Model-based Testing for Crosschain Token Transfers
Crosschain Workshop, March 2021

Andrey Kupriianov
Senior Research Engineer @ Informal Systems
Who We Are

We envision an open-source ecosystem of cooperatively owned and governed distributed organizations running on reliable distributed systems.
Who We Are

Blockchain Infrastructure
- Tendemint-rs
- ibc-rs

Formal Verification Tools
- Apalache

Business Operations Tools
- Themis Contract

Our Infrastructure powers the CØSMØS Network
COSMOS
INTERNET OF BLOCKCHAINS

https://cosmos.network
by @zcpeng
CAPRI team at Informal

Correctness Assurance and Protocol Re-Invention

• Verification-Driven Development (VDD)
  – Guidelines and specifications for rigorous software development
  – https://github.com/informalsystems/vdd

• Apalache: symbolic model checker for TLA+
  – Efficient model checker with unique feature (e.g. type checking)
  – https://apalache.informal.systems/

• Protocol design & auditing
  – https://github.com/informalsystems/audits

• Model-based testing
  – Connecting English specs, formal TLA+ specs and real implementations
  – Special thanks to Shivani Joshi & Vitor Enes
IBC ICS-20 Fungible Token Transfer

- **Critical:**
  - The first and most-important IBC application
  - one of the main focus points in the Informal Systems IBC Audit

- **Complex:** ADR 001 Coin Source Tracing

In this context, upon a receive of a cross-chain fungible token transfer, if the sender chain is the source of the token, the protocol prefixes the denomination with the port and channel identifiers in the following format:

\[ \text{prefix + denom} = \{\text{destPortN}\}/\{\text{destChannelN}\}/.../\{\text{destPort0}\}/\{\text{destChannel0}\}/\text{denom} \]

Example: transferring \(100\ \text{uatom}\) from port `HubPort` and channel `HubChannel` on the Hub to Ethermint's port `EthermintPort` and channel `EthermintChannel` results in `100 EthermintPort/EthermintChannel/uatom`, where `EthermintPort/EthermintChannel/uatom` is the new denomination on the receiving chain.

The new proposed format will be the following:

\[ \text{ibcDenom} = "ibc/" + \text{hash(\text{trace path + "/" + base denom})} \]
Token Transfer: standard testing

```go
func (suite *TransferTestSuite) TestHandleMsgTransfer() {
    channelA, channelB := suite.coordinator.CreateTransferChannels(suite.chainA, suite.chainB, connA, connB, channeltypes.UNORDERED)
    timeoutHeight := clienttypes.NewHeight(0, 100)

    coinToSendToB := sdk.NewCoin(sdk.DefaultBondDenom, sdk.NewInt(100))

    // send from chainA to chainB
    msg := types.NewMsgTransfer(channelA.PortID, channelA.ID, coinToSendToB, suite.chainA.SenderAccount.GetAddress(), suite.chainB.SenderAccount.GetAddress())
    err := suite.coordinator.SendMsg(suite.chainA, suite.chainB, clientB, msg)
    suite.Require().NoError(err) // message committed

    // relay send
    fungibleTokenPacket := types.NewFungibleTokenPacketData(coinToSendToB.Denom, coinToSendToB.Amount.Uint64(), suite.chainA.SenderAccount.GetAddress().String(), channelA.PortID, channelA.ID)
    packet := channeltypes.NewPacket(fungibleTokenPacket.GetBytes(), 1, channelB.PortID, channelB.ID, timeoutHeight, 0)
    ack := channeltypes.NewResultAcknowledgement([0]byte{4})
    suite.Require().NoError(err) // relay committed

    // check that voucher exists on chain B
}
```
Token Transfer: standard testing

coinToSendBackToA := types.GetTransferCoin(channelB.PortID, channelB.ID, sdk.DefaultBondDenom, 100)
suite.Require().Equal(coinToSendBackToA, balance)

