Blockchain System

1 Definitions

Definition 1. *Let* **Elt** *be the set of (concrete) elements. Let* $\emptyset$ *be an empty set and* $\mathbf{e} \in \mathbf{Elt}$*. A set of elements is expressed as the following syntax:*
$\mathbf{s} ::= \emptyset \mid \mathbf{e} \mid \mathbf{s} :: \mathbf{s}$

1.1 Local Node

Definition 2. *An implicit* account *is a tuple* $\langle \boldsymbol{pak}, \boldsymbol{puk} \rangle$*, where* ***pak*** *is its private key and* ***puk*** *is its public key.*

Definition 3. Operations *are defined by the following grammar:*

$$\begin{aligned} \boldsymbol{op} ::= & \textit{ transfer } \boldsymbol{n} \textit{ from } \boldsymbol{puk} \textit{ to } \boldsymbol{addr} \textit{ arg } \boldsymbol{p} \textit{ fee } \boldsymbol{m} \\ \mid & \textit{ originate contract transferring } \boldsymbol{n} \textit{ from } \boldsymbol{puk} \textit{ running } \boldsymbol{code} \textit{ init } \boldsymbol{s} \textit{ fee } \boldsymbol{m} \end{aligned}$$

where $\boldsymbol{n}$ *and* $\boldsymbol{m}$ *are amount of Tez,* ***addr*** *is either a public key for an implicit account or a public hash for a smart contract,* $\boldsymbol{p}$ *is an argument passed to the smart contract's script, which is emtry if it is a transfer to an implicit account,* ***code*** *is a script of the smart contract,* $\boldsymbol{s}$ *is an initial value of the contract's storage.*

Definition 4. Queries *are defined by the following grammar:*

$$\begin{aligned} \boldsymbol{qry} ::= & \textit{ get balance for } \boldsymbol{addr} \\ \mid & \textit{ get status for } \boldsymbol{oph} \\ \mid & \textit{ get storage for } \boldsymbol{puh} \\ \mid & \textit{ get code for } \boldsymbol{puh} \\ \mid & \textit{ get counter for } \boldsymbol{puk} \end{aligned}$$

Definition 5. Programs *are defined by the following grammar:*

$$\begin{aligned} \boldsymbol{e} ::= & \ \boldsymbol{x} \mid \boldsymbol{v} \mid \lambda \boldsymbol{x} \boldsymbol{e} \\ \mid & \ \boldsymbol{e}\boldsymbol{e} \mid (\boldsymbol{e}, \boldsymbol{e}) \\ \mid & \ \boldsymbol{e} + \boldsymbol{e} \mid \boldsymbol{e} \text{ - } \boldsymbol{e} \mid \boldsymbol{e} = \boldsymbol{e} \mid \boldsymbol{e} < \boldsymbol{e} \\ \mid & \ \boldsymbol{e} \ \boldsymbol{and} \ \boldsymbol{e} \mid \boldsymbol{e} \ \boldsymbol{or} \ \boldsymbol{e} \mid \boldsymbol{not} \ \boldsymbol{e} \\ \mid & \ \boldsymbol{nil} \mid \boldsymbol{cons} \ \boldsymbol{e}\boldsymbol{e} \\ \mid & \ \lambda \boldsymbol{x} \boldsymbol{q} \boldsymbol{r} \boldsymbol{y} \end{aligned}$$

Definition 6. *The* state of a node *is a tuple* $\boldsymbol{N} = [\boldsymbol{E}, \boldsymbol{C}, \boldsymbol{O}]$*, where* $\boldsymbol{E}$ *is the set of the programs,* $\boldsymbol{C}$ *is a set of accounts, and* $\boldsymbol{O}$ *is a set of operations.*

1.2 Global

Definition 7. *A* manager *manages a single implicit account. It is represented by a tuple* $\langle \textbf{\textit{puk}}, \textbf{\textit{bal}}, \textbf{\textit{cou}} \rangle$*, where* $\textbf{\textit{puk}}$ *is the public key of an account,* $\textbf{\textit{bal}}$ *is its balance and* $\textbf{\textit{cou}}$ *is its counter whose form is a value-flag pair* (n, b)*, where* n*, which is called the value of the counter, is a natural number and* b*, which is called the flag of the counter, is a boolean value.*

Definition 8. *A* contractor *manages a smart contract. It is represented by a tuple* $\langle \textbf{\textit{puh}}, \textbf{\textit{bal}}, \textbf{\textit{code}}, \textbf{\textit{storage}} \rangle$*, where* $\textbf{\textit{puh}}$ *is the public hash of the contract,* $\textbf{\textit{bal}}$ *is its current balance,* $\textbf{\textit{code}}$ *is its code, and* $\textbf{\textit{storage}}$ *is its current storage.*

When an operation is injected in a node, it enters a *pending pool* (and is called a *pending operation*).

