runtime verification

K-ESDT

ESDT Semantics in K

Burak Yalcinkaya

Runtime Verification Inc.

Overview

runtime verification

- Configuration
- Constructors
- Shards and the main loop
- Transfer function
- Error handling
- Concrete tests
- Symbolic tests

CONFIGURATION.

Configuration

```
<esdt>
    <is-running> #no:Running </is-running>
    <meta>
        ...
    </meta>
    <shards>
        <shard multiplicity="*" type="Map">
            <shard-id> 0:Int </shard-id>
            ...
        </shard>
    </shards>
</esdt>
```

Metachain:

- Token meta-data
- Token properties
- ESDT system SC

Shard:

- Local token settings
- Account database

Configuration

```
<shard multiplicity="*" type="Map">
    <shard-id> 0:Int </shard-id>
    <accounts>
        <account multiplicity="*" type="Map">
            <account-name> "":AccountName </account-name>
            ...
        </account>
    </accounts>
    <token-settings>
       <token-setting multiplicity="*" type="Map">
       ...
       </token-setting>
    </token-settings>

    <user-txs> .TxList </user-txs>
    <incoming-txs> .MQueue </incoming-txs>
    <out-txs> .TxList </out-txs>
    ...
</shard>
```

Balances
ESDT roles
User meta-data

Frozen accounts
Paused, limited

User-created transactions (input) (queue)

System-generated transactions (input) (multi-queue)

System-generated transactions (output) (queue)

Configuration - transaction queues

```
<shard>
   <shard-id> ShardC </shard-id>
   <user-txs>
       TxU1 TxU2 TxU3
   </user-txs>
   <incoming-txs>
       ShardA M|-> TxA1 TxA2 TxA3
       ShardB M|-> TxB1 TxB2
   </incoming-txs>
   ...
</shard>
```

Transactions created by users in this shard

Transactions sent from ShardA

Transactions sent from ShardB

Configuration

```
<meta>
    <global-token-settings >
        <global-token-setting multiplicity="*" type="Map">
            <global-token-id>     0:TokenId </global-token-id>
            ...
        </global-token-setting>
    </global-token-settings>

    <meta-incoming> .MQueue </meta-incoming>
    <meta-out-txs> .TxList </meta-out-txs>
</meta>
```

Token meta-data
Owner, paused, limited, properties, roles

System-generated transactions (input) (multi-queue)

System-generated transactions (output) (queue)

CONSTRUCTORS.

Constructors - IDs

```
syntax ShardId ::= Int
                 | "#metachainShardId"

syntax AccountName ::= String
syntax AccountAddr ::= addr(ShardId, AccountName )

syntax TokenId ::= Int
```

Constructors - Transactions

runtime verification

```
syntax Transaction ::= BuiltinCall
                     | ESDTManage

syntax ESDTManage
    ::= "issue" "(" AccountAddr "," TokenId "," Int ")" Properties
      | freeze( AccountAddr , AccountAddr , TokenId ,Bool )
      | ...

syntax BuiltinCall
    ::= transfer( AccountAddr, AccountAddr, TokenId,Int, Bool )
      | localMint( AccountAddr, TokenId,Int )
      | localBurn( AccountAddr, TokenId,Int )
      | ...
```

Constructors - Transactions

runtime verification

```
syntax Transaction ::= BuiltinCall
                     | ESDTManage

syntax ESDTManage
    ::= "issue" "(" AccountAddr "," TokenId "," Int ")" Properties
      | freeze( AccountAddr , AccountAddr , TokenId ,Bool )
      | ...

syntax BuiltinCall
    ::= transfer( AccountAddr, AccountAddr, TokenId,Int, Bool )
      | localMint( AccountAddr, TokenId,Int )
      | localBurn( AccountAddr, TokenId,Int )
      | ...
```

```
transfer( addr(1, "Alice"), addr(2, "Bob"), 2, 20, false )
```

runtime verification

SHARDS and the MAIN LOOP.

Shards

runtime verification

Main loop:

1. Choose a transaction to execute (nondeterministically)
   - A user-created transaction from <user-txs>, or
   - An internal transaction from <incoming-txs>.

Shards

runtime verification

Main loop:

1. Choose a transaction to execute (nondeterministically)
   - A user-created transaction from <user-txs>, or
   - An internal transaction from <incoming-txs>.

```
<user-txs> TxL( TxU1 ) TxL( TxU2 ) </user-txs>
<incoming-txs>
   ShardA M|-> TxL( TxA1 ) TxL( TxA2 )
   ShardB M|-> TxL( TxB1 ) TxL( TxB2 ) TxL( TxB3 )
<incoming-txs>
```

Shards

runtime verification

Main loop:

1. Choose a transaction to execute (nondeterministically)
   - A user-created transaction from <user-txs>, or
   - An internal transaction from <incoming-txs>.

```
<user-txs> TxL( TxU1 ) TxL( TxU2 ) </user-txs>
<incoming-txs>
   ShardA M|-> TxL( TxA1 ) TxL( TxA2 )
   ShardB M|-> TxL( TxB1 ) TxL( TxB2 ) TxL( TxB3 )
<incoming-txs>
```

