Pirouette - Model-checking Plutus Smart Contracts

Guillaume Genestier

joint work with Victor Micaldi, Georg Rudoy and Florent Chevrou

Tuesday, November 23 2021
A Few Words about Transactions on the Blockchain

- UTxO Model
- Validator scripts

Example of Contracts

Model-checker

Demo

Translating Plutus to TLA+
Unspent Transaction Outputs (UTxO)

50 ADAs

30 ADAs

15 ADAs

30 ADAs

40 ADAs
Unspent Transaction Outputs (UTxO)

Bob pays 40 ADAs to Charlie

50 ADAs
30 ADAs
15 ADAs
30 ADAs
40 ADAs

20 ADAs
40 ADAs
Unspent Transaction Outputs (UTxO)

Bob pays 40 to Charlie

50 ADAs

30 ADAs

15 ADAs

30 ADAs

40 ADAs

20 ADAs

40 ADAs
Contract Outputs

30 ADAs → Bob pays 40 to Charlie → 20 ADAs

30 ADAs → 40 ADAs

Some Data
30 ADAs

Another input is present
Expected output data
Fee must be paid to contract owner
Contract Outputs

Bob pays 40 ADAs to Charlie

Some Data
30 ADAs

Other Data

Another input is present
Expected output data
Fee must be paid to contract owner

Another Transaction

...
1. A Few Words about Transactions on the Blockchain

2. Example of Contracts
   - Multi-Signature Account

3. Model-checker

4. Demo

5. Translating Plutus to TLA+
Multi-Signature Account – Gathering Funds

Holding \{ \text{required} = 2; \text{signers} = [\text{Alice}, \text{Bob}, \text{Charlie}] \}

20 ADAs -> Contract Start -> 60 ADAs

20 ADAs

20 ADAs

20 ADAs
Multi-Signature Account – Proposing Payment

Holding \{ \text{required} = 2; \}
\text{signers} = [\text{ملك, مشتركة, شركة}]

60 ADAs \rightarrow \text{ProposePay}

Transaction signed by allowed signer
Amount smaller than deposit
Deadline is early enough
Parameters did not change

Collecting \{ \text{required} = 2; \}
\text{allowed} = [\text{ملك, مشتركة, شركة}]
\text{collected} = []
\text{amount} = 30 ADAs
\text{receiver} = 
\text{deadline} = 31/11/2021

60 ADAs
Collecting \{ \text{required} = 2; \\
\text{allowed} = [\hat{A}, \hat{B}, \hat{C}] \\
\text{collected} = [] \\
\text{amount} = 30 \text{ ADAs} \\
\text{receiver} = \hat{B} \\
\text{deadline} = 31/11/2021 \} \\
60 \text{ ADAs}

Transaction signed by allowed signer
Did not already signed
Collected changed accordingly
Other fields did not change
Multi-Signature Account – Paying

Collecting \( \{ \text{required} = 2; \) \\
\text{allowed} = \{\text{A}, \text{A}, \text{A}\} \) \\
\text{collected} = \{\text{A}, \text{A}\} \\
\text{amount} = 30 \text{ ADAs} \\
\text{receiver} = \text{A} \\
\text{deadline} = 31/11/2021\} \\
\text{60 ADAs} \rightarrow \text{Pay} \\
\begin{aligned}
\text{Deadline not reached} \\
\text{Enough collected signatures} \\
\text{Transaction is the declared one} \\
\text{Parameters did not change}
\end{aligned}

Send Payment

Holding \( \{ \text{required} = 2; \) \\
\text{allowed} = \{\text{A}, \text{A}, \text{A}\}\} \\
\begin{aligned}
\text{30 ADAs} \\
\text{30 ADAs}
\end{aligned}
A Few Words about Transactions on the Blockchain

Example of Contracts

Model-checker
- Temporal Logic of Actions

Demo

Translating Plutus to TLA+
Why Model-Checking?