// send from chainB back to chainA
msg = types.NewMsgTransfer(channelB.PortID, channelB.ID, coinToSendBackToA, suite.chainB.SenderAccount.GetAddress(), suite.chainA.SenderAccount.GetAddress())

err = suite.coordinator.SendMsg(suite.chainB, suite.chainA, clientA, msg)
suite.Require().NoError(err) // message committed

// relay send
// NOTE: fungible token is prefixed with the full trace in order to verify the packet commitment
voucherDenom := voucherDenomTrace.GetPrefix() + voucherDenomTrace.BaseDenom
fungibleTokenPacket = types.NewFungibleTokenPacketData(voucherDenom, coinToSendBackToA.Amount.Uint64(), suite.chainB.SenderAccount.GetAddress().String(),
packet = channeltypes.NewPacket(fungibleTokenPacket.GetBytes(), 1, channelB.PortID, channelB.ID, channelA.PortID, channelA.ID, timeoutHeight, 0)
suite.Require().NoError(err) // relay committed


// check that the balance on chainA returned back to the original state
suite.Require().Equal(originalBalance, balance)

// check that module account escrow address is empty
escrowAddress := types.GetEscrowAddress(packet.GetDestPort(), packet.GetDestChannel())

// check that balance on chain B is empty
suite.Require().Zero(balance.Amount.Int64())
Token Transfer: model-based testing

We have developed:

- TLA+ model of the token transfer relay functions
- Set of simple model-based tests (also in TLA+)
- Test driver (in Go)

Allowed us to catch the receiving denom bug: Cosmos SDK #8120
Token Transfer: changes to the standard test

```go
group := suite.mgr.Group() // setup suite

func (suite *TransferTestSuite) SetupTest() {
    suite.coordinate = ibctesting.NewCoordinator(suite.T(), 2)
}

func (suite *TransferTestSuite) TestHandleMsgTransfer() {
    suite.coordinate = ibctesting.NewCoordinator(suite.T(), 3)
    suite.chainA = suite.coordinate.GetChain(ibctesting.GetChainID(0))
    suite.chainB = suite.coordinate.GetChain(ibctesting.GetChainID(1))
    suite.chainC = suite.coordinate.GetChain(ibctesting.GetChainID(2))
}

// constructs a send from chainA to chainB on the established channel/connection
// and sends the same coin back from chainB to chainA.

func (suite *TransferTestSuite) TestHandleMsgTransfer() {
    channelA, channelB := suite.coordinate.CreateTransferChannels(suite.chainA, suite.chainB, connA, connB, channeltypes.ChannelTypeTransfer)
    timeoutHeight := clientA.NewHeight(0, 110)
    coinToSendToB := sdk.NewCoin(sdk.DefaultBondDenom, sdk.NewInt(100))

    coinToSendBackToA := types.GetTransferCoin(channelB.PortID, channelB.ID, sdk.DefaultBondDenom, 100)
    suite.Require().Equal(coinToSendBackToA, balance)
    coinSentFromAtoB := types.GetTransferCoin(channelB.PortID, channelB.ID, sdk.DefaultBondDenom, 100)
    suite.Require().Equal(coinSentFromAtoB, balance)
}
```