TxU1, TxA1, or TxB1 will be chosen

Shards

runtime verification

Main loop:

1. Choose a transaction to execute (nondeterministically)
   - A user-created transaction from <user-txs>, or
   - An internal transaction from <incoming-txs>.
2. Execute the transaction
   - ESDTManage: directly forward to Metachain
   - BuiltinCall:
     1. Take a snapshot and execute the function
     2. If there is an error:
        revert to the snapshot,
        If cross-shard, create a return transaction

Shards

runtime verification

Main loop:

1. Choose a transaction to execute (nondeterministically)
   - A user-created transaction from <user-txs>, or
   - An internal transaction from <incoming-txs>.
2. Execute the transaction
   - ESDTManage: directly forward to Metachain
   - BuiltinCall:
     1. Take a snapshot and execute the function
     2. If there is an error:
        revert to the snapshot,
        If cross-shard, create a return transaction
3. Log the result
4. Send internal transactions if there are any
5. Finalize the transaction

Shards

runtime verification

Main loop:

1. Choose a transaction to execute (nondeterministically)
   - A user-created transaction from <user-txs>, or
   - An internal transaction from <incoming-txs>.
2. Execute the transaction
   - ESDTManage: directly forward to Metachain
   - BuiltinCall:
     1. Take a snapshot and execute the function
     2. If there is an error:
        revert to the snapshot,
        If cross-shard, create a return transaction
3. Log the result
4. Send internal transactions if there are any
5. Finalize the transaction

Inside a mutex lock

- <is-running>
- reduce branching & state space

Shards

runtime verification

These cases require sending out an internal transaction to another shard:

- Cross-shard transfer successfully executed in the sender's shard:
  - Send the original transaction to the receiver's shard
- Cross-shard transfer failed in the receiver's shard:
  - Send the tokens back to the sender.
- User submitted an ESDTManage transaction:
  - Forward the transaction to Metachain.

Shards

The rule for sending transactions:

```
rule <shard>
       <steps> #finalizeTransaction </steps>
       <shard-id> SndShrId </shard-id>
       <out-txs> TxL(Tx) => .TxList ... </out-txs>
       ...
     </shard>
     <shard>
       <shard-id> DestShrId </shard-id>
       <incoming-txs>
         MQ => push(MQ, SndShrId, Tx)
       </incoming-txs>
       ...
     </shard>
     requires DestShrId ==Shard #txDestShard(Tx)
     [label(relay-cross-shard)]
```

When finalizing the transaction

Take the first Tx in <out-txs>

Push to the destination shard's

<incoming-txs> queue

Shards

runtime verification

Example:

```
<shard>
   <steps> #finalizeTransaction </steps>
   <shard-id> ShardA </shard-id>
   <out-txs> TxL(TxA3) </out-txs>
   …
</shard>
<shard>
   <shard-id> ShardB </shard-id>
   <incoming-txs>
     ShardA M|-> TxL(TxA1) TxL(TxA2)
     ShardC M|-> TxL(TxC1) TxL(TxC2)
   </incoming-txs>
   …
</shard>
```

2 separate queues

=>

```
<shard>
   <steps> #finalizeTransaction </steps>
   <shard-id> ShardA </shard-id>
   <out-txs> .TxList </out-txs>
   …
</shard>
<shard>
   <shard-id> ShardB </shard-id>
   <incoming-txs>
     ShardA M|-> TxL(TxA1) TxL(TxA2) TxL(TxA3)
     ShardC M|-> TxL(TxC1) TxL(TxC2)
   </incoming-txs>
   …
</shard>
```

Empty list

TRANSFER.

Transfer function

```
transfer( Sender, Recv, TokId, Amount, IsReturn )
```

address | token ID (int) | int | boolean

1. Basic checks (arguments, limited transfer etc.)
2. Process sender
   a. Check frozen/paused/balance
   b. Decrease the balance
3. Process destination
   a. Check payable/frozen/paused
   b. Increase the balance

Transfer function

```
transfer( Sender, Recv, TokId, Amount, IsReturn )
```

address | token ID (int) | int | boolean

1. Basic checks (arguments, limited transfer etc.)
2. Process sender
   a. Check frozen/paused/balance
   b. Decrease the balance
3. Process destination
   a. Check payable/frozen/paused
   b. Increase the balance

```
rule <shard>
     <steps> #processBuiltinFunction
          => #createDefaultTokenSettings(TokId)
          ~> #basicChecks
          ~> #checkLimitedTransfer
          ~> #processSender
          ~> #processDest ...
     </steps>
     <current-tx> transfer(_,_,TokId,_,_) </current-tx>
     ...
   </shard>
```

Transfer function - cross-shard

runtime verification

```
transfer( Sender, Recv, TokId, Amount, IsReturn )
```

address | token ID (int) | int | boolean

1. Basic checks (arguments, limited transfer etc.)
2. Process sender → Skip in the destination's shard
   a. Check frozen/paused/balance
   b. Decrease the balance
3. Process destination → Skip in the sender's shard
   a. Check payable/frozen/paused
   b. Increase the balance