Fully machine-checked proof of correctness:
+ Absolute certainty of correctness wrt. assumptions;
  – Difficult to develop and only as good as the spec;
  – Tools are harder to learn.

Model-checking:
+ More accessible;
+ Enables user to refine and correct their spec more easily;
  – Not a proof of correctness: the checker only analyzes a finite subset of states;
  – Slow to check large models.
Temporal Logic of Actions

Two modalities:
- □ Always
- ◊ Eventually

One relation: ⇝ Then

Next step variables: \( y' = x' + y \)
- Next value of \( y \) is the next value of \( x \) plus the current value of \( y \).

Action: \([A]_{\{x,y\}}\)
- Either \( A \) holds or
  \[ x' = x \land y' = y \]
1 A Few Words about Transactions on the Blockchain
2 Example of Contracts
3 Model-checker
4 Demo
5 Translating Plutus to TLA+
--- proposals over a period of time, using separate transactions. All contract
--- state is kept on the chain so there is no need for off-chain communication.

--- A proposal for making a payment under the multisig scheme.
data Payment = Payment
    { paymentAmount :: Value
    -- How much to pay out.
    , paymentRecipient :: PubKeyHash
    -- Address to pay the value to
    , paymentDeadline :: Slot
    -- Time until the required amount of signatures has to be collected.
    }
deriving stock (Haskell.Show, Generic, Haskell.Eq)
deriving anyclass (ToJSON, FromJSON)

instance Eq Payment where
    (==) Payment vl pk sl = vl' == vl && pk' == pk' && sl' == sl'

instance Eq Params where
    (==) Params (Params vl' pk' sl') = vl' == vl && pk' == pk && sl' == sl

instance Eq MSState where
    (==) MSState Holding
        -- Money is locked, anyone can make a proposal for a payment. If there is
        -- no value here then this is a final state and the machine will terminate.
    (==) MSState [CollectingSignatures Payment [PubKeyHash]]
        -- A payment has been proposed and is awaiting signatures.
        deriving stock (Haskell.Show, Generic, Haskell.Eq)
deriving anyclass (ToJSON, FromJSON)
(termbind
  (strict)
  (vardecl
   addInteger_30853
   (fun (con integer) (fun (con integer) (con integer)))
  )
  (lam
   x_30851
   (con integer)
   (lam
    y_30852
    (con integer)
    [[ (builtin addInteger) x_30851 ] y_30852 ]
  )
)
)
cabal run pirouette -- tests/integration/MultiSigStateMachine/MultiSigStateMachine.flat \
    --prefix transition \ 
    --with-args param,st,INPUT \ 
    --tla-skel tests/integration/MultiSigStateMachine/Skeleton.tla \ 
    --action-wrapper "LET res == ___ IN st' = res.arg1 ~/\ txConstr' = res.arg0"
--- MODULE Contract ---

EXTENDS Integers, TLC, Sequences

VAR st, txConstr

CONSTANT MAXDEPTH, PlutusByteString, ParamSigners, ParamN

vars =>
  <-<st, txConstr>>>

PlutusInteger => 0..6

PlutusData => {}

fst(tup) => tup[1]

snd(tup) => tup[2]

Init =>
  \ txConstr = TxConstraints1(Nil, Nil, Nil)
  \ st = State1(Holding, Nil)

Spec =>
  Init \ \ (Next) vars \ \ WP_vars(Next)

RECURSIVE len(_
  len(xs) =>
    Nil_match(xs, 0, LAMBDA i22, i31 : 1 + len(i31))

SillyInv => [i => ((st.arg0.cons = "CollectingSignatures" \ len(st.arg0.arg1) = 1))]

===
SetOfList(a) ==
    UNION {{1..n -> a: n \in 0..MAXDEPTH-1}}

Nil_match(l,cN,cC(_,_)) ==
    IF l = <>>
      THEN cN
    ELSE LET hd ==
        Head(l)
      tl ==
        Tail(l)
      IN cC(hd,tl)

