, p_t)
6:       if not Satisfiable(χ) then
7:         return p_t
8:     n ← n + 1
```

EncBso: $\exists \vec{x}$ to distinguish p_s & p_t?

Templates [Gulwani et al. 2011]

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

Templates [Gulwani et al. 2011]

- PUSH w where w is a 256 bit word $\implies$ 2^{256} candidates
- **idea:** function (a : position $\to$ word): "argument of instruction at position"

Templates [Gulwani et al. 2011]

- PUSH w where w is a 256 bit word $\implies$ 2^{256} candidates
- **idea:** function (a : position $\to$ word): "argument of instruction at position"

Example

PUSH 0 SUB PUSH 3 ADD

$$a(0) = 0 \qquad a(2) = 3 \qquad a(j) = \_$$

Templates [Gulwani et al. 2011]

- PUSH w where w is a 256 bit word $\implies$ 2^{256} candidates
- **idea:** function (a : position $\to$ word): "argument of instruction at position"

Example

```
PUSH 0 SUB PUSH 3 ADD
```

$$a(0) = 0 \qquad a(2) = 3 \qquad a(j) = \_$$

EncBso: $\exists(a : \text{position} \to \text{word})\ \forall \vec{x}$ s.t. p_t implements p_s?

Basic Superoptimization (2)

EncBso: $\exists(a : \text{position} \to \text{word})\ \forall \vec{x}$ s.t. p_t implements p_s?

Basic Superoptimization (2)

EncBso: $\exists(a : \text{position} \to \text{word})\ \forall \vec{x}$ s.t. p_t implements p_s?

```
1:  function BasicSo(p_s, C)
2:    n ← 0
3:    while true do
4:      for all p_t ∈ {p | C(p) = n} do
5:        χ ← EncBso(p_s, p_t)
6:        if Satisfiable(χ) then
7:          m ← GetModel(χ)
8:          p_t ← DecBso(m)
9:          return p_t
10:     n ← n + 1
```

Unbounded Superoptimization [Jangda&Yorsh 2017]

EncUso: $\exists(\text{instr} : \text{position} \rightarrow \text{instruction})\ \forall \vec{x}$ s.t. p_t implements p_s?

Unbounded Superoptimization [Jangda&Yorsh 2017]

EncUso: $\exists$(instr : position $\rightarrow$ instruction) $\forall \vec{x}$ s.t. p_t implements p_s?

1: **function** UnboundedSo(p_s, C)
2: $p_t \leftarrow p_s$
3: $\chi \leftarrow \text{EncUso}(p_t) \wedge \text{Bound}(p_t, C)$
4: **while** Satisfiable(χ) **do**
5: $m \leftarrow \text{GetModel}(\chi)$
6: $p_t \leftarrow \text{DecUso}(m)$
7: $\chi \leftarrow \chi \wedge \text{Bound}(p_t, C)$
8: **return** p_t

Unbounded Superoptimization [Jangda&Yorsh 2017]

EncUso: $\exists$(instr : position $\rightarrow$ instruction) $\forall \vec{x}$ s.t. p_t implements p_s?