Token Transfer: changes to the standard test

```
67    + channelOnBForC, channelOnCForB := suite.coordinator.CreateTransferChannels(suite.chainB, suite.chainC, connOnBForC, connOnCForB)
65    - err = suite.coordinator.SendMsg(suite.chainB, suite.chainA, clientA, msg)
66    + // send from chainB to chainC
67    + msg = types.NewMsgTransfer(channelOnBForC.PortID, channelOnCForB.ID, coinSentFromAtoB, suite.chainB.SenderAccount.GetAddress(), suite.chainC.SenderAccount.GetAddress(), msg)
68    + err = suite.coordinator.SendMsg(suite.chainB, suite.chainC, clientOnCForB, msg)
69    + suite.Require().NoError(err) // message committed
70    + // relay send
71    + // NOTE: fungible token is prefixed with the full trace in order to verify the packet commitment
72    + fullDenomPath := types.GetPrefixesDenom(channelOnCForB.PortID, channelOnCForB.ID, voucherDenomTrace.GetFullDenomPath(), suite.chainC.SenderAccount.GetAddress(), coinSentFromAtoB)
73    + fungibleTokenPacket = types.NewFungibleTokenPacketData(voucherDenomTrace.GetFullDenomPath(), fullDenomPath, coinSentFromAtoB.Amount).getBytes()
74    + packet = channeltypes.NewPacket(fungibleTokenPacket.GetBytes(), 1, channelOnBForC.PortID, channelOnBForC.ID, channelOnCForB.PortID, channelOnCForB.ID)
76    + suite.Require().NoError(err) // relay committed
77    + coinSentFromBtoC := sdk.NewInt64Coin(types.ParseDenomTrace(fullDenomPath).IBCDenom(), 100)
79    + // check that the balance is updated on chainC
80    + suite.Require().Equal(coinSentFromBtoC, balance)
81    + // check that balance on chain B is empty
83    + suite.Require().Zero(balance.Amount.Int64())
84    + // send from chainC back to chainB
85    + msg = types.NewMsgTransfer(channelOnCForB.PortID, channelOnBForB.ID, coinSentFromBtoC, suite.chainC.SenderAccount.GetAddress(), suite.chainC.SenderAccount.GetAddress(), msg)
86    + err = suite.coordinator.SendMsg(suite.chainC, suite.chainB, clientOnBForB, msg)
87    + suite.Require().NoError(err) // message committed
```
Token Transfer: changes to the standard test

```go
// relay send
// NOTE: fungible token is prefixed with the full trace in order to verify the packet commitment

fusableTokenPacket = types.NewFungibleTokenPacketData(voucherDenom, coinSendBackToA.Amount.Unit64(), suite.chainB.SenderAccount.GetAddress(), 0)
packet = channeltypes.NewPacket(fusableTokenPacket.GetBytes(), 1, channelID, channelID, channelID, chainA.PortID, chainA.GetAddress(), chainB.GetAddress(), chainB.GetAddress(), chainB.GetAddress(), chainB.GetAddress())

suite.Require().NoError(err) // relay committed

balance = suite.chainA.App.BankKeeper.GetBalance(suite.chainA.GetAddress(), suite.chainB.SenderAccount.GetAddress(), 0)

suite.Require().Equal(coinSentFromAToB, balance)
```

Token Transfer: model-based test that catches #8120

```plaintext
TestUnescrowTokens ==
  \E s \in DOMAIN history :
  /* IsSource(history[s].packet)
  /* history[s].handler = "OnRecvPacket"
  /* history[s].error = FALSE
```
Token Transfer: TLA+ model

The model is built specifically for testing purposes, both from the spec and from the code, and contains:

- Packets
- Pre- and post-conditions for token transfer relay functions
  - SendTransfer
  - OnRecvPacket
  - OnPacketAcknowledgment
  - OnTimeoutPacket
- Captures the history of arbitrary sequences of calls to these functions
TLA+ model: Packets

```
--- MODULE relay ---

(*
  * A primitive model for account arithmetics and token movement
  * of the Cosmos SDK ICS20 Token Transfer
  * We completely abstract away many details,
  * and want to focus on a minimal spec useful for testing
  *
  * We also try to make the model modular in that it uses denomination traces
  * and accounts via abstract interfaces, outlined in denom.tla and account.tla
*)

EXTENDS Integers, FiniteSets, Sequences, identifiers, denom_record2, account_record

CONSTANT
  MaxAmount

Amounts == 0..MaxAmount

FungibleTokenPacketData == [
  sender: AccountIds,
  receiver: AccountIds,
  denomTrace: DenomTraces,
  amount: Amounts
]

Packets == [
  /* We abstract those packet fields away
  * sequence: uint64
  * timeoutHeight: Height
  * timeoutTimestamp: uint64
  * sourcePort: Identifiers,
  * sourceChannel: Identifiers,
  * destPort: Identifiers,
  * destChannel: Identifiers,
  * data: FungibleTokenPacketData
  */
]```
TLA+ model: Precondition

OnRecvPacketPre(packet) ==

LET data == packet.data
trace == data.denomTrace
denom == GetDenom(trace)
amount == data.amount

IN
/
WellFormedPacket(packet)
/
IsValidRecvChannel(packet)
/
IsValidDenomTrace(trace)
/
amount > 0
/* if there is no receiver account, it is created by the bank */
/
data.receiver /= NullId
/
IsSource(packet) =>

LET escrow == GetDestEscrowAccount(packet) IN
LET denomTrace == ReduceDenomTrace(trace) IN
/
\langle escrow, denomTrace \rangle in DOMAIN bank
/
bank[escrow, denomTrace] >= amount

IsSource(packet) ==
/
GetPort(packet.data.denomTrace) = packet.sourcePort
/
GetChannel(packet.data.denomTrace) = packet.sourceChannel
OnRecvPacketNext(packet) ==
  LET data == packet.data IN
  LET trace == data.denomTrace IN
  LET denom == GetDenom(trace) IN
  LET amount == data.amount IN
  LET receiver == data.receiver IN
  \ IF OnRecvPacketPre(packet)
  \ THEN
  \* This condition is necessary so that denomination traces do not exceed the maximum length
  \* (IsSource(packet) \ TraceLen(trace) < MaxDenomLength)
  \* error' = FALSE
  \* IF IsSource(packet)
  \ THEN
  \* transfer from the escrow account to the receiver account
  LET denomTrace == ReduceDenomTrace(trace) IN
  LET escrow == GetDestEscrowAccount(packet) IN
  LET bankwithreceiver == BankWithAccount(bank, MakeAccount(receiver), denomTrace) IN
  bank' = [bankwithreceiver
    EXCEPT ![MakeAccount(receiver), denomTrace] = @ + amount,
    ![escrow, denomTrace] = @ - amount]
  ELSE
  \* create new tokens with new denomination and transfer it to the receiver account
  LET denomTrace == ExtendDenomTrace(packet.destPort, packet.destChannel, trace) IN
  LET bankwithreceiver ==
    BankWithAccount(bank, MakeAccount(receiver), denomTrace) IN
  bank' = [bankwithreceiver
    EXCEPT ![MakeAccount(receiver), denomTrace] = @ + amount]
  ELSE
  \ error' = TRUE
  \ UNCHANGED bank
TLA+ model: Model execution and history

```plaintext
Next ==
  \p' in Packets
  \count' = \count + 1
  /\ (SendTransferNext(p) /\ handler' = "SendTransfer")
  /\ (OnRecvPacketNext(p) /\ handler' = "OnRecvPacket")
  /\ (OnTimeoutPacketNext(p) /\ handler' = "OnTimeoutPacket")
  /\ (OnAcknowledgementPacketResultNext(p) /\ handler' = "OnRecvAcknowledgementResult")
  /\ (OnAcknowledgementPacketErrorNext(p) /\ handler' = "OnRecvAcknowledgementError")
  /\ history' = [ n in DOMAIN history \union \{count\} |->
    IF n = count' THEN
      [ packet |-> p, handler |-> handler', error |-> error', bankBefore |-> bank, bankAfter |-> bank' ]
    ELSE history[n]
  ]
```
Model-based tests

```cpp
/* Generic test for handler pass */
TestHandlerPass(handlerName) ==
  \E s \in DOMAIN history :
    \// history[s].handler = handlerName
    \// history[s].error = FALSE
    \// history[s].packet.data.amount > 0

