

Zoom in on EOF stack validation



Exploring the rationale behind one of the most complicated parts of EOF spec

Andrei Maiboroda
github.com/gumb0
twitter.com/gumb00

EOF intro

EIP-7692: EVM Object Format (EOFv1) Meta

- EIP-3540: EOF - EVM Object Format v1
- EIP-3670: EOF - Code Validation
- EIP-4200: EOF - Static relative jumps
- EIP-4750: EOF - Functions
- EIP-5450: EOF - Stack Validation
- EIP-6206: EOF - JUMPF and non-returning functions
- EIP-7480: EOF - Data section access instructions
- EIP-663: SWAPN, DUPN and EXCHANGE instructions
- EIP-7069: Revamped CALL instructions
- EIP-7620: EOF Contract Creation
- EIP-7698: EOF - Creation transaction



EOF intro

EIP-7692: EVM Object Format (EOFv1) Meta

- EIP-3540: EOF - EVM Object Format v1
- EIP-3670: EOF - Code Validation
- EIP-4200: EOF - Static relative jumps
- EIP-4750: EOF - Functions
- **EIP-5450: EOF - Stack Validation**
- EIP-6206: EOF - JUMPF and non-returning functions
- EIP-7480: EOF - Data section access instructions
- EIP-663: SWAPN, DUPN and EXCHANGE instructions
- EIP-7069: Revamped CALL instructions
- EIP-7620: EOF Contract Creation
- EIP-7698: EOF - Creation transaction



Stack Validation: goals

- Part of deploy-time code validation
 - What can be checked once, should not be checked at every run
- Stack validation: check operand stack-correctness of each instruction
 - Get rid of run-time stack underflow and overflow checks
- $O(n)$ complexity

Stack Validation: general idea

- Traverse control flow graph of the code
- Examine every instruction
- Keep track of how many values are pushed/popped
- If stack underflow or overflow, code is invalid

Stack (underflows) Validation: 1st attempt

- Each code section is validated independently
- Go through the code following jumps (BFS or DFS)
- Record stack height at each instruction
- Check each instruction for underflow
- Extra restriction: if instruction can be reached via different code paths, stack height must be equal across all paths
- This allows to examine each branch only once
- Unreachable code is invalid

Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0		[0]
1	calldataload		
2	rjumpi x		
5	push1 1		
7	rjump y		
10	x: push1 2		
12	y: push0		
13	sstore		
14	stop		

Stack Validation: 1st attempt - walkthrough

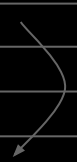
offset	code	stack height before	work list
0 >	push0	0	[]
1	calldataload		
2	rjumpi x		
5	push1 1		
7	rjump y		
10	x: push1 2		
12	y: push0		
13	sstore		
14	stop		

Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[]
1 >	calldataload	1	
2	rjumpi x		
5	push1 1		
7	rjump y		
10	x: push1 2		
12	y: push0		
13	sstore		
14	stop		


Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[10, 5]
1	calldataload	1	
2 >	rjumpi x	1	
5	push1 1		
7	rjump y		
10	x: push1 2		
12	y: push0		
13	sstore		
14	stop		

A curved arrow originates from the 'rjumpi x' instruction at offset 2 and points to the 'x: push1 2' instruction at offset 10, indicating a jump to a label.

Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[10, 5]
1	calldataload	1	
2 >	rjumpi x	1	
5	push1 1	0	
7	rjump y		
10	x: push1 2	0	
12	y: push0		
13	sstore		
14	stop		



Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[5]
1	calldataload	1	
2	rjumpi x	1	
5	push1 1	0	
7	rjump y		
10 >	x: push1 2	0	
12	y: push0		
13	sstore		
14	stop		

Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[5]
1	calldataload	1	
2	rjumpi x	1	
5	push1 1	0	
7	rjump y		
10	x: push1 2	0	
12 >	y: push0	1	
13	sstore		
14	stop		

Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[5]
1	calldataload	1	
2	rjumpi x	1	
5	push1 1	0	
7	rjump y		
10	x: push1 2	0	
12	y: push0	1	
13 >	sstore	2	
14	stop		

Stack Validation: 1st attempt - walkthrough


offset	code	stack height before	work list
0	push0	0	[5]
1	calldataload	1	
2	rjumpi x	1	
5	push1 1	0	
7	rjump y		
10	x: push1 2	0	
12	y: push0	1	
13	sstore	2	
14 >	stop	0	

Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[]
1	calldataload	1	
2	rjumpi x	1	
5 >	push1 1	0	
7	rjump y		
10	x: push1 2	0	
12	y: push0	1	
13	sstore	2	
14	stop	0	

Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[12]
1	calldataload	1	
2	rjumpi x	1	
5	push1 1	0	
7 >	rjump y	1	
10	x: push1 2	0	
12	y: push0	1	
13	sstore	2	
14	stop	0	



Stack Validation: 1st attempt - walkthrough

offset	code	stack height before	work list
0	push0	0	[]
1	calldataload	1	
2	rjumpi x	1	
5	push1 1	0	
7	rjump y	1	
10	x: push1 2	0	
12 >	y: push0	1 == 1	
13	sstore	2	
14	stop	0	

Stack Validation: 1st attempt - invalid case

offset	code	stack height before	work list
0	push0	0	[10]
1	calldataload	1	
2	rjumpi x	1	
5 >	rjump y	0	
8	x: push1 2	0	
10	y: push0	1 != 0	
11	sstore	2	
12	stop	0	