1: **function** UnboundedSo(p_s, C)
2: $p_t \leftarrow p_s$
3: $\chi \leftarrow \text{EncUso}(p_t) \wedge \text{Bound}(p_t, C)$
4: **while** Satisfiable(χ) **do**
5: $m \leftarrow \text{GetModel}(\chi)$
6: $p_t \leftarrow \text{DecUso}(m)$
7: $\chi \leftarrow \chi \wedge \text{Bound}(p_t, C)$
8: **return** p_t

- can stop early with non-optimal solution

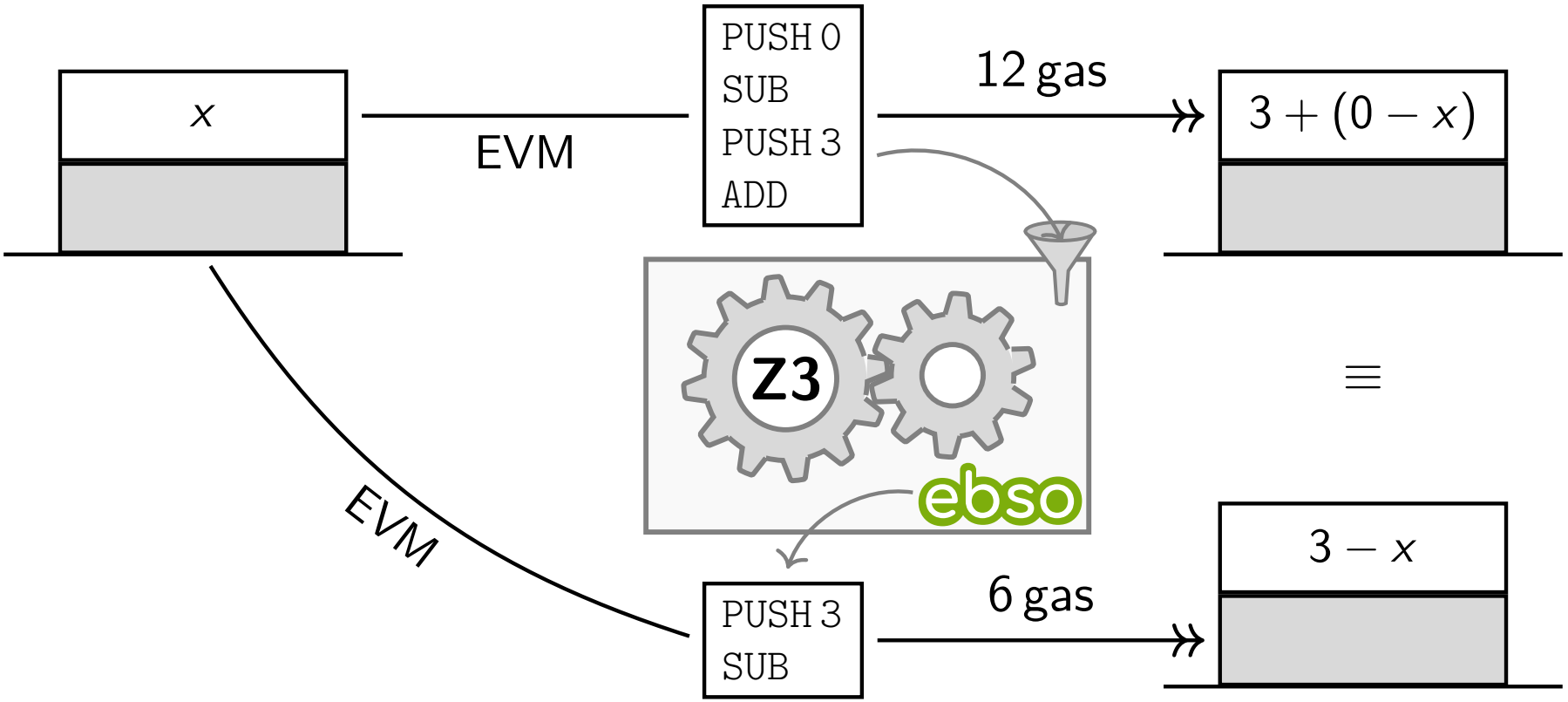
SMT Encoding

Encoding

Satisfiability Modulo Theories

- first-order logic with background theories
  - ⋆ bit vectors, integers, uninterpreted functions, arrays, ...
- ∃ powerful off-the-shelf solvers
- ■

Overview


x
EVM
PUSH 0
SUB
PUSH 3
ADD
12 gas
3 + (0 − x)
Z3
ebso
≡
EVM
PUSH 3
SUB
6 gas
3 − x

Ingredients

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

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.

