Model Checking Blockchain-based System: A Case Study

Lung-Chen Huang, Naipeng Dong,
Guangdong Bai, Siau Cheng Khoo, Jin Song Dong
Outline

• Model Checking and PAT Introduction
• Blockchain-based Booking System and its Modeling
• Demo
Model checking

• Model checking: check whether a model satisfies a property by **exhaustive** searching.
Tool and modeling language used in this project

• Process Analysis Toolkit (PAT)

• Modeling language: CSP#, which is an extension of classic process algebra
Communicating Sequential Processes (CSP)
  • Constants
    • `#define N 5;`
  • Variables of Type Bool, Integer, Arrays of integers
    • `var x: {0..10} = 5;`
    • `var x[N];`
  • Channels
    • `channel m 0;`
    • `channel m[M] channelBufferSize;`
Tool and modeling language used in this project (cont.)

• Modeling language: CSP# (cont.)

<table>
<thead>
<tr>
<th>Process Expression</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>Do nothing</td>
</tr>
<tr>
<td>Skip</td>
<td>Termination, like Return</td>
</tr>
<tr>
<td>e{x:=1} -&gt; P</td>
<td>Event prefixing</td>
</tr>
<tr>
<td>if(b) {P} else {Q}</td>
<td>Choice</td>
</tr>
<tr>
<td>P; Q</td>
<td>Sequential Composition</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>
Blockchain-based hotel booking system

• Time and again, **travelers** experience difficulty in selecting hotels
  • Each time when a search result appears on screen, they still have hard time browsing through the choices and finding the best suited hotel room

• **Hoteliers** have extra cost to get known by travelers

• How blockchain can be beneficial and applicable?
  • A transparent ecosystem
  • The booking requests are similar (easy to contract)
Make a request

1. Client
   - Create a Request in DSL
2. Client App
   - Send Request
   - Compile Request into Smart Contract
3. Blockchain
   - Deploy Smart Contract
4. DSL Owner (SC)
   - Request (SC)
   - Add Request Address
   - Emit Success Event
Offer a proposal
Accept/Reject proposals
Lease the room
Modeling – user issues a request

Traveler: any address that is willing to book a hotel room with the deployed request

User(addr, val, gas, gasPrice, function)

ID

Gas required for executing this contract

ProposeFunction, SwitchFunction, FetchFunction, SettleFunction...

= TxBrdcst(θ, addr, val, gas, gasPrice, function);
Modeling – transactions are broadcast to miners

\[ \text{TxBrdcst}(\text{iter}, \text{addr}, \text{val}, \text{gas}, \text{gasPrice}, \text{function}) = \text{// broadcasting a transaction} \]
\[
\text{atomic}\{
\text{if } (\text{iter} < M) \{ \\
\text{mnet}[\text{iter}].!\text{addr}.\text{val}.\text{gas}.\text{gasPrice}.\text{function} \rightarrow \\
\text{TxBrdcst}(\text{iter} + 1, \text{addr}, \text{val}, \text{gas}, \text{gasPrice}, \text{function}) \\
\} \text{ else } \{
\text{Skip}
\}\}
\]

\[ \text{TxPool}(i) = \text{// inserting new transactions into the pool} \]
\[
\text{mnet}[i].!\text{addr}.\text{val}.\text{gas}.\text{gasPrice}.\text{function} \rightarrow \\
\text{if } (\text{userWallet}[i][\text{addr}] \geq (\text{val} + (\text{gas} \times \text{gasPrice})) \&\& \text{gas} > 0) \{ \text{// a checkpoint on whether or not the sender has enough funds and gas before a miner accepts the transaction} \\
\text{TxInsert}(i, \text{addr}, \text{val}, \text{gas}, \text{gasPrice}, \text{function}) \\
\} \text{ else } \{
\text{TxPool}(i)
\}\]

Number of miners

Omitted, which insert the transaction into the miner-owned pools
Modeling – block construction

