1 The ADD instruction

Here is the Yellow Paper semantics (Gavin, 2019) of the ADD operation adding the two elements on top of the execution stack.

ADD	2	1	Addition operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] + \boldsymbol{\mu}_{\mathbf{s}}[1]$

In this semantics, μ_s is the execution stack and the effect of this operation is to compute a new execution stack μ'_s whose first element $\mu'_s[0]$ is the sum of the two top elements of μ, i.e., $\mu_s[0]$ and $\mu_s[1]$. The cost of the ADD operation is defined in another part of the semantics and is fixed to 3 gas units. Here are the two rules of the EtherTrust small-step semantics of this operation in (ethertrust, 2017).

$$\frac{\iota.code[\mu.\mathsf{pc}] = \mathsf{ADD} \qquad \mu.\mathsf{s} = a :: b :: s \qquad \mu.\mathsf{gas} \geq 3 \qquad \mu' = \mu[\mathsf{s} \rightarrow (a+b) :: s][\mathsf{pc} \mathrel{+}= 1][\mathsf{gas} \mathrel{-}= 3]}{\Gamma \vDash (\mu, \iota, \sigma, \eta) :: S \;\rightarrow (\mu', \iota, \sigma, \eta) :: S}$$

$$\frac{\iota.code[\mu.\mathsf{pc}] = \mathsf{ADD} \qquad (|\mu.\mathsf{s}| < 2 \vee \mu.\mathsf{gas} < 3)}{\Gamma \vDash (\mu, \iota, \sigma, \eta) :: S \;\rightarrow EXC :: S}$$

In this semantics, μ is the local state of the stack machine where $\mu.s$ denotes the execution stack, $\mu.pc$ the program counter, $\mu.gas$ the available gas. The other record ι represents the parameters of the transaction where $\iota.code$ denotes the program under execution. Thus $\iota.code[\mu.pc]$ is the current instruction to execute. Below the line of the semantic rules, $(\mu, \iota, \sigma, \eta) :: S$ is the current call stack. An element of the call stack is called a *frame*, e.g., $(\mu, \iota, \sigma, \eta)$ is the top frame of the current call stack. The field η is a transaction effect where the only information that could be relevant for us w.r.t. gas consumption would be the refund counter. However, as explained in Section **??**, this refund counter is separate from the gas available for operation execution. Finally, σ is the current state of the global state. Since, there are no side effects, every update on this global state is propagated by the semantic rules. In the first rule, for ADD, there is enough gas to execute ADD and an execution stack with at least two elements. Thus, the call stack becomes $(\mu', \iota, \sigma, \eta) :: S$ where μ' is μ with an updated execution stack, an increased program counter $\mu.pc$, and a $\mu.gas$ decreased of 3 gas units. The second rule defines the execution of ADD when there are not enough elements on the execution stack or not enough gas to execute ADD. This results into stacking an exception frame (EXC) on top of the call stack.

2 The CALL instruction

Yellow Paper's definition for the CALL operation is given in Figure 1. In this definition, $\mu'.g$ is the gas after the execution of this instruction. The value of $\mu'.g$ is set to $\mu.g + g'$ where g' is the gas remaning after the execution of the called contract (gas refund). In fact, the execution of the CALL instruction itself has a cost which is subtracted from $\mu'.g$ (rule (135) of the Yellow Paper semantics) for executing the instruction. Thus $\mu'.g$ should be read as $\mu.g - CallCost + g'$. Fortunately, EtherTrust provides a higher level and self-contained small-step interpretation of the semantics of this operation. In the following, since they are more readable, we only present the EtherTrust version of EVM semantic rules.

0xf1	CALL	7	1	Message-call into an account.

