

# Blockchain System

## 1 Definitions

**Definition 1.** Let  $\mathbf{Elt}$  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  $\langle \mathbf{pak}, \mathbf{puk} \rangle$ , where  $\mathbf{pak}$  is its private key and  $\mathbf{puk}$  is its public key.

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

$\mathbf{op} ::= \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{addr} \text{ arg } \mathbf{p} \text{ fee } \mathbf{m}$   
| originate contract transferring  $\mathbf{n}$  from  $\mathbf{puk}$  running  $\mathbf{code}$  init  $\mathbf{s}$  fee  $\mathbf{m}$

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

**Definition 4.** The state of a node is a tuple  $\mathbf{N} = [\mathbf{C}, \mathbf{O}]$ , where  $\mathbf{C}$  is a set of accounts and  $\mathbf{O}$  is a set of operations.

### 1.2 Global

**Definition 5.** A manager manages a single implicit account. It is represented by a tuple  $\langle \mathbf{puk}, \mathbf{bal}, \mathbf{cou} \rangle$ , where  $\mathbf{puk}$  is the public key of an account,  $\mathbf{bal}$  is its balance and  $\mathbf{cou}$  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 6.** A contractor manages a smart contract. It is represented by a tuple  $\langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle$ , where  $\mathbf{puh}$  is the public hash of the contract,  $\mathbf{bal}$  is its current balance,  $\mathbf{code}$  is its code, and  $\mathbf{storage}$  is its current storage.

**Definition 7.** A query is defined by the following grammar:

$\mathbf{qry} ::= \text{get balance for } \mathbf{addr}$   
| get status for  $\mathbf{oph}$   
| get storage for  $\mathbf{puh}$   
| get code for  $\mathbf{puh}$   
| get counter for  $\mathbf{puk}$

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

**Definition 8.** A pending operation is a tuple  $\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle$ , where  $\mathbf{op}$  is an operation,  $\mathbf{oph}$  is the operation hash, and  $\mathbf{t}$  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 9.** An accepted operation is a tuple  $\langle \mathbf{op}, \mathbf{oph}, \mathbf{t}, \mathbf{t}' \rangle$ , where  $\mathbf{op}$  is an operation,  $\mathbf{oph}$  is the operation hash,  $\mathbf{t}$  is the time when the operation was injected, and  $\mathbf{t}'$  is the time when it was included in the blockchain.

**Definition 10.** The state of a blockchain is a tuple  $[P, A, M, T, t]$  where  $P$  is a set of pending operations,  $A$  is a set of accepted operations,  $M$  is a set of managers,  $T$  is a set of contractors, and  $t$  is the current time.

**Definition 11.** A blockchain configuration is a pair  $N \parallel B$  where

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

## 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**.

$$\begin{array}{ccc}
\text{NODE-OP} & \text{NODE-SYSTEM} & \text{BLOCK-SYSTEM} \\
\frac{\langle \mathbf{pak}, \mathbf{op.puk} \rangle \in \mathbf{C}}{[\mathbf{C}, \mathbf{O}] \longrightarrow_N [\mathbf{C}, \mathbf{op} :: \mathbf{O}]} & \frac{\mathbf{N} \longrightarrow_N \mathbf{N}'}{\mathbf{N} :: \bar{\mathbf{N}} \parallel \mathbf{B} \longrightarrow \mathbf{N}' :: \bar{\mathbf{N}} \parallel \mathbf{B}} & \frac{\mathbf{B} \longrightarrow_B \mathbf{B}'}{\bar{\mathbf{N}} \parallel \mathbf{B} \longrightarrow \bar{\mathbf{N}} \parallel \mathbf{B}'}
\end{array}$$

### 2.2 Transfers

Rule 1 [injected]:

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

Rule 2 [rejected of counter]:

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

Rule 3 [rejected of balance]:

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

Rule 4 [rejected of public key]:

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

Rule 5 [rejected of fee]:

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

Rule 6 [included]:

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

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

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

### 2.3 Smart Contracts

A. Originate

Rule 1 [injected]:

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

Rule 2 [rejected of code]:

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

Rule 3 [rejected of counter]:

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

Rule 4 [rejected of balance]:

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

Rule 5 [rejected of fee]:

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

Rule 6 [included]:

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

Rule 7 [timeout]:

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

## B. Transfer

Rule 1 [injected]:

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

Rule 2 [rejected of counter]:

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

Rule 3 [rejected of balance]:

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

Rule 4 [rejected of public hash]:

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

Rule 5 [rejected of argument]:

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

Rule 6 [rejected of fee]:

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

Rule 7 [included]:

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

### 3 Queries

Query 1 [balance of an account]:

$$\frac{\mathbf{qry} = \text{get balance for } \mathbf{puk} \quad \mathbf{B} = [\mathbf{P}, \mathbf{A}, \langle \mathbf{puk}, \mathbf{bal}, \mathbf{cou} \rangle :: \mathbf{M}, \mathbf{T}, \mathbf{t}]}{\mathbf{bal}}$$

Query 2 [balance of smart contract]:

$$\frac{\mathbf{qry} = \text{get balance for puh} \quad \mathbf{B} = [\mathbf{P}, \mathbf{A}, \mathbf{M}, \langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle :: \mathbf{T}, \mathbf{t}]}{\mathbf{bal}}$$

Query 3 [storage]:

$$\frac{\mathbf{qry} = \text{get storage for puh} \quad \mathbf{B} = [\mathbf{P}, \mathbf{A}, \mathbf{M}, \langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle :: \mathbf{T}, \mathbf{t}]}{\mathbf{storage}}$$

Query 4 [code]:

$$\frac{\mathbf{qry} = \text{get storage for puh} \quad \mathbf{B} = [\mathbf{P}, \mathbf{A}, \mathbf{M}, \langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle :: \mathbf{T}, \mathbf{t}]}{\mathbf{code}}$$

Query 5 [counter]:

$$\frac{\mathbf{qry} = \text{get counter for puk} \quad \mathbf{B} = [\mathbf{P}, \mathbf{A}, \langle \mathbf{puk}, \mathbf{bal}, \mathbf{cou} \rangle :: \mathbf{M}, \mathbf{T}, \mathbf{t}]}{\mathbf{cou}}$$

Query 6 [status-pending]:

$$\frac{\mathbf{qry} = \text{get status for oph} \quad \mathbf{B} = [\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}{\mathbf{Pending}}$$

Query 7 [status-including]:

$$\frac{\mathbf{qry} = \text{get status for oph} \quad \mathbf{B} = [\mathbf{P}, \langle \mathbf{op}, \mathbf{oph}, \mathbf{t}, \mathbf{t}' \rangle :: \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}]}{\mathbf{Including}}$$

## 4 Functions

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