State

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

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

$\mathsf{st}(\vec{x}, j, \mathsf{c}(j) - 1)$ returns top of stack $\sim \top$

State

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

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

Example

`PUSH 41 PUSH 1 ADD`

$$j = 2: \qquad \mathsf{st}(j, 0) = 41 \qquad \mathsf{st}(j, 1) = 1 \qquad \mathsf{c}(j) = 2 \qquad \mathsf{g}(j) = 6$$

Instructions

semantics of instruction ι

$$\tau_{\mathsf{st}}(\texttt{ADD}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1)$$

Instructions

semantics of instruction ι

$$\begin{aligned}
\tau_{\mathsf{st}}(\texttt{ADD}, \sigma, j) &\equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1) \\
\tau_{\mathsf{st}}(\texttt{SWAP2}, \sigma, j) &\equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top - 2) \\
&\quad \wedge \mathsf{st}_\sigma(j+1, \top - 1) = \mathsf{st}_\sigma(j, \top - 1) \\
&\quad \wedge \mathsf{st}_\sigma(j+1, \top - 2) = \mathsf{st}_\sigma(j, \top)
\end{aligned}$$

Instructions

semantics of instruction ι

$$\tau_{\mathsf{st}}(\texttt{ADD}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1)$$
$$\tau_{\mathsf{c}}(\iota, \sigma, j) \equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(\iota) - \delta(\iota)$$

Instructions

number of elements added (α) to and deleted (δ) from stack

$$\tau_{\mathsf{st}}(\texttt{ADD}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1)$$

$$\tau_{\mathsf{c}}(\iota, \sigma, j) \equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(\iota) - \delta(\iota)$$

Instructions

semantics of instruction ι

$$\tau_{\mathsf{st}}(\texttt{ADD}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1)$$
$$\tau_{\mathsf{c}}(\iota, \sigma, j) \equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(\iota) - \delta(\iota)$$

Instructions

semantics of instruction ι

$$\tau_{\mathsf{st}}(\texttt{ADD}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1)$$
$$\tau_{\mathsf{c}}(\iota, \sigma, j) \equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(\iota) - \delta(\iota)$$
$$\tau_{\mathsf{pres}}(\iota, \sigma, j) \equiv \forall n < \mathsf{c}_\sigma(j) - \delta(\iota).\ \mathsf{st}_\sigma(j+1, n) = \mathsf{st}_\sigma(j, n)$$

Instructions

semantics of instruction ι