\[
\text{TxBlock}(i) = \begin{cases} 
// including available transactions from the pool 
\text{if} \ (\text{inPoolPtr}[i] < \text{poolSize}) \{ 
\quad \text{if} \ (!\text{call}(\text{cempty}, \text{datach}[i])) \{ 
\quad \quad \text{datach}[i] = \text{newcomer} -> 
\quad \quad \text{if} \ (\text{poolTxGas}[i][\text{inPoolPtr}[i]] == 0) \{ 
\quad \quad \quad \text{if} \ (\text{blkSize}[i] \geq 1) \{ 
\quad \quad \quad \quad \text{Miner}(i, 0) 
\quad \quad \quad \} \text{else} \{ 
\quad \quad \quad \quad \text{TxBlock}(i) \} 
\quad \quad \} \text{else} \{ 
\quad \quad \text{txInclude}\{ 
\quad \quad \quad \text{blkTxAddr}[i][\text{bIter}] = \text{poolTxAddr}[i][\text{pIter}]; 
\quad \quad \quad \text{blkTxVal}[i][\text{bIter}] = \text{poolTxVal}[i][\text{pIter}]; 
\quad \quad \quad \text{blkTxGas}[i][\text{bIter}] = \text{poolTxGas}[i][\text{pIter}]; 
\quad \quad \quad \text{blkTxGasPrice}[i][\text{bIter}] = \text{poolTxGasPrice}[i][\text{pIter}]; 
\quad \quad \quad \} \text{inPoolPtr}[i]++;
\quad \quad \quad \} \text{blkSize}[i]++;
\quad \quad \} \text{else} \{ 
\quad \quad \text{txInclude}\{ 
\quad \quad \quad \text{blkTxAddr}[i][\text{bIter}] = \text{poolTxAddr}[i][\text{pIter}]; 
\quad \quad \quad \text{blkTxVal}[i][\text{bIter}] = \text{poolTxVal}[i][\text{pIter}]; 
\quad \quad \quad \text{blkTxGas}[i][\text{bIter}] = \text{poolTxGas}[i][\text{pIter}]; 
\quad \quad \quad \text{blkTxGasPrice}[i][\text{bIter}] = \text{poolTxGasPrice}[i][\text{pIter}]; 
\quad \quad \quad \} \text{inPoolPtr}[i]++;
\quad \quad \quad \} \text{blkSize}[i]++;
\quad \quad \} 
\end{cases}
\]
Modeling – mining

Miner(i, iter) = // start mining the block where transactions were collected
if (iter < blkSize[i]) {
    LockUp(i, iter)
} else {
    BlkUpdate(i, 0)
}

LockUp(i, iter) = // to lock up the total value including the gas to execute the transaction from a user
weiLockUp{
    var addr = blkTxAddr[i][iter];
    var val = blkTxVal[i][iter];
    var gas = blkTxGas[i][iter];
    var gasPrice = blkTxGasPrice[i][iter];
    var lockedTotal = val + (gas * gasPrice);
    userWallet[i][addr] = userWallet[i][addr] - lockedTotal;
    lockedWallet[i] = lockedTotal;
} ->
TxExec(i, iter);
Modeling – contract execution

\texttt{TxExec(i, \text{iter}) = // acting like invoking predefined functions in the}
\texttt{smart contract; here just dispatch the execution for each of the functions}
\texttt{case \{}
\texttt{blkTxFunction[i][\text{iter}] == SwitchFunction:}
\texttt{\hspace{10pt} ... ...}
\texttt{blkTxFunction[i][\text{iter}] == ProposeFunction:}
\texttt{\hspace{10pt} if (contractSwitch[i] == on) \{}
\texttt{\hspace{20pt} GasConsume(i, \text{iter, UPDATE)} || Execution(i, \text{iter})}
\texttt{\hspace{10pt} \} else \{ LockedReturn(i, \text{iter, false}) \}}
\texttt{blkTxFunction[i][\text{iter}] == SettleFunction:}
\texttt{\hspace{10pt} ... ...}
\texttt{blkTxFunction[i][\text{iter}] == FetchFunction:}
\texttt{\hspace{10pt} ... ...}
\texttt{\}};

\texttt{BlkDetect(i, \text{iter, success}) = // before executing the next transaction, check whether there is any}
\texttt{block from another miner. Under assumption that the miner chooses the first arrival block}
\texttt{if (call(cempty, bnet[i])) \{ Miner(i, \text{iter}+1)}
\texttt{\} else \{ bnet[i]?j.newBlockNum.blockid -> BlkAppend(i, j, newBlockNum, blockid) \}};
Modeling – commit a mined block

Miner(i, iter) = // start mining the block where transactions were collected
if (iter < blkSize[i]) {
    LockUp(i, iter)
} else {
    BlkUpdate(i, 0)
}