/* Generic test for handler fail */
TestHandlerFail(handlerName) ==
  \E s \in DOMAIN history :
    \// history[s].handler = handlerName
    \// history[s].error = TRUE
    \// history[s].packet.data.amount > 0

TestSendTransferPass == TestHandlerPass("SendTransfer")
TestSendTransferFail == TestHandlerFail("SendTransfer")

TestOnRecvPacketPass == TestHandlerPass("OnRecvPacket")
TestOnRecvPacketFail == TestHandlerFail("OnRecvPacket")

TestOnTimeoutPass == TestHandlerPass("OnTimeoutPacket")
TestOnTimeoutFail == TestHandlerFail("OnTimeoutPacket")

TestOnRecvAcknowledgementResultPass == TestHandlerPass("OnRecvAcknowledgementResult")
TestOnRecvAcknowledgementResultFail == TestHandlerFail("OnRecvAcknowledgementResult")

TestOnRecvAcknowledgementErrorPass == TestHandlerPass("OnRecvAcknowledgementError")
TestOnRecvAcknowledgementErrorFail == TestHandlerFail("OnRecvAcknowledgementError")
```
Model-based tests

Test5PacketsAllDifferent ==
\ A s1, s2 \in DOMAIN history :
    s1 /= s2 => history[s1].handler /= history[s2].handler

Test5PacketsAllDifferentPass ==
\ \ A s \in DOMAIN history :
    s > 0 =>
        \ history[s].error = FALSE
        \ history[s].packet.data.amount > 0

TestUnescrowTokens ==
\ E s \in DOMAIN history :
    \ IsSource(history[s].packet)
    \ history[s].handler = "OnRecvPacket"
    \ history[s].error = FALSE
Logic, or how a TLA+ assertion generates a multi-chain execution

Step 0: Initialization; empty bank

Bank of chain B:

<table>
<thead>
<tr>
<th>Account</th>
<th>Denomination</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Logic, or how a TLA+ assertion generates a multi-chain execution

**Step 1:** Receive 5 atoms from chain A to account a3

Transferred denomination trace: **atom**

<table>
<thead>
<tr>
<th>Account</th>
<th>Denomination</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>a3</td>
<td>transfer/channel-0/atom</td>
<td>5</td>
</tr>
</tbody>
</table>
Logic, or how a TLA+ assertion generates a multi-chain execution

**Step 2:** Send 3 atoms from account a3 on chain B to chain C

Transferred denomination trace: `transfer/channel-0/atom`

<table>
<thead>
<tr>
<th>Account</th>
<th>Denomination</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>a3</td>
<td><code>transfer/channel-0/atom</code></td>
<td>2</td>
</tr>
<tr>
<td><code>transfer/channel-1</code></td>
<td><code>transfer/channel-0/atom</code></td>
<td>3</td>
</tr>
</tbody>
</table>
Logic, or how a TLA+ assertion generates a multi-chain execution

**Step 3:** Receive 1 atom from chain C to account a1 on chain B

Transferred denomination trace: `transfer/channel-0`\slash`transfer/channel-0`\slash`atom`

<table>
<thead>
<tr>
<th>Bank of chain B:</th>
<th>Account</th>
<th>Denomination</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a3</td>
<td><code>transfer/channel-0</code>\slash<code>atom</code></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td><code>transfer/channel-1</code></td>
<td><code>transfer/channel-0</code>\slash<code>atom</code></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>a1</td>
<td><code>transfer/channel-0</code>\slash<code>atom</code></td>
<td>1</td>
</tr>
</tbody>
</table>
Model-based test vs standard test
Apalache model checker

Apalache
The Symbolic Model Checker for TLA+

View the Project on GitHub
informatics/apsalache

Download ZIP File  Download TAR Ball  View On GitHub

Tweets by @ApalacheTLA

Apalache: symbolic model checker
@ApalacheTLA

After a short break in our weekly release cycle the last week, we have enough features and bugfixes for 0.10.1. This release improves the type checker and the parser:
github.com/informatics.../Apalache

Release v0.10.1 - informatics.com 0.10.1 Features Support for...github.com