$$\tau_{\mathsf{st}}(\texttt{ADD}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1)$$
$$\tau_{\mathsf{c}}(\iota, \sigma, j) \equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(\iota) - \delta(\iota)$$
$$\tau_{\mathsf{pres}}(\iota, \sigma, j) \equiv \forall n < \mathsf{c}_\sigma(j) - \delta(\iota).\ \mathsf{st}_\sigma(j+1, n) = \mathsf{st}_\sigma(j, n)$$
$$\tau_{\mathsf{g}}(\iota, \sigma, j) \equiv \mathsf{g}_\sigma(j+1) = \mathsf{g}_\sigma(j) + C(\iota)$$

Instructions

semantics of instruction ι

$$\tau_{\mathsf{st}}(\texttt{ADD}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1)$$
$$\tau_{\mathsf{c}}(\iota, \sigma, j) \equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(\iota) - \delta(\iota)$$
$$\tau_{\mathsf{pres}}(\iota, \sigma, j) \equiv \forall n < \mathsf{c}_\sigma(j) - \delta(\iota).\ \mathsf{st}_\sigma(j+1, n) = \mathsf{st}_\sigma(j, n)$$
$$\tau_{\mathsf{g}}(\iota, \sigma, j) \equiv \mathsf{g}_\sigma(j+1) = \mathsf{g}_\sigma(j) + C(\iota)$$

$$\tau(\iota, \sigma, j) \equiv \tau_{\mathsf{g}}(\iota, \sigma, j) \wedge \tau_{\mathsf{c}}(\iota, \sigma, j) \wedge \tau_{\mathsf{pres}}(\iota, \sigma, j) \wedge \tau_{\mathsf{st}}(\iota, \sigma, j)$$

Instructions

semantics of instruction ι

$$\begin{aligned}
\tau_{\mathsf{st}}(\mathtt{ADD}, \sigma, j) &\equiv \mathsf{st}_\sigma(j+1, \top) = \mathsf{st}_\sigma(j, \top) +_{bv} \mathsf{st}_\sigma(j, \top - 1) \\
\tau_{\mathsf{c}}(\iota, \sigma, j) &\equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(\iota) - \delta(\iota) \\
\tau_{\mathsf{pres}}(\iota, \sigma, j) &\equiv \forall n < \mathsf{c}_\sigma(j) - \delta(\iota).\ \mathsf{st}_\sigma(j+1, n) = \mathsf{st}_\sigma(j, n) \\
\tau_{\mathsf{g}}(\iota, \sigma, j) &\equiv \mathsf{g}_\sigma(j+1) = \mathsf{g}_\sigma(j) + C(\iota)
\end{aligned}$$

$$\tau(\iota, \sigma, j) \equiv \tau_{\mathsf{g}}(\iota, \sigma, j) \wedge \tau_{\mathsf{c}}(\iota, \sigma, j) \wedge \tau_{\mathsf{pres}}(\iota, \sigma, j) \wedge \tau_{\mathsf{st}}(\iota, \sigma, j)$$

for program $p = \iota_0, \ldots, \iota_n$ we define

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

Instructions

semantics of instruction ι

$$\tau_{\mathsf{st}}(\mathtt{ADD}, \vec{x}, \sigma, j) \equiv \mathsf{st}_\sigma(\vec{x}, j+1, \top) = \mathsf{st}_\sigma(\vec{x}, j, \top) +_{bv} \mathsf{st}_\sigma(\vec{x}, j, \top - 1)$$