Nil ==
    <>>

Cons(x,tl) ==
    <x> \ o tl

RECURSIVE foldr0(_,_,_)
foldr0(f30077(_,_),acc30078,l30079) ==
    Nil_match(l30079,
               acc30078,
               LAMBDA x30082,xs30083 : f30077(x30082,
                                          foldr0(f30077,acc30078,xs30083)))
ActionAddSignature(x0) ==
  /
  \ st.cons = "State1"
  /
  LET ds31007 ==
    st.arg0
  ds31008 ==
    st.arg1
  IN \ ds31007.cons = "CollectingSignatures"
  /
  LET pmt31014 ==
    ds31007.arg0
  pks31015 ==
    ds31007.arg1
  IN \ Params_match(param,
    LAMBDA sigs30885,ds30886 : foldr0(LAMBDA aSpz3,
    bSpz4 : Bool_match
      (x0 = aSpz3,
        True,
        bSpz4),
    False,
    sigs30885))
  /
  LET res ==
    Tuple21(TxConstraints1(Cons(MustBeSignedBy(x0),Nil),
      Nil,
      Nil),
    State1(CollectingSignatures(pmt31014,
      Cons(x0,
      pks31015)),
    ds31008))
  IN st' = res[2] \ txConstr' = res[1]

WrappedAddSignature ==
  \ E x0 \in PlutusByteString: ActionAddSignature(x0)
Finding Faults

```haskell
{-# INLINABLE valuePreserved #-}

proposalAccepted :: Params -> [PubKeyHash] -> Bool
proposalAccepted (Params signatories numReq) pks =
  let numSigned = length (filter (\pk -> containsPk pk pk signatories) pks)
  in numSigned >= numReq

{-# INLINEIR #}

transition params State{ stateData =s, stateValue=currentValue} i = case (s, i)
  |
  (CollectingSignatures pmt pks, AddSignature pk)
-|  isSignatory pk params && not (containsPk pk pk signatories) pks ->
+|  isSignatory pk params ->
    let constraints = Constraints.mustBeSignedBy pk in
    Just ( constraints
      , State
```
```tla
vars == << signs', st >>

ProposePayment == st = st_Holding /
    \ st' = st_Collecting /
    \ signs' = () /
    \ paid' = FALSE

Pay == st = st_Collecting /
    \ Cardinality(signs) >= N /
    \ st' = st_Holding /
    \ signs' = () /
    \ paid' = TRUE

Cancel == st = st_Collecting /
    \ st' = st_Holding /
    \ signs' = () /
    \ UNCHANGED paid

AddSig(s) == st = st_Collecting /
    \ st' = st_Collecting /
    \ signs' = ( s ) \ Union signs /
    \ UNCHANGED paid

Next == ProposePayment /
    /\ Pay /
    /\ Cancel /
    /\ ( \forall s \in Signers : AddSig(s))

Init == signs = () /
    \ st = st_Holding /
    \ paid = FALSE

Spec == Init \ [ [Next] vars \ WF_vars(Next) 
```
Pirouette - Model-checking Plutus Smart Contracts

Guillaume Genestier

---

**Status**
Checking Contract_MC.tla / Contract_MC.cfg
Errors: 1 error(s)

**Coverage**

<table>
<thead>
<tr>
<th>States</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Module</td>
</tr>
<tr>
<td>Contract</td>
<td>0</td>
</tr>
<tr>
<td>Contract</td>
<td>Action</td>
</tr>
<tr>
<td>Contract</td>
<td>Action</td>
</tr>
<tr>
<td>Contract</td>
<td>Action</td>
</tr>
</tbody>
</table>

**Errors**

Invariant SilyInv is violated.