$$\mathbf{i} \equiv \boldsymbol{\mu}_{\mathbf{m}}[\boldsymbol{\mu}_{\mathbf{s}}[3] \ldots (\boldsymbol{\mu}_{\mathbf{s}}[3] + \boldsymbol{\mu}_{\mathbf{s}}[4] - 1)]$$

$$(\boldsymbol{\sigma}', g', A^+, \mathbf{o}) \equiv \begin{cases} \Theta(\boldsymbol{\sigma}, I_{\mathrm{a}}, I_{\mathrm{o}}, t, t, C_{\mathrm{CALLGAS}}(\boldsymbol{\mu}), & \text{if} \quad \boldsymbol{\mu}_{\mathbf{s}}[2] \leqslant \boldsymbol{\sigma}[I_{\mathrm{a}}]_{\mathrm{b}} \wedge \\ \quad I_{\mathrm{p}}, \boldsymbol{\mu}_{\mathbf{s}}[2], \boldsymbol{\mu}_{\mathbf{s}}[2], \mathbf{i}, I_{\mathrm{e}} + 1, I_{\mathrm{w}}) & \quad I_{\mathrm{e}} < 1024 \\ (\boldsymbol{\sigma}, g, \varnothing, ()) & \text{otherwise} \end{cases}$$

$$n \equiv \min(\{\boldsymbol{\mu}_{\mathbf{s}}[6], \|\mathbf{o}\|\})$$

$$\boldsymbol{\mu}'_{\mathbf{m}}[\boldsymbol{\mu}_{\mathbf{s}}[5] \ldots (\boldsymbol{\mu}_{\mathbf{s}}[5] + n - 1)] = \mathbf{o}[0 \ldots (n-1)]$$

$$\boldsymbol{\mu}'_{\mathbf{o}} = \mathbf{o}$$

$$\boldsymbol{\mu}'_{\mathrm{g}} \equiv \boldsymbol{\mu}_{\mathrm{g}} + g'$$

$$\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv x$$

$$A' \equiv A \uplus A^+$$

$$t \equiv \boldsymbol{\mu}_{\mathbf{s}}[1] \mod 2^{160}$$

where $x = 0$ if the code execution for this operation failed due to an exceptional halting (or for a REVERT) $\boldsymbol{\sigma}' = \varnothing$ or if $\boldsymbol{\mu}_{\mathbf{s}}[2] > \boldsymbol{\sigma}[I_{\mathrm{a}}]_{\mathrm{b}}$ (not enough funds) or $I_{\mathrm{e}} = 1024$ (call depth limit reached); $x = 1$ otherwise.

$$\boldsymbol{\mu}'_{\mathrm{i}} \equiv M(M(\boldsymbol{\mu}_{\mathrm{i}}, \boldsymbol{\mu}_{\mathbf{s}}[3], \boldsymbol{\mu}_{\mathbf{s}}[4]), \boldsymbol{\mu}_{\mathbf{s}}[5], \boldsymbol{\mu}_{\mathbf{s}}[6])$$

Thus the operand order is: gas, to, value, in offset, in size, out offset, out size.

$$C_{\mathrm{CALL}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv C_{\mathrm{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) + C_{\mathrm{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu})$$

$$C_{\mathrm{CALLGAS}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} C_{\mathrm{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) + G_{callstipend} & \text{if} \quad \boldsymbol{\mu}_{\mathbf{s}}[2] \neq 0 \\ C_{\mathrm{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) & \text{otherwise} \end{cases}$$

$$C_{\mathrm{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} \min\{L(\boldsymbol{\mu}_{\mathrm{g}} - C_{\mathrm{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu})), \boldsymbol{\mu}_{\mathbf{s}}[0]\} & \text{if} \quad \boldsymbol{\mu}_{\mathrm{g}} \geq C_{\mathrm{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \\ \boldsymbol{\mu}_{\mathbf{s}}[0] & \text{otherwise} \end{cases}$$

$$C_{\mathrm{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv G_{call} + C_{\mathrm{XFER}}(\boldsymbol{\mu}) + C_{\mathrm{NEW}}(\boldsymbol{\sigma}, \boldsymbol{\mu})$$

$$C_{\mathrm{XFER}}(\boldsymbol{\mu}) \equiv \begin{cases} G_{callvalue} & \text{if} \quad \boldsymbol{\mu}_{\mathbf{s}}[2] \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$C_{\mathrm{NEW}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} G_{newaccount} & \text{if} \quad \mathtt{DEAD}(\boldsymbol{\sigma}, \boldsymbol{\mu}_{\mathbf{s}}[1] \mod 2^{160}) \wedge \boldsymbol{\mu}_{\mathbf{s}}[2] \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Figure 1: The Yellow Paper definition of the CALL operation.