Definition 9. *A* pending operation *is a tuple* $\langle \textbf{\textit{op}}, \textbf{\textit{oph}}, \textbf{\textit{t}} \rangle$*, where* $\textbf{\textit{op}}$ *is an operation,* $\textbf{\textit{oph}}$ *is the operation hash, and* $\textbf{\textit{t}}$ *is the time when the operation was injected.*

After some time, a pending operation may be included in the blockchain as an accepted operation.

Definition 10. *An* accepted operation *is a tuple* $\langle \textbf{\textit{op}}, \textbf{\textit{oph}}, \textbf{\textit{t}}, \textbf{\textit{t}}' \rangle$*, where* $\textbf{\textit{op}}$ *is an operation,* $\textbf{\textit{oph}}$ *is the operation hash,* $\textbf{\textit{t}}$ *is the time when the operation was injected, and* $\textbf{\textit{t}}'$ *is the time when it was included in the blockchain.*

Definition 11. *The* state of a blockchain *is a tuple* $[\boldsymbol{P}, \boldsymbol{A}, \boldsymbol{M}, \boldsymbol{T}, \boldsymbol{t}]$ *where* $\boldsymbol{P}$ *is a set of pending operations,* $\boldsymbol{A}$ *is a set of accepted operations,* $\boldsymbol{M}$ *is a set of managers,* $\boldsymbol{T}$ *is a set of contractors, and* $\boldsymbol{t}$ *is the current time.*

Definition 12. *A* blockchain configuration *is a pair* $\boldsymbol{N} \| \boldsymbol{B}$ *where*

1. $\boldsymbol{N} = [\boldsymbol{C}, \boldsymbol{O}]$ *is the state of a node, and*
2. $\boldsymbol{B} = [\boldsymbol{P}, \boldsymbol{A}, \boldsymbol{M}, \boldsymbol{T}, \boldsymbol{t}]$ *is the state of a blockchain such that* $\forall c \in \boldsymbol{C} \implies \exists k \in \boldsymbol{M}, k.\textbf{\textit{puk}} = c.\textbf{\textit{puk}}$*.*

2 Rules

2.1 Transitions on Nodes

Each node has (nondeterministic) rules to propose an operation. When an operation **op** appears, we check that the account is local by looking up its public key **op**.**puk** in the local accounts **C** and consider it signed with the corresponding private key **pak**.

$$\textsc{Node-Op}\ \frac{\langle \mathbf{pak}, \mathbf{op.puk} \rangle \in \mathbf{C}}{[\mathbf{E}, \mathbf{C}, \mathbf{O}] \longrightarrow_N [\mathbf{E}, \mathbf{C}, \mathbf{op} :: \mathbf{O}]} \qquad \textsc{Node-Oph}\ \frac{}{[\epsilon[\mathbf{op}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \longrightarrow_N [\epsilon[\mathbf{oph}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}]}$$

$$\textsc{Node-System}\ \frac{\mathbf{N} \longrightarrow_N \mathbf{N}'}{\mathbf{N} :: \overline{\mathbf{N}} \| \mathbf{B} \longrightarrow \mathbf{N}' :: \overline{\mathbf{N}} \| \mathbf{B}} \qquad \textsc{Block-System}\ \frac{\mathbf{B} \longrightarrow_B \mathbf{B}'}{\overline{\mathbf{N}} \| \mathbf{B} \longrightarrow \overline{\mathbf{N}} \| \mathbf{B}'}$$

2.2 Transfers

Rule 1 [injected]:

$$\frac{\begin{array}{c}\text{chkBal}(\mathbf{M}, \mathbf{puk}, \mathbf{n}, \mathbf{m}) \qquad \text{chkCount}(\mathbf{M}, \mathbf{puk}) \qquad \text{chkPub}(\mathbf{M}, \mathbf{puk}') \\ \text{chkFee}(\mathbf{puk}, \mathbf{puk}', \mathbf{n}, \mathbf{m}) \qquad \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk}' \text{ arg } () \text{ fee } \mathbf{m} \\ \mathbf{oph} = \text{genOpHash}(\mathbf{puk}, \mathbf{puk}', \mathbf{n}, \mathbf{m}, \mathbf{t})\end{array}}{\begin{array}{c}[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow \\ [\mathbf{C}, \mathbf{O}] \| [\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{M}, \mathbf{puk}, \mathbf{True}), \mathbf{T}, \mathbf{t}]\end{array}}$$