**Error Trace**

1: Initial predicate

- 1: (3) \[ arg0 \rightarrow \{ \text{Holding} \}, \text{arg1} \rightarrow \{ \}, \text{cons} \rightarrow \{ \text{State1} \} \]
- txConstr (4) \[ arg0 \rightarrow \{ \}, arg1 \rightarrow \{ \}, arg2 \rightarrow \{ \}, \text{cons} \rightarrow \{ \text{TxConstraints} \} \]

2: ActionProposePayment in Contract

- 1: (3) \[ M \] \[ arg0 \rightarrow \{ \}, \text{arg1} \rightarrow \{ \}, \text{arg2} \rightarrow \{ \}, \text{cons} \rightarrow \{ \text{Payment} \}, \text{arg} \rightarrow \{ \} \]
- txConstr (4) \[ arg0 \rightarrow \{ \}, \text{arg1} \rightarrow \{ \}, \text{arg2} \rightarrow \{ \}, \text{cons} \rightarrow \{ \text{TxConstraints} \} \]

3: ActionAddSignature in Contract

- 2: (4) \[ M \] \[ arg0 \rightarrow \{ \}, \text{arg1} \rightarrow \{ \}, \text{arg2} \rightarrow \{ \} \]
- txConstr (4) \[ arg0 \rightarrow \{ \}, \text{arg1} \rightarrow \{ \}, \text{cons} \rightarrow \{ \text{MustBeSignedBy} \}, \text{arg} \rightarrow \{ \}, \text{arg2} \rightarrow \{ \} \]
Checking Contract_MC.tla / Contract_MC.cfg

Errors: 1 error(s)

States
<table>
<thead>
<tr>
<th>Time</th>
<th>Diameter</th>
<th>Found</th>
<th>Distinct</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00:00</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>00:00:01</td>
<td>4</td>
<td>612</td>
<td>612</td>
<td>602</td>
</tr>
</tbody>
</table>

Coverage

<table>
<thead>
<tr>
<th>Module</th>
<th>Action</th>
<th>Total</th>
<th>Distinct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract</td>
<td>ActionAddSignature</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>Contract</td>
<td>ActionCancel</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Contract</td>
<td>ActionPay</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Contract</td>
<td>ActionProposerPayment</td>
<td>589</td>
<td>588</td>
</tr>
</tbody>
</table>

Errors

Invariant PayCorrect is violated.

Error Trace

1: Initial predicate
   ▶ VX (11) (arg0) -> [arg0] -> "Holding", arg1 -> arg2 -> cons -> "State1"
   ▶ InConstr (4) (arg0) -> arg1 -> arg2 -> cons -> "TxConstraints1"

2: ActionProposerPayment in Contract
   ▶ VX (11) (arg0) -> arg1 -> arg2 -> cons -> "Payment1", arg...
   ▶ InConstr (4) (arg0) -> arg1 -> arg2 -> cons -> "TxConstraints1"

3: ActionAddSignature in Contract
   ▶ VX (11) (arg0) -> arg1 -> arg2 -> cons -> "Payment1", arg...
   ▶ InConstr (4) (arg0) -> arg1 -> arg2 -> cons -> "MustBeSignedBy"...

4: ActionAddSignature in Contract
   ▶ VX (11) (arg0) -> arg1 -> arg2 -> cons -> "Payment1", arg...
   ▶ InConstr (4) (arg0) -> arg1 -> arg2 -> cons -> "MustBeSignedBy"...
A Few Words about Transactions on the Blockchain

Example of Contracts

Model-checker

Demo

Translating Plutus to TLA+
- State Machine Library
- Representing Datas
We developed *Pirouette*, an automatic generator of *TLA*+ programs from *Plutus* smart contract.

- From *Haskell* to *PlutusIR*;
  Already existed, provided by Plutus developers.

- From *PlutusIR* to our *System F* encoding;
  Quite straightforward. Put every inner lets to top-level. Do some inlining.

- From *System F* to *TLA*+.
Identifying action in a program is not straightforward.

\( TLA+ \) is designed to represent state machine.

The State Machine library existed and came with examples.
Haskell programs declares a lot of inductive data types.

$\textit{TLA+}$ relies mainly on a ”set-theoretic” view of the object. Fortunately, it features ”heterogeneous finite domain function”.