BlkUpdate(i, iter) = // update the globally shared state
if (iter < blkSize[i]) {
    update{
        block[blockUid][iter] = pendUid[i][iter];
        txFromAddr[pendUid[i][iter]] = pendFromAddr[i][iter];
        txToAddr[pendUid[i][iter]] = pendToAddr[i][iter];
        txOp[pendUid[i][iter]] = pendOp[i][iter];
        ... ...
    } ->
    BlkUpdate(i, iter+1)
} else {
    reward{minerCoinbase[i] = minerCoinbase[i] + succAppendPrice; rewardCount++} ->
    BlkBrdcst(i, 0, blockNum[i], blockUid)
}
Modeling – append to blockchain

```
BlkAppend(i, j, newBlockNum, blockid) = // append the latest block and update the state in the blockchain
  append{
    chain[i][newBlockNum] = blockid;
    blockNum[i]++;
    blkSize[i] = blkSize[j];
  } ->
  ChainUpdate(i, 0, blockid);

ChainUpdate(i, iter, blockid) = // update their own chain
  case {
    txOp[block[blockid][iter]] == FetchFunction:
      ...
    txOp[block[blockid][iter]] == SwitchFunction:
      ...
    txOp[block[blockid][iter]] == ProposeFunction:
      propose{ ... } -> ChainUpdate(i, iter+1, blockid)
    txOp[block[blockid][iter]] == SettleFunction:
      settle{ ... } -> ChainUpdate(i, iter+1, blockid)
  }
```
# Experiment results

- C = 5, M = 2 (5 users, 2 miners)

<table>
<thead>
<tr>
<th>Assertion</th>
<th>States</th>
<th>Transitions</th>
<th>Time</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProposerExecution deadlockfree</td>
<td>8328</td>
<td>18823</td>
<td>1.76</td>
<td>Valid</td>
</tr>
<tr>
<td>ProposerExecution</td>
<td>= []! GasRunOut</td>
<td>13413</td>
<td>30563</td>
<td>2.68</td>
</tr>
<tr>
<td>ProposerExecution reaches sameBlockNumEventually</td>
<td>285</td>
<td>336</td>
<td>0.04</td>
<td>Valid</td>
</tr>
<tr>
<td>ListenerExecution</td>
<td>= proposalReceived → listenerReceiving</td>
<td>13413</td>
<td>30563</td>
<td>2.76</td>
</tr>
<tr>
<td>UnavailableExecution</td>
<td>= []! proposalReceived</td>
<td>13413</td>
<td>30563</td>
<td>2.73</td>
</tr>
<tr>
<td>NotOwnerExecution</td>
<td>= []! fetch</td>
<td>4537</td>
<td>11940</td>
<td>0.78</td>
</tr>
<tr>
<td>MultipleUsersExecution reaches sameBlockNumEventually</td>
<td>504</td>
<td>521</td>
<td>0.05</td>
<td>Valid</td>
</tr>
<tr>
<td>SettlementExecution reaches receiveSettlement</td>
<td>362</td>
<td>605</td>
<td>0.06</td>
<td>Valid</td>
</tr>
</tbody>
</table>
# Experiment results

- \( C = 5, M = 10 \) (5 users, 10 miners)

<table>
<thead>
<tr>
<th>Assertion</th>
<th>States</th>
<th>Transitions</th>
<th>Time</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProposerExecution deadlockfree</td>
<td>6774622</td>
<td>68280939</td>
<td>3647.43</td>
<td>Incomplete</td>
</tr>
<tr>
<td>ProposerExecution ( \models []! \text{GasRunOut} )</td>
<td>169626</td>
<td>2165266</td>
<td>112.52</td>
<td>Incomplete</td>
</tr>
<tr>
<td>ProposerExecution reaches sameBlockNumEventually</td>
<td>297717</td>
<td>415836</td>
<td>21.45</td>
<td>Valid</td>
</tr>
<tr>
<td>ListenerExecution ( \models \text{proposalReceived} \rightarrow \text{listenerReceiving} )</td>
<td>230685</td>
<td>2897974</td>
<td>415.428</td>
<td>Incomplete</td>
</tr>
<tr>
<td>UnavailableExecution ( \models []! \text{proposalReceived} )</td>
<td>169929</td>
<td>2168902</td>
<td>111.65</td>
<td>Incomplete</td>
</tr>
<tr>
<td>NotOwnerExecution ( \models []! \text{fetch} )</td>
<td>317081</td>
<td>1511802</td>
<td>189.67</td>
<td>Incomplete</td>
</tr>
<tr>
<td>MultipleUsersExecution reaches sameBlockNumEventually</td>
<td>10312</td>
<td>10425</td>
<td>0.98</td>
<td>Valid</td>
</tr>
<tr>
<td>SettlementExecution reaches receiveSettlement</td>
<td>3313604</td>
<td>32332261</td>
<td>1607.98</td>
<td>Incomplete</td>
</tr>
</tbody>
</table>
Thank You!