$$\tau_{\mathsf{c}}(\iota, \sigma, j) \equiv \mathsf{c}_\sigma(j+1) = \mathsf{c}_\sigma(j) + \alpha(\iota) - \delta(\iota)$$
$$\tau_{\mathsf{pres}}(\iota, \vec{x}, \sigma, j) \equiv \forall n < \mathsf{c}_\sigma(j) - \delta(\iota).\ \mathsf{st}_\sigma(\vec{x}, j+1, n) = \mathsf{st}_\sigma(\vec{x}, j, n)$$
$$\tau_{\mathsf{g}}(\iota, \vec{x}, \sigma, j) \equiv \mathsf{g}_\sigma(j+1) = \mathsf{g}_\sigma(j) + C(\iota)$$

$$\tau(\iota, \vec{x}, \sigma, j) \equiv \tau_{\mathsf{g}}(\iota, \sigma, j) \wedge \tau_{\mathsf{c}}(\iota, \sigma, j) \wedge \tau_{\mathsf{pres}}(\iota, \vec{x}, \sigma, j) \wedge \tau_{\mathsf{st}}(\iota, \vec{x}, \sigma, j)$$

for program $p = \iota_0, \ldots, \iota_n$ we define

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

Equality

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

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

Superoptimization

Basic

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

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

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

$$\textsc{EncUso}(p) = \forall \vec{x}.\, \tau(p, \vec{x}, \sigma) \wedge \epsilon(\vec{x}, \sigma, \sigma', 0, 0) \wedge \epsilon(\vec{x}, \sigma, \sigma', |p|, n)$$

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

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

Uninterpreted Instructions

- `BLOCKHASH`, `ADDRESS`, `BALANCE`, ... arbitrary, but fixed, result

Uninterpreted Instructions

- BLOCKHASH, ADDRESS, BALANCE, ... arbitrary, but fixed, result

Constant

- **idea:** add $\forall$-quantified variables u_ι
- e.g. $\tau_{\mathsf{st}}(\texttt{ADDRESS}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = u_{\texttt{ADDRESS}}$

Uninterpreted Instructions

- BLOCKHASH, ADDRESS, BALANCE, ... arbitrary, but fixed, result

Constant

- **idea:** add $\forall$-quantified variables u_ι
- e.g. $\tau_{\mathsf{st}}(\texttt{ADDRESS}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = u_{\texttt{ADDRESS}}$

Non-constant

- **idea:** add f_ι and $\forall$-quantified variables u_{ι_j} for j occurences of ι
- initalize f_ι as read-only memory with arguments from stack

Uninterpreted Instructions

- BLOCKHASH, ADDRESS, BALANCE, ... arbitrary, but fixed, result

Constant

- **idea:** add $\forall$-quantified variables u_ι