Just($x_0$) ==

[cons |-> "Just", arg0 |-> $x_0$]

Nothing ==

[cons |-> "Nothing"]

Maybe_match($x$, cJust(_), cNothing) ==

CASE $x$.cons = "Just" -> cJust($x$.arg0) []

$x$.cons = "Nothing" -> cNothing

SetOfMaybe($a$) ==

UNION ({{[cons: {"Just"}, arg0: $a$], [cons: {"Nothing"}]}})
Two very different kind of functions:

- \([x : 1..8 \mapsto x + 3]\)
  - Set-theorist definition of a function, so it is the set
    \(\{(1, 4), (2, 5), (3, 6), (4, 7), (5, 8), (6, 9), (7, 10), (8, 11)\}\).
  - No constraints about arity or recursivity.
  - The whole set is computed at the declaration, and the arguments are always checked to be in the domain.

- \(f(x) == x + 3\)
  - An operator, can mainly be thought of as a macro. Almost always inlined.
  - Its arity and the arity of its arguments are fixed at declaration time.
  - Restriction on the possibility to have recursive ones (No mutual recursion on higher-order operators).
Pirouette - Model-checking Plutus Smart Contracts

Guillaume Genestier

Contract_MC.cfg - pirouette - Visual Studio Code

States

Coverage

Errors

Action property line 44, col 17 to line 44, col 29 of module ContractSpec is violated.

Error Trace

1: Initial predicate
   ST (3) {arg0} {cons | Holding, arg1 | <===, cons | State1}
   txConstr (4) {arg0 | <===, arg1 | <===, arg2 | <===, cons | TxConstraints}

2: ActionProposePayment in Contract
   ST (13) M {arg0 | <===, arg1 | <===, arg2 | <===, arg1 | s1, arg2 | >= 0, cons | Payment1, arg1 | <===, arg2 | <===, cons | TxConstraints}

3: ActionAddSignature in Contract
   ST (3) M {arg0 | <===, arg1 | <===, arg1 | s1, arg2 | >= 0, cons | Payment1, arg1 | <===, arg2 | <===, cons | TxConstraints}
   txConstr (4) M {arg0 | <===, arg1 | <===, arg2 | <===, cons | MustBeSignedBy, arg1 | <===, arg2 | <===}

4: ActionAddSignature in Contract
   ST (3) M {arg0 | <===, arg1 | <===, arg1 | s1, arg2 | >= 0, cons | Payment1, arg1 | <===, arg2 | <===, cons | TxConstraints}
   arg0 (4) M {arg0 | <===, arg1 | s1, arg2 | >= 0, cons | Payment1, arg1 | <===, arg2 | <===, cons | TxConstraints}
   arg1 (4) M {arg0 | <===, arg1 | s1, arg2 | >= 0, cons | Payment1, arg1 | <===, arg2 | <===, cons | TxConstraints}
   arg2 (2) M {arg0 | <===, arg1 | s2, arg2 | >= 0, cons | MustBeSignedBy, arg1 | <===, arg2 | <===, cons | TxConstraints}
   tConstr (4) M {arg0 | <===, arg1 | s2, cons | MustBeSignedBy, arg1 | <===, arg2 | <===, cons | TxConstraints}

5: ActionPay in Contract
   ST (3) M {arg0 | <===, arg1 | <===, cons | Holding, arg1 | <===, cons | State1}
   txConstr (4) M {arg0 | <===, arg1 | <===, arg2 | <===, cons | Negligible, arg1 | TRUE, cons | TxConstraints}
Pirouette is open-source.
Give it a try: https://github.com/tweag/pirouette

Still a lot to do:
- Use validator scrit directly;
- Handle time constraints;
- Monomorphisation;
- Generate skeletons.
Pirouette - Model-checking Plutus Smart Contracts

Guillaume Genestier

joint work with Victor Micali, Georg Rudoy and Florent Chevrou

Tuesday, November 23 2021

TWEAG