Stack Validation: 1st attempt - problems

- Not possible to jump into a helper from different stack heights
- Some compiler optimizations become not worth it
- Code size and gas regressions
- Extracting helper to a separate section has overhead in EOF header size

```
function foo() {  
    if (msg.sender != owner)  
        goto exit;  
    ...  
    if (msg.value < required)  
        goto exit;  
    ...  
    return(0, 0);  
exit:  
    revert(0, 0);  
}
```

Stack Validation: current EIP-5450 spec

- For each code location track the [**stack_min**; **stack_max**] range
- Traverse the code sequentially
- Forward jump instruction expands the target's stack range
 - For any block reached from multiple jumps we collect all possible heights *before* we get to validate it
- Backwards jump (typically a loop) can only target equal stack range (same restriction as in the 1st attempt)
 - If backwards jump to a helper, code should be reordered
- Non-jump instructions shift the range by pushed/popped number

Stack Validation: current EIP-5450 spec

- For each code location track the [**stack_min**; **stack_max**] range
- Traverse the code sequentially
- Stack underflow if **stack_min** is not enough
- Extra restriction: code that is reached only by backwards jump is considered unreachable and invalid
 - This makes it possible to traverse sequentially
- Each branch is still examined only once

Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0		
1	calldataload		
2	rjumpi x		
5	push0		
6	push1 32		
8	calldataload		
9	rjumpi x		
12	stop		
13	x: push0		
14	push0		
15	revert		

Stack Validation: walkthrough


offset	code	stack height min before	stack height max before
0 >	push0	0	0
1	calldataload		
2	rjumpi x		
5	push0		
6	push1 32		
8	calldataload		
9	rjumpi x		
12	stop		
13	x: push0		
14	push0		
15	revert		

Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1 >	calldataload	1	1
2	rjumpi x		
5	push0		
6	push1 32		
8	calldataload		
9	rjumpi x		
12	stop		
13	x: push0		
14	push0		
15	revert		


Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2 >	rjumpi x	1	1
5	push0		
6	push1 32		
8	calldataload		
9	rjumpi x		
12	stop		
13	x: push0		
14	push0		
15	revert		



Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2 >	rjumpi x	1	1
5	push0		
6	push1 32		
8	calldataload		
9	rjumpi x		
12	stop		
13	x: push0	0	0
14	push0		
15	revert		



Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5 >	push0	0	0
6	push1 32		
8	calldataload		
9	rjumpi x		
12	stop		
13	x: push0	0	0
14	push0		
15	revert		

Stack Validation: walkthrough


offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5	push0	0	0
6 >	push1 32	1	1
8	calldataload		
9	rjumpi x		
12	stop		
13	x: push0	0	0
14	push0		
15	revert		

Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5	push0	0	0
6	push1 32	1	1
8 >	calldataload	2	2
9	rjumpi x		
12	stop		
13	x: push0	0	0
14	push0		
15	revert		


Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5	push0	0	0
6	push1 32	1	1
8	calldataload	2	2
9 >	rjumpi x	2	2
12	stop		
13	x: push0	0	0
14	push0		
15	revert		



Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5	push0	0	0
6	push1 32	1	1
8	calldataload	2	2
9 >	rjumpi x	2	2
12	stop		
13	x: push0	0	1
14	push0		
15	revert		



Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5	push0	0	0
6	push1 32	1	1
8	calldataload	2	2
9	rjumpi x	2	2
12 >	stop	1	1
13	x: push0	0	1
14	push0		
15	revert		

Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5	push0	0	0
6	push1 32	1	1
8	calldataload	2	2
9	rjumpi x	2	2
12	stop	1	1
13 >	x: push0	0	1
14	push0		
15	revert		

Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5	push0	0	0
6	push1 32	1	1
8	calldataload	2	2
9	rjumpi x	2	2
12	stop	1	1
13	x: push0	0	1
14 >	push0	1	2
15	revert		

Stack Validation: walkthrough

offset	code	stack height min before	stack height max before
0	push0	0	0
1	calldataload	1	1
2	rjumpi x	1	1
5	push0	0	0
6	push1 32	1	1
8	calldataload	2	2
9	rjumpi x	2	2
12	stop	1	1
13	x: push0	0	1
14	push0	1	2
15 >	revert	2	3

Stack overflow validation

- We can find out maximum stack height of a function
 - If max stack height ≥ 1024 , code is invalid
 - Save max in types section of EOF header
- CALLF and JUMPF validation checks that current stack + callee max stack ≤ 1024
- At runtime check for stack overflow *only* at CALLF and JUMPF

JUMPF and non-returning functions

- [EIP-6206: EOF - JUMPF and non-returning functions](#)
- Functions that end execution with STOP, RETURN, REVERT, RETURNCONTRACT, INVALID, infinite loop
- Code after transferring control flow into non-returning function is **unreachable**
 - Stack validation needs to identify non-returning functions (0x80 output in type section)
 - Stack validation considers a call to non-returning function to be a terminating instruction
- JUMPF is an instruction to call non-returning functions (CALLF to non-returning is not allowed)
- But JUMPF can also target returning functions, providing “tail call” optimization
- More details: <https://github.com/ipsilon/eof/issues/42>

Links

- [Slides](#)
- evmobjectformat.org
- [EIP-7692: EVM Object Format \(EOFv1\) Meta](#)
- ["Mega EOF Endgame" Specification](#)
- evm.codes/?fork=EOF
- Explainers: [1](#) [2](#) [3](#) [4](#)