Rule 2 [rejected of counter]:

$$\frac{\begin{array}{c}\neg\ \text{chkCount}(\mathbf{M}, \mathbf{puk}) \\ \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk}' \text{ arg } () \text{ fee } \mathbf{m}\end{array}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 3 [rejected of balance]:

$$\frac{\begin{array}{c}\neg\ \text{chkBal}(\mathbf{M}, \mathbf{puk}, \mathbf{n}, \mathbf{m}) \\ \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk}' \text{ arg } () \text{ fee } \mathbf{m}\end{array}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 4 [rejected of public key]:

$$\frac{\neg\ \text{chkPub}(\mathbf{M}, \mathbf{puk}') \qquad \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk}' \text{ arg } () \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 5 [rejected of fee]:

$$\frac{\neg\ \text{chkFee}(\mathbf{puk}, \mathbf{puk}', \mathbf{n}, \mathbf{m}) \quad \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk}' \text{ arg } () \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 6 [included]:

$$\frac{\mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk}' \text{ arg } () \text{ fee } \mathbf{m} \qquad \mathbf{t}' - \mathbf{t} < 60}{[\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}'] \longrightarrow_B [\mathbf{P}, \langle \mathbf{op}, \mathbf{oph}, \mathbf{t}, \mathbf{t}' \rangle :: \mathbf{A}, \text{updSucc}(\mathbf{M}, \mathbf{puk}, \mathbf{puk}', \mathbf{n}, \mathbf{m}), \mathbf{T}, \mathbf{t}' + 1]}$$

Rule 7 [timeout]: (applies to both, implicit transfers and contract invocations)

$$\frac{\mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{addr} \text{ arg } \mathbf{s} \text{ fee } \mathbf{m} \qquad \mathbf{t}' - \mathbf{t} \geq 60}{[\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}'] \longrightarrow_B [\mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{M}, \mathbf{puk}, \mathbf{False}), \mathbf{T}, \mathbf{t}']}$$

2.3 Smart Contracts

A. Originate

Rule 1 [injected]:

$$\frac{\begin{array}{c}\text{chkBal}(\mathbf{M}, \mathbf{puk}, \mathbf{n}, \mathbf{m}) \\ \text{chkCount}(\mathbf{M}, \mathbf{puk}) \qquad \text{chkFee}(\mathbf{code}, \mathbf{s}, \mathbf{n}, \mathbf{m}) \qquad \text{chkPrg}(\mathbf{code}, \mathbf{s}) \\ \mathbf{op} = \text{originate contract transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m} \\ \mathbf{oph} = \text{genOpHash}(\mathbf{puk}, \mathbf{code}, \mathbf{n}, \mathbf{m}, \mathbf{s}, \mathbf{t})\end{array}}{\begin{array}{c}[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \rightarrow \\ [\mathbf{C}, \mathbf{O}] \| [\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{M}, \mathbf{puk}, \mathbf{True}), \mathbf{T}, \mathbf{t}]\end{array}}$$

Rule 2 [rejected of code]:

$$\frac{\neg\ \text{chkPrg}(\mathbf{code}, \mathbf{s}) \quad \mathbf{op} = \text{originate contract transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \rightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 3 [rejected of counter]:

$$\frac{\neg\ \text{chkCount}(\mathbf{M}, \mathbf{puk}) \quad \mathbf{op} = \text{originate contract transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \rightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 4 [rejected of balance]:

$$\frac{\neg\ \text{chkBal}(\mathbf{M}, \mathbf{puk}, \mathbf{n}, \mathbf{m}) \quad \mathbf{op} = \text{originate contract transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \to [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 5 [rejected of fee]:

$$\frac{\neg\ \text{chkFee}(\mathbf{code}, \mathbf{s}, \mathbf{n}, \mathbf{m}) \quad \mathbf{op} = \text{originate contract transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \to [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 6 [included]:

Block-Accept

$$\frac{\mathbf{op} = \text{originate contract transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m} \quad \mathbf{t}' - \mathbf{t} < 60}{[\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}'] \longrightarrow [\mathbf{P}, \langle \mathbf{op}, \mathbf{oph}, \mathbf{t}, \mathbf{t}' \rangle :: \mathbf{A}, \text{updSucc}(\mathbf{M}, \mathbf{puk}, \mathbf{n}, \mathbf{m}), \langle \text{genHash}(\mathbf{code}, \mathbf{s}, \mathbf{t}, \mathbf{t}'), \mathbf{n}, \mathbf{code}, \mathbf{s} \rangle :: \mathbf{T}, \mathbf{t}' + 1]}$$

