MODULE *ForceMove*

EXTENDS $Integers$, TLC, $Utils$

CONSTANTS

$StartingTurnNumber$,
$NumParticipants$,
$AlicesIDX$,
$NULL$

The purpose of this specification is to outline an algorithm that guarantees that a challenge is registered on chain with *turnNumber* equal to *LatestTurnNumber*. It is guaranteed even with an antagonist who can do anything (including front-run *Alice* an arbitrary number of times) except

- signing data with *Alice*'s private key
- corrupting the blockchain

This guarantee has a key assumption, namely: 1. When a challenge is recorded on the *adjudicator*, *Alice* is always
able to
a) notice the event
b) submit a transaction
c) receive confirmation that that transaction was mined
all before the challenge times out.

If guarantee is met, then either A. the channel concludes at this state; or *B*. someone responds with a move that progresses the channel *C*. someone responds with an alternative move that progresses the
channel

Alice must accept A. She must also accept *C* – indeed, she must accept any alternative round that is recorded on chain, since she must have signed exactly one state in that round, and has no control over what the other participants does after that state. She would be most satisfied with *B*.

In reality, it is possible that *Alice* receives a state with *turnNumber* *LatestTurnNumber* + 1, and in this case *Alice* could (gracefully) abort her algorithm and continue the channel. A future version of this specification could consider this possibility.

By inductively applying her algorithm, *Alice* can therefore guarantee that either the channel progresses as long as she wishes, or it concludes on the latest state that she has.

$LatestTurnNumber \triangleq StartingTurnNumber + NumParticipants - 1$
$AlicesCommitments \triangleq StartingTurnNumber \,..\, LatestTurnNumber$
$ParticipantIDXs \triangleq 1 \,..\, NumParticipants$
$ParticipantIDX(turnNumber) \triangleq 1 + ((turnNumber - 1) \% NumParticipants)$
$AlicesMove(turnNumber) \triangleq ParticipantIDX(turnNumber) = AlicesIDX$

ASSUME
$\land\ StartingTurnNumber \in Nat$
$\land\ NumParticipants \in Nat \setminus \{1\}$
$\land\ AlicesIDX \in ParticipantIDXs$
$\land\ \neg AlicesMove(LatestTurnNumber + 1)$

--algorithm *forceMove*

Alice calls *adjudicator* functions by submitting a pending transaction with the function type and arguments. The *adjudicator* processes this transaction and modifies the channel state on her behalf. However, when *Eve* calls functions, she directly modifies the channel state. This emulates a reality where *Eve* can consistently front-run *Alice*'s transactions, when desired.