- e.g. $\tau_{\mathsf{st}}(\texttt{ADDRESS}, \sigma, j) \equiv \mathsf{st}_\sigma(j+1, \top) = u_{\texttt{ADDRESS}}$

Non-constant

- **idea:** add f_ι and $\forall$-quantified variables u_{ι_j} for j occurences of ι
- initalize f_ι as read-only memory with arguments from stack

+ Storage

Initalize

- BLOCKHASH at positions $j_1, \ldots, j_\ell$
- variables $u_1, \ldots, u_\ell$ for result
- arguments $a_{j_i} = \mathsf{st}_\sigma(j_i, \top)$

$$\forall w.\, f_{\texttt{BLOCKHASH}}(w) = \mathit{ite}(w = a_{j_1}, u_1,$$
$$\mathit{ite}(w = a_{j_2}, u_2,$$
$$\ldots$$
$$\mathit{ite}(w = a_{j_\ell}, u_\ell, \_)))$$

Initalize

- BLOCKHASH at positions $j_1, \ldots, j_\ell$
- variables $u_1, \ldots, u_\ell$ for result
- arguments $a_{j_i} = \mathsf{st}_\sigma(\vec{x}, j_i, \top)$

$$\forall w.\, f_{\texttt{BLOCKHASH}}(\vec{x}, w) = ite(w = a_{j_1}, u_1,$$
$$ite(w = a_{j_2}, u_2,$$
$$\ldots$$
$$ite(w = a_{j_\ell}, u_\ell, \_)))$$

Implementation

Implementation

- available at

  `github.com/juliannagele/ebso`

- implemented in OCaml

- ∼1.6 kloc, 635 tests
- using Z3 as SMT solver

Tanslation Validation

- **given:** large word size of EVM 256 bit $\Longrightarrow$ scalability problems
- **solution:** find model for small word size & validate for 256 bit

Tanslation Validation

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

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

Evaluation

EvaluationS

1. **“Optimize the Optimized”**
   - ⋆ Gas Golfing Contest: 199 `Solidity` contracts
   - ⋆ $\Longrightarrow$ 2743 ebso blocks

2. **“BasicSo vs. UnboundedSo”**
   - ⋆ bytecode of 2500 most called contracts from Etherum Blockchain
   - ⋆ $\Longrightarrow$ 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)

	#	%
(O) optimized (optimal)		
(P) proved optimal		
(T) time-out (transl. val. fail)		

Optimize The Optimized (1)

	#	%
(O) optimized (optimal)	19 (10)	0.69 %
(P) proved optimal		
(T) time-out (transl. val. fail)		

Optimize The Optimized (1)

	#	%
(O) optimized (optimal)	19 (10)	0.69 %
(P) proved optimal	481	17.54 %
(T) time-out (transl. val. fail)		

Optimize The Optimized (1)

	#	%
(O) optimized (optimal)	19 (10)	0.69 %
(P) proved optimal	481	17.54 %
(T) time-out (transl. val. fail)	2243 (196)	81.77 %

Optimize The Optimized (2)

(12) `CALLVALUE DUP1 ISZERO PUSH 81` and