$$\frac{\begin{array}{c}\iota.code\,[\mu.\mathsf{pc}] = \mathsf{CALL} \qquad \mu.\mathsf{s} = g :: to :: va :: io :: is :: oo :: os :: s \\ \sigma(to) \neq \bot \qquad |A| + 1 < 1024 \qquad \sigma(\iota.\mathsf{actor}).\mathsf{b} \geq va \qquad aw = M\,(M\,(\mu.\mathsf{i}, io, is), oo, os) \\ c_{call} = C_{gascap}\,(va, 1, g, \mu.\mathsf{gas}) \qquad c = C_{base}\,(va, 1) + C_{mem}\,(\mu.\mathsf{i}, aw) + c_{call} \\ \mu.\mathsf{gas} \geq c \qquad \sigma' = \sigma\langle to \to \sigma(to)[\mathsf{b} \mathrel{+}= va]\rangle\langle \iota.\mathsf{actor} \to \sigma(\iota.\mathsf{actor})[\mathsf{b} \mathrel{-}= va]\rangle \\ d = \mu.\mathsf{m}\,[io, io + is - 1] \qquad \mu' = (c_{call}, 0, \lambda x.\,0, 0, \epsilon) \\ \iota' = \iota[\mathsf{sender} \to \iota.\mathsf{actor}][\mathsf{actor} \to to][\mathsf{value} \to va][\mathsf{input} \to d][\mathsf{code} \to \sigma(to).\mathsf{code}]\end{array}}{\Gamma \vDash (\mu, \iota, \sigma, \eta) :: S \;\to\; (\mu', \iota', \sigma', \eta) :: (\mu, \iota, \sigma, \eta) :: S}$$

This small-step semantic rule defines the CALL execution when everything is OK: the execution stack contains enough arguments to perform the call ($\mu.s$ has at least 7 elements), there is enough gas to perform the call $\mu.gas \geq c$, and there is room in the call stack to add a new frame ($|A| + 1 < 1024$). The cost c is the sum of the costs for calling the CALL instruction itself (700 gas units) plus a variable cost depending on the size of the input and output of the contract: this gas is paid when reading contract parameters and outputing its future result. On the lower part of this rule, the call stack $(\mu, \iota, \sigma, \eta) :: S$ becomes $(\mu', \iota', \sigma', \eta') :: (\mu, \iota, \sigma, \eta) :: S$ where $(\mu', \iota', \sigma', \eta')$ is a new frame stack which has been added on top of the call stack, where μ' is a new record, where $\mu'.gas = c_{call}$ is the gas transferred to the new frame stack by the old one and $\mu'.pc$ is set to 0. The code to execute in this new frame is $\iota'.code = \sigma(to).code$ where $\sigma(to)$ is the account receiving the call. Note that, like it was stated in the above sections, the new call stack is $(\mu', \iota', \sigma', \eta') :: (\mu, \iota, \sigma, \eta) :: S$ where the gas sent to the new frame ($\mu'.gas$) has not been subtracted from the frame $(\mu, \iota, \sigma, \eta)$ (μ is the same, thus so is $\mu.gas$). The gas is retracted when the contracts returns. Note also that this is compatible with the Yellow Paper semantics where, to update the gas w.r.t. the execution of the CALL, one has to know how much gas g' will be refunded *after* the execution of the called contract.

REFERENCES

ethertrust (2017). Ethertrust - Trustworthy smart contracts. `https://www.netidee.at/ethertrust`.

evmcodeyellow (2020). Ethereum Formal Semantics for Gas Consumption Analysis - Yellow paper semantics. file `abstEvm.thy` in `https://anonymous.4open.science/r/56a5da34-18bb-40a2-9484-846c3c9ff663/`.

Gavin, W. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151, EIP-150 Revision, `http://gavwood.com/Paper.pdf`.

Gavin, W. (2019). Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, BYZANTIUM VERSION e7515a3, `https://ethereum.github.io/yellowpaper/paper.pdf`.

Grishchenko, I., Maffei, M., and Schneidewind, C. (2018a). Foundations and Tools for the Static Analysis of Ethereum Smart Contracts. In *CAV'18*, volume 10981 of *LNCS*, pages 51–78. Springer.

Grishchenko, I., Maffei, M., and Schneidewind, C. (2018b). A Semantic Framework for the Security Analysis of Ethereum Smart Contracts. In *POST'18*, volume 10804 of *LNCS*, pages 243–269. Springer.