Rule 7 [timeout]:

Block-Timeout

$$\frac{\mathbf{op} = \text{originate contract transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m} \quad \mathbf{t}' - \mathbf{t} \geq 60}{[\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}'] \longrightarrow_B [\mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{M}, \mathbf{puk}, \mathbf{False}), \mathbf{T}, \mathbf{t}']}$$

B. **Transfer**

Rule 1 [injected]:

Block-Call

$$\frac{\begin{array}{c}\text{chkBal}(\mathbf{M}, \mathbf{puk}, \mathbf{n}, \mathbf{m}) \\ \text{chkCount}(\mathbf{M}, \mathbf{puk}) \quad \text{chkPuh}(\mathbf{T}, \mathbf{puh}) \quad \text{chkArg}(\mathbf{T}, \mathbf{puh}, \mathbf{p}) \\ \text{chkFee}(\mathbf{T}, \mathbf{puh}, \mathbf{p}, \mathbf{m}) \quad \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{p} \text{ fee } \mathbf{m} \\ \mathbf{oph} = \text{genOpHash}(\mathbf{puk}, \mathbf{puh}, \mathbf{p}, \mathbf{n}, \mathbf{m}, \mathbf{t})\end{array}}{\begin{array}{c}[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow \\ [\mathbf{C}, \mathbf{O}] \| [\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{M}, \mathbf{puk}, \mathbf{True}), \mathbf{T}, \mathbf{t}]\end{array}}$$

Rule 2 [rejected of counter]:

$$\frac{\neg\ \text{chkCount}(\mathbf{M}, \mathbf{puk}) \qquad \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{p} \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 3 [rejected of balance]:

$$\frac{\begin{array}{c}\neg\ \text{chkBal}(\mathbf{M}, \mathbf{puk}, \mathbf{n}, \mathbf{m}) \\ \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{p} \text{ fee } \mathbf{m}\end{array}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 4 [rejected of public hash]:

$$\frac{\neg\ \text{chkPuh}(\mathbf{T}, \mathbf{puh}) \qquad \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{p} \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 5 [rejected of argument]:

$$\frac{\neg\ \text{chkArg}(\mathbf{T}, \mathbf{puh}, \mathbf{p}) \qquad \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{p} \text{ fee } \mathbf{m}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 6 [rejected of fee]:

$$\frac{\begin{array}{c}\neg\ \text{chkFee}(\mathbf{T}, \mathbf{puh}, \mathbf{p}, \mathbf{m}) \\ \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{p} \text{ fee } \mathbf{m}\end{array}}{[\mathbf{C}, \mathbf{op} :: \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}] \longrightarrow [\mathbf{C}, \mathbf{O}] \| [\mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}$$

Rule 7 [included]:

$$\frac{\mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{p} \text{ fee } \mathbf{m} \qquad \mathbf{t}' - \mathbf{t} < 60}{\begin{array}{c}[\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}'] \longrightarrow_B \\ [\mathbf{P}, \langle \mathbf{op}, \mathbf{oph}, \mathbf{t}, \mathbf{t}' \rangle :: \mathbf{A}, \text{updSucc}(\mathbf{M}, \mathbf{puk}, '', \mathbf{n}, \mathbf{m}), \text{updConstr}(\mathbf{T}, \mathbf{puh}, \mathbf{n}, \mathbf{p}), \mathbf{t}' + 1]\end{array}}$$

3 Queries

Query 1 [balance of an account]:

$$\frac{\mathbf{B} = [\mathbf{P}, \mathbf{A}, \langle \mathbf{puk}, \mathbf{bal}, \mathbf{cou} \rangle :: \mathbf{M}, \mathbf{T}, \mathbf{t}]}{[\epsilon[\text{get balance for } \mathbf{puk}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B} \longrightarrow_N [\epsilon[\mathbf{bal}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B}}$$

Query 2 [balance of smart contract]:

$$\frac{\mathbf{B} = [\mathbf{P}, \mathbf{A}, \mathbf{M}, \langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle :: \mathbf{T}, \mathbf{t}]}{[\epsilon[\text{get balance for } \mathbf{puh}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B} \longrightarrow_N [\epsilon[\mathbf{bal}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B}}$$

