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

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

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

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

1.2 Global

Definition 5. *A* manager *manages a single account. It is represented by a tuple* $\langle \boldsymbol{puk}, \boldsymbol{pkh}, \boldsymbol{bal}, \boldsymbol{cou} \rangle$*, where* $\boldsymbol{puk}$ *is the public key of an account,* $\boldsymbol{pkh}$ *is its public key hash,* $\boldsymbol{bal}$ *is its balance and* $\boldsymbol{cou}$ *is its counter whose form is a pair* (n, b)*, where* n *is a natural number and* b *is a boolean value.*

Definition 6. *A* contractor *manages a smart contract. It is represented by a tuple* $\langle \boldsymbol{puh}, \boldsymbol{bal}, \boldsymbol{code}, \boldsymbol{storage} \rangle$*, where* $\boldsymbol{puh}$ *is the public hash of the contract,* $\boldsymbol{bal}$ *is its current balance,* $\boldsymbol{code}$ *is its code, and* $\boldsymbol{storage}$ *is its current storage.*

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

$$\begin{aligned} \boldsymbol{qry} ::= &\ \textit{get balance for } \boldsymbol{pkh}/\boldsymbol{puh} \\ &\mid \textit{get status for } \boldsymbol{oph} \\ &\mid \textit{getStorage } \boldsymbol{puh} \\ &\mid \textit{get code for } \boldsymbol{puh} \\ &\mid \textit{get public key for } \boldsymbol{pkh} \\ &\mid \textit{get counter for } \boldsymbol{pkh} \end{aligned}$$

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* $\langle \boldsymbol{op}, \boldsymbol{oph}, \boldsymbol{t} \rangle$*, where* $\boldsymbol{op}$ *is an operation,* $\boldsymbol{oph}$ *is the operation hash, and* $\boldsymbol{t}$ *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* $\langle \boldsymbol{op}, \boldsymbol{oph}, \boldsymbol{t} \rangle$*, where* $\boldsymbol{op}$ *is an operation,* $\boldsymbol{oph}$ *is the operation hash, and* $\boldsymbol{t}$ *is the time when it was included in the blockchain.*

Definition 10. *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 11. *A* blockchain configuration *is a pair* $\boldsymbol{N} \| \boldsymbol{B}$ *where*

1. $\boldsymbol{N} = [\boldsymbol{C}, \boldsymbol{O}, \boldsymbol{S}]$ *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* $\forall c \in \boldsymbol{C} \implies \exists k \in \boldsymbol{M}, k.\boldsymbol{pkh} = c.\boldsymbol{pkh}$ *and* $\forall s \in \boldsymbol{S} \implies \exists p \in \boldsymbol{T}, s.\boldsymbol{puh} = p.\boldsymbol{puh}$.

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**.

$$\textsc{Node-Op}\quad \frac{\langle \mathbf{pak}, \mathbf{op}.\mathbf{puk}\rangle \in \mathbf{C}}{[\mathbf{C}, \mathbf{O}] \longrightarrow_N [\mathbf{C}, \mathbf{op} :: \mathbf{O}]} \qquad \textsc{Node-System}\quad \frac{\mathbf{N} \longrightarrow_N \mathbf{N}'}{\mathbf{N} :: \overline{\mathbf{N}} \| \mathbf{B} \longrightarrow \mathbf{N}' :: \overline{\mathbf{N}} \| \mathbf{B}} \qquad \textsc{Block-System}\quad \frac{\mathbf{B} \longrightarrow_B \mathbf{B}'}{\overline{\mathbf{N}} \| \mathbf{B} \longrightarrow \overline{\mathbf{N}} \| \mathbf{B}'}$$

2.2 Transfers

Rule 2 [injected]:

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

Rule 3 [rejected of counter]:

$$\frac{\neg\ \mathrm{checkCou}(\mathbf{M},\ pkh)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle \rightarrow \\ \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{1}$$

Rule 4 [rejected of balance]:

$$\frac{\neg\ \mathrm{checkBal}(\mathbf{M},\ pkh, m, n)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle \rightarrow \\ \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{2}$$

Rule 5 [rejected of public key]:

$$\frac{\neg\ \mathrm{checkPub}(\mathbf{M},\ pkh')}{\begin{array}{l}\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle \rightarrow \\ \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{3}$$

Rule 6 [included]:

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

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

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

2.3 Smart Contracts

A. Originate

Rule 2 [injected]:

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

Rule 3 [rejected of code]:

$$\frac{\neg\ \text{checkPrg}(code, s)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) \\ ::\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle \rightarrow \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{4}$$

Rule 4 [rejected of counter]:

$$\frac{\neg\ \text{checkCou}(\mathbf{M},\ pkh)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s)) \\ ::\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle \rightarrow \langle\ [\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{5}$$

Rule 5 [rejected of balance]:

$$\frac{\neg\ \text{checkBal}(\mathbf{M},\ pkh, n, m)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) \\ ::\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle \to \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{6}$$

Rule 6 [included]:

Block-Accept

$$\frac{\begin{array}{c}\mathbf{op} = \text{originate contract } \mathbf{id} \text{ transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m} \\ \mathbf{t}' - \mathbf{t} < 60\end{array}}{\begin{array}{c}\langle[\mathbf{C}, \mathbf{O}, \mathbf{S}], [\langle\mathbf{op}, \mathbf{oph}, \mathbf{t}\rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}']\rangle \longrightarrow \qquad \langle[\mathbf{C}, \mathbf{O}, \langle\mathbf{id}, \mathbf{oph}, \mathbf{code}\rangle :: \mathbf{S}], \\ [\mathbf{P}, \langle\mathbf{op}, \mathbf{oph}, \mathbf{t}, \mathbf{t}'\rangle :: \mathbf{A}, \text{updSucc}(\mathbf{M}, \mathbf{puk}, \mathbf{n}, \mathbf{m}), \langle\mathbf{oph}, \text{getStorage } (\mathbf{code}, \mathbf{s})\rangle :: \mathbf{T}, \mathbf{t}' + 1]\rangle\end{array}}$$

Rule 7 [timeout]:

Block-Timeout

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

B. **Transfer**

Rule 2 [injected]:

Block-Call

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

Rule 3 [rejected of counter]:

$$\frac{\neg\ \text{checkCou}(\mathbf{M},\ pkh)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m)\ ::\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle \\ \to \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{7}$$

Rule 4 [rejected of balance]:

$$\frac{\neg\ \text{checkBal}(\mathbf{M},\ pkh, n, m)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m)\ ::\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle \\ \to \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{8}$$

Rule 5 [rejected of public hash]:

$$\frac{\neg\ \text{checkPuh}(\mathbf{T},\ puh)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\\ \to \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{9}$$

Rule 6 [rejected of argument]:

$$\frac{\neg\ \text{checkArg}(\mathbf{T},\ puh, s)}{\begin{array}{l}\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\\ \to \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{M},\ \mathbf{T},\ \mathbf{t}]\rangle\end{array}} \tag{10}$$

Rule 7 [included]:

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

Rule 8 [timeout]:

$$\frac{\mathbf{t}' - \mathbf{t} \geq 60}{\begin{array}{l}[\langle \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{s} \text{ fee } \mathbf{m}, \mathbf{t}\rangle :: \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{T}, \mathbf{t}'] \to\\ [\mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{M}, \mathbf{puk}, \mathbf{False}), \mathbf{T}, \mathbf{t}']\end{array}} \tag{11}$$

3 Functions

1. Function `checkAcc(pkh, C)` checks whether an account *pkh* exists in **C**
2. Function `checkPub(K, pkh)` checks whether the public key of the public key hash *pkh* is reveled to the blockchain.
3. Function `checkBal(K, pkh, n, m)` checks whether the balance of the account *pkh* is greater or equal to $m + n$
4. Function `checkCou(K, pkh)` checks whether the current counter of an account *phk* is unlocked (i.e., its flag is False)
5. Function `updateSucc(K, pkh, pkh', n, m)` updates the balance and the counter of the account *phk* and the balance of the account *phk*′, where
   – `< puk, pkh, bal, (n, True) > =>`
     `< puk, pkh, bal - n - m, (n + 1, False) >`
   – `< puk', pkh', bal', cou' > => < puk', pkh', bal' + n, cou' >`
6. Function `updateCou(K, puk, b')` updates the counter lock of the account *phk* (True = lockeed, False = unlocked), where

  - `< puk, pkh, bal, (n, b) > => < puk, pkh, bal, (n, b') >`

7. Function `checkId(id, S)` checks whether a contract *id* does not already exist in **S**
8. Function `checkPrg(code, s)` checks whether the code *code* are well type and s is well type input
9. Function `generateOph(pkh,pkh', n, m, t)` generates a operation hash
10. Function `generateHash(S, id, puh, code, t)` generates the public hash of a contract
11. Function `getStorage(code, s)` gets the storage for the code *code* and the input s

4 Some implementations

Function `checkAcc(puh, C)` checks whether an account exists and and `checkPuk(puh, K)` checks the revelation of its public key to the blockchain.

```
let rec checkAcc puh C =
  match C with
  | 0 -> false
  | < als, pak, puk, pkh' > :: C' ->
    if (puh = puh')  then true
    else checkAcc (puh, C')

let rec checkPuk puh K =
  match C with
  | 0 -> false
  | < als, pak, puk, pkh' > :: K' ->
    if (puh = puh') and (puk =/= nil) then true
    else 5checkPuk (puh, K')
```

The following functions interact with **M**.

```
let rec checkBal K puk n m =
  match K with
  | 0 -> true
  | < puk', bal, cou > :: K' ->
    if (puk = puk') and (n + m) <= bal then true
    else checkBal (K', puk, n, m)

let rec checkPub K puk =
  match K with
  | 0 -> false
  | < puk', bal, cou > :: K' ->
    if (puk = puk') then true
    else checkExi (K', puk)
```

```
let rec checkCou K puk =
  match K with
  | 0 -> false
  | < puk', bal, cou > :: K' ->
    if (puk = puk') and (cou = T) then true
    else checkCou (K', puk)

let rec updateCou K puk =
  match K with
  | 0 -> 0
  | < puk', bal, cou > :: K' ->
    if (puk = puk') then < puk', bal, F > :: K'
    else < puk', bal, cou > :: updateCou (K', puk)

let rec updateSucc K puk puk' m n =
  match K with
  | 0 -> 0
  | < puk'', bal, cou > :: K' ->
    if (puk = puk'') then < puk'', bal - (n + m), T >
       :: updateSucc (K', puk, puk', n, m)
    else if (puk' = puk'') then < puk'', bal + n, cou > :: K'
         else  < puk'', bal, cou >
               ::updateSucc (K', puk, puk', n, m)
```