Transfer function - process sender

runtime verification

If this is the sender's shard, update the balance.

```
rule
 <shard>
   <shard-id> ShrId </shard-id>
   <steps> #processSender
        => #checkTokenSettings(TokId, ActName)
        ~> #updateBalance(ActName, TokId,0 -Int Val)
         ...
   </steps>
   <current-tx>
     transfer(addr ShrId, ActName), _, TokId, Val, _)
   </current-tx>
   ...
 </shard>
```

If this is not the sender's shard, skip.

```
rule
  <shard>
    <shard-id> ShrId </shard-id>
    <steps> #processSender => . ... </steps>
    <current-tx> Tx </current-tx>
    ...
  </shard>
  requires ShrId =/=Shard #txSenderShard(Tx)
```

Transfer function - process destination

If this is the receiver's shard, update the balance.

```
rule
 <shard>
   <shard-id> ShrId </shard-id>
   <steps> #processDest
        => #checkPayable
        ~> #checkTokenSettings(TokId, ActName)
        ~> #updateBalance(ActName, TokId, Val)
         ...
   </steps>
   <current-tx>
     transfer(_, addr(ShrId, ActName), TokId, Val, _)
   </current-tx>
   ...
 </shard>
```

If this is not the receiver's shard, create an internal tx.

```
rule
 <shard>
     <shard-id> ShrId </shard-id>
     <steps> #processDest => . ... </steps>
     <current-tx> Tx </current-tx>
     <out-txs> ... (.TxList => TxL(Tx)) </out-txs>
     ...
 </shard>
 requires ShrId =/=Shard #txDestShard(Tx)
```

runtime verification

ERROR HANDLING.

Error handling

runtime verification

In case of an error:

1. Skip remaining steps.
2. Revert to the snapshot.
3. If cross-shard,
   send a tx to the sender.

```
rule
 <shard>
   <steps> #checkTokenSettings(TokId, ActName)
        => #failure(#ErrESDTIsFrozenForAccount) ...
   </steps>
   <token-settings>
     <token-setting>
       <token-setting-id> TokId </token-setting-id>
       <frozen> Frozen </frozen>
       ...
     </token-setting>
     ...
   </token-settings>
   ...
 </shard> requires ActName in Frozen
```

Error handling

In case of an error:

1. Skip remaining steps.
2. Revert to the snapshot.
3. If cross-shard,
   send a tx to the sender.

```
rule <steps> #failure(_) ~> (T:TxStep => .) ... </steps>
     requires T =/=K #finalizeTransaction

rule
  <shard>
    <shard-id> ShrId </shard-id>
    <steps> (#failure(Err) => .) ~> #finalizeTransaction </steps>
    <current-tx> Tx </current-tx>
    <snapshot> ACTS </snapshot>
    (_:AccountsCell => ACTS)
    <logs> L => (L ; #failure(Err) ; Tx ) </logs>
    <out-txs> Txs => Txs TxL(#mkReturnTx(Tx)) </out-txs>
    ...
  </shard>
  requires #isCrossShard(Tx) andBool (#txDestShard(Tx) ==Shard ShrId)
```

Error handling

In case of an error:

1. Skip remaining steps.
2. Revert to the snapshot.
3. If cross-shard,
   send a tx to the sender.

```
syntax Transaction ::= #mkReturnTx(Transaction)        [function, total]
// ----------------------------------------------------------------------
rule #mkReturnTx(transfer(Sender, Dest, TokId, Val, _))
                => transfer(Dest, Sender, TokId, Val, true)
```

flip Sender and Dest, set IsReturn

IsReturn = true skips all the checks

CONCRETE TESTS.

Concrete tests

runtime verification

- Testing module: https://github.com/runtimeverification/elrond-esdt/blob/development/tests/concrete/tester.k
- Issue: https://github.com/runtimeverification/elrond-esdt/blob/development/tests/concrete/issue.test
- Transfer: https://github.com/runtimeverification/elrond-esdt/blob/development/tests/concrete/in-shard-transfer.test
- Limited: https://github.com/runtimeverification/elrond-esdt/blob/development/tests/concrete/limited.test

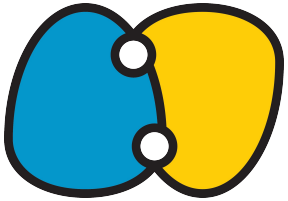
SYMBOLIC TESTS.

Symbolic tests

runtime verification

- Simple: https://github.com/runtimeverification/elrond-esdt/blob/development/tests/specs/simple-spec.k
- Total balance: https://github.com/runtimeverification/elrond-esdt/blob/development/tests/specs/total-balance-cross-spec.k

runtime verification



Questions?

https://runtimeverification.com/

@rv_inc

https://discord.com/invite/CurfmXNtbN

contact@runtimeverification.com