Query 3 [storage]:

$$\frac{\mathbf{B} = [\mathbf{P}, \mathbf{A}, \mathbf{M}, \langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle :: \mathbf{T}, \mathbf{t}]}{[\epsilon[\text{get storage for } \mathbf{puh}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B} \longrightarrow_N [\epsilon[\mathbf{storage}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B}}$$

Query 4 [code]:

$$\frac{\mathbf{B} = [\mathbf{P}, \mathbf{A}, \mathbf{M}, \langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle :: \mathbf{T}, \mathbf{t}]}{[\epsilon[\text{get code for } \mathbf{puh}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B} \longrightarrow_N [\epsilon[\mathbf{code}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B}}$$

Query 5 [counter]:

$$\frac{\mathbf{B} = [\mathbf{P}, \mathbf{A}, \langle \mathbf{puk}, \mathbf{bal}, \mathbf{cou} \rangle :: \mathbf{M}, \mathbf{T}, \mathbf{t}]}{[\epsilon[\text{get counter for } \mathbf{puk}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B} \longrightarrow_N [\epsilon[\mathbf{cou}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B}}$$

Query 6 [status-pending]:

$$\frac{\mathbf{B} = [\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}{[\epsilon[\text{get status for } \mathbf{oph}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B} \longrightarrow_N [\epsilon[\mathbf{pending}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B}}$$

Query 7 [status-including]:

$$\frac{\mathbf{B} = [\mathbf{P}, \langle \mathbf{op}, \mathbf{oph}, \mathbf{t}, \mathbf{t}' \rangle :: \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}{[\epsilon[\text{get status for } \mathbf{oph}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B} \longrightarrow_N [\epsilon[\mathbf{including}] :: \mathbf{E}, \mathbf{C}, \mathbf{O}] \| \mathbf{B}}$$

4 Functions

1. Function chkPub(**puk**, **M**) checks whether a public key **puk** exists in **M**.
2. Function chkPuh(**puh**, **T**) checks whether a public hash **puh** exists in **T**.

3. Function chkBal(**M**, **puk**, **n**, **m**) checks whether the balance of the account that associates with the public key **puk** is greater or equal to **n** plus **m**.
4. Function chkCount(**M**, **puk**) checks whether the current counter of an account that associates with the public key **puk** is unlocked (its flag is False).
5. Function updSucc(**M**, **puk**, **puk**', **n**, **m**) updates the balance and the counter of the account that associates with the public key **puk** and the balance of the account that associates with the public key **puk**', where
   $\langle \mathbf{puk}, \mathbf{bal}, (\mathbf{n}, \mathbf{True}) \rangle \Rightarrow \langle \mathbf{puk}, \mathbf{bal} - \mathbf{n} - \mathbf{m}, (\mathbf{n}+1, \mathbf{False}) \rangle$
   $\langle \mathbf{puk'}, \mathbf{bal'}, \mathbf{cou'} \rangle \Rightarrow \langle \mathbf{puk'}, \mathbf{bal'} + \mathbf{n}, \mathbf{cou'} \rangle$
6. Function updCount(**M**, **puk**, **b**) updates the counter flag (True = locked, False = unlocked) of the account that associates with the public key **puk**, where
   $\langle \mathbf{puk}, \mathbf{bal}, (\mathbf{n}, \mathbf{b'}) \rangle \Rightarrow \langle \mathbf{puk}, \mathbf{bal}, (\mathbf{n}, \mathbf{b}) \rangle$
7. Function updConstr(**T**, **puh**, **n**, **p**) updates the contractor that associates with the public hash **puh** where
   $\langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage}, \rangle \Rightarrow \langle \mathbf{puh}, \mathbf{bal} + \mathbf{n}, \mathbf{code},$
   $\text{updStor}(\mathbf{storage}, \mathbf{code}, \mathbf{p}) \rangle$
8. Function chkPrg(**code**, **s**/**p**) checks whether the code **code** and the initil stogare's valude (or the input parameter **p**) are well type.
9. Function chkFee(...) checks whether the fee that is consumed to emit the opertion is less or equal to the fee **m**.
10. Function genOpHash(**puk**, **addr**/**code**, **n**, **m**, **s**/**p**, **t**) generates an operation hash.
11. Function genHash(**code**, **s**, **t**, **t**') generates the public hash of a smart contract.
12. Function updStor(**storage**, **code**, **p**) returns the new storage by running the code **code** on the storage **storage** with the input parameter **p**.