(13) `CALLVALUE DUP1 ISZERO PUSH 298`

to `CALLVALUE CALLVALUE ISZERO PUSH _`

Optimize The Optimized (2)

(12) `CALLVALUE DUP1 ISZERO PUSH 81` and

(13) `CALLVALUE DUP1 ISZERO PUSH 298`

to `CALLVALUE CALLVALUE ISZERO PUSH _`

(17) `POP PUSH 1 SWAP1 POP PUSH 0`

Optimize The Optimized (2)

(12) `CALLVALUE DUP1 ISZERO PUSH 81` and

(13) `CALLVALUE DUP1 ISZERO PUSH 298`

to `CALLVALUE CALLVALUE ISZERO PUSH _`

(17) `POP PUSH 1 SWAP1 POP PUSH 0`

to `SLT DUP1 EQ PUSH 0`

BasicSo vs. UnboundedSo (1)

	UnboundedSo		BasicSo	
	#	%	#	%
(O)				
(P)				
(T)				

- (O) optimized (optimal)
- (P) proved optimal
- (T) time-out (translation validation failed)

BasicSo vs. UnboundedSo (1)

	UnboundedSo		BasicSo	
	#	%	#	%
(O)	943 (393)	1.54 % (0.64 %)	184	0.3 %
(P)				
(T)				

- (O) optimized (optimal)
- (P) proved optimal
- (T) time-out (translation validation failed)

BasicSo vs. UnboundedSo (1)

	UnboundedSo		BasicSo	
	#	%	#	%
(O)	943 (393)	1.54 % (0.64 %)	184	0.3 %
(P)	3882	6.34 %	348	0.57 %
(T)				

- (O) optimized (optimal)
- (P) proved optimal
- (T) time-out (translation validation failed)

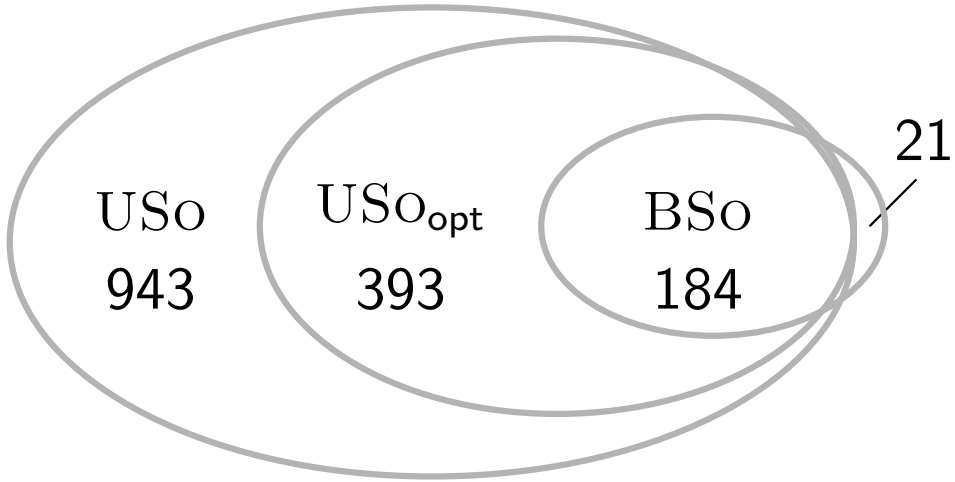
BasicSo vs. UnboundedSo (1)

	UnboundedSo		BasicSo	
	#	%	#	%
(O)	943 (393)	1.54 % (0.64 %)	184	0.3 %
(P)	3882	6.34 %	348	0.57 %
(T)	56 392 (1467)	92.12 % (2.4 %)	60 685	99.13 %

- (O) optimized (optimal)
- (P) proved optimal
- (T) time-out (translation validation failed)

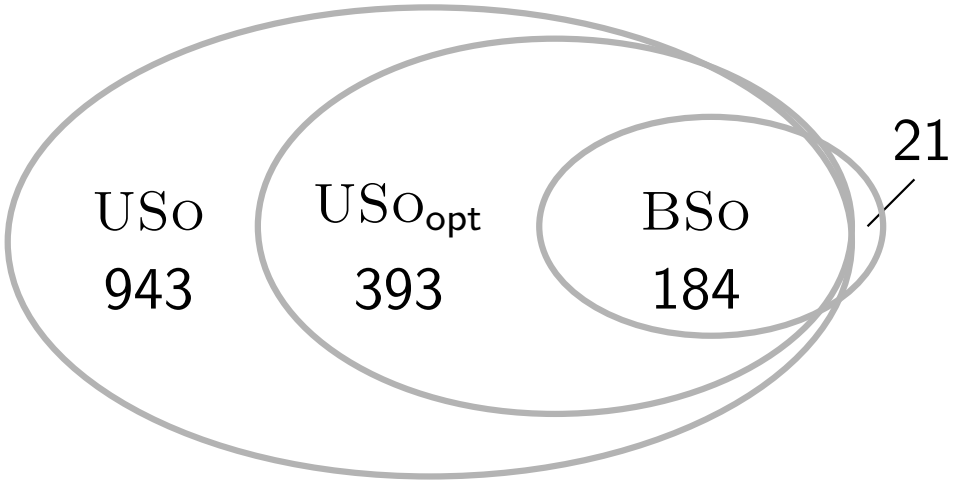
BasicSo vs. UnboundedSo (2)

- optimizations for basic (BSo) and unbounded (USo) superoptimization



BasicSo vs. UnboundedSo (2)

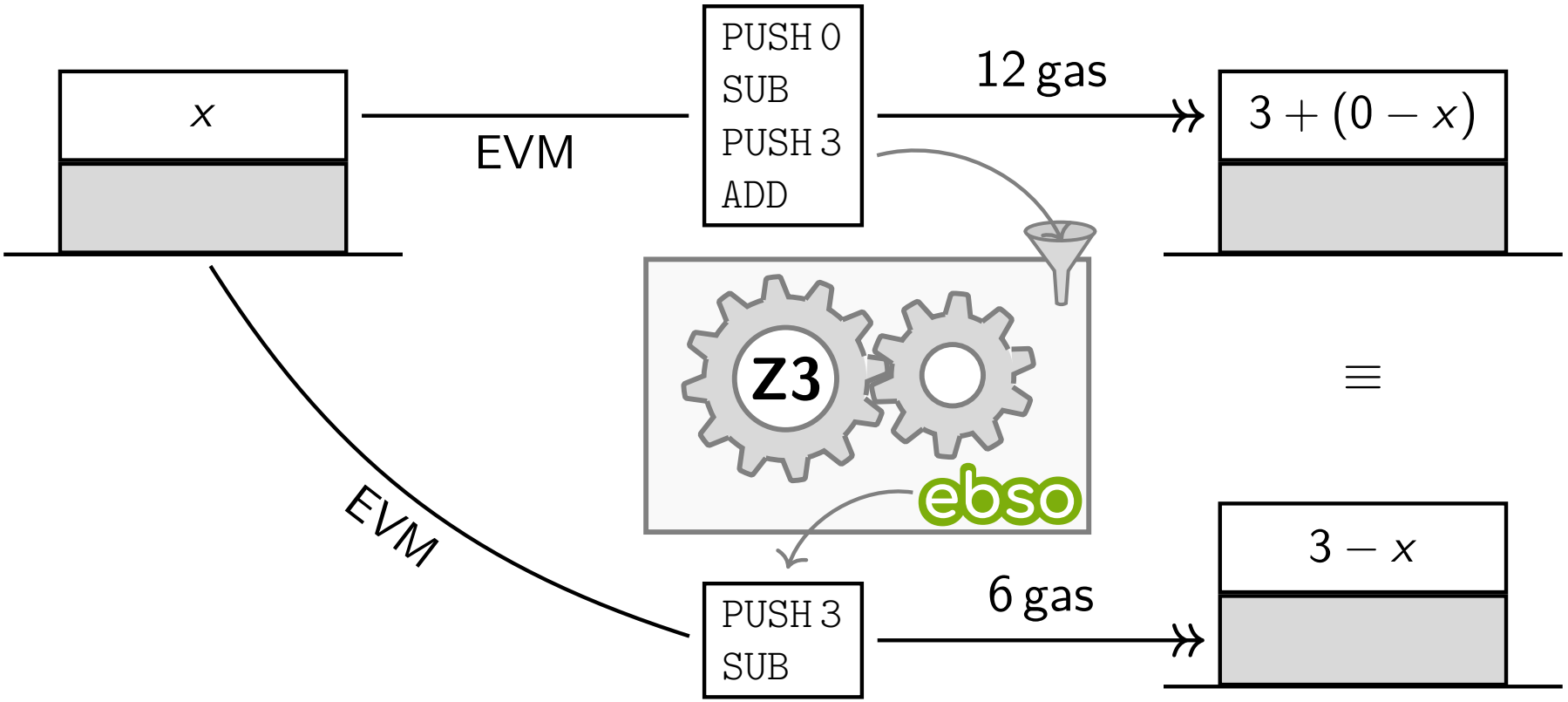
- optimizations for basic (BSo) and unbounded (USo) superoptimization



- `PUSH 0 PUSH 12 SLOAD LT ISZERO ISZERO ISZERO` `PUSH 12250` to `PUSH 1 PUSH 12250`

Conclusion

Overview


x
EVM
PUSH 0
SUB
PUSH 3
ADD
12 gas
3 + (0 − x)
Z3
ebso
≡
EVM
PUSH 3
SUB
6 gas
3 − x

Future

Application

- generalize patterns to rewrite
- beyond straight-line code with control flow analysis
- eWasm

Future

Application

- generalize patterns to rewrite
- beyond straight-line code with control flow analysis
- eWasm

SMT

- extract benchmarks
- strategies to guide solver(s)

Thank you & questions?

available at `github.com/juliannagele/ebso`

`www.maria-a-schett.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