Feb 22, 2021

Apalache: symbolic model checker
@ApalacheTLA

It's Apalache 0.10.0 this week 🎉. It includes bugfixes 🛠 and new features: Java-like annotations in comments for the new type checker, polymorphic signatures in the new type checker, sorting declarations in topological order, parallel assignments.

More: github.com/informatics...

Features • Installation • Manual • Releases • Chat • Contribute

Apalache translates TLA+ into the logic supported by SMT solvers such as Microsoft Z3. Apalache can check inductive invariants (for fixed or bounded parameters) and check safety of bounded executions (bounded model checking). To see the list of supported TLA+ constructs, check the supported features. In general, Apalache runs under the same assumptions as TLC.

To learn more about TLA+, visit Leslie Lamport's page on TLA+ and see his video course.

Talks and tutorials

- How TLA+ and Apalache Helped Us to Design the Tendermint Light Client. Interchain Conversations 2020 (December 2020).
- Model-based testing with TLA+ and Apalache. TLA+ Community Event 2020 (October 2020).
- Type inference for TLA+ in Apalache. TLA+ Community Event 2020 (October 2020).
- Formal Spec and Model Checking of the Tendermint Blockchain Synchronization Protocol 2nd Workshop on Formal Methods for Blockchains (July 2020).
- TLA+ model checking made symbolic OOPSLA 2019 (October 2019).
- Bounded model checking of TLA+ specifications with SMT TLA+ Community Event 2018 (July 2018).
Current instantiations of MBT for Cosmos infrastructure
Current instantiations of MBT for Cosmos infrastructure

Model
- TLA+

Tests
- TLA+

Research engineer
- Manual & stored

Developer
- Manual & stored

Model checker
- Apalache / TLC

Counterexample
- TLA+ / JSON

Translator
- Manual & stored

Unit tests
- Manual & stored

Abstraction
- Testgen

Tendermint-rs
- Light Client
- Manual test driver

Tendermint-go
- Light Client
- Manual test driver

IBC-rs
- ICS02 Client
- ICS03 Connection

Cosmos-SDK
- Token Transfer
- Test driver

Transformation spec
- Manual & stored

Fuzzing

Conformance tests
- Manual & stored
New MBT infrastructure

- Model: TLA+
- Tests: TLA+
- Unit tests PB-tests
- Abstraction
- MBT-Core
  - Automatic:
    - model instantiation
    - history tracking
    - model & test combination
  - Model checker: Apache / TLC
  - Counterexample: TLA+ / JSON
  - Translator
  - Tests
  - MBT-Rust
    - communication with MBT-core
    - test execution
    - fuzzing via PB-libraries
    - presentation of results
  - MBT-Go
  - Tendermint-rs
    - Light Client
  - Tendermint-go
    - Light Client
  - Developer
    - Automated
  - Research engineer
    - Automated
Conclusion

- Model-based testing is an adequate tool for complex & critical systems, such as blockchain communication protocols
- MBT significantly improves
  - ease of writing / using the tests
  - tests maintainability
  - code coverage
- MBT allows to keep specifications and code synchronized
- The benefits substantially outweigh infrastructure investments
Current & Future work

- Improving the MBT infrastructure: **make it fun for developers!**
  - generation & execution of massive test suites (1000s of tests)
  - automated partial generation of models, abstractions, and test drivers
  - natural interoperability with developer tools: continuous integration, unit- and property-based tests

- Executable TLA+
  - code generation from TLA+; will allow to substantially speed up testing

- Distributed testing
  - cut the system at the interface points
  - replace some modules with executable specs

- Real-time monitoring
  - Execute the real system, and check execution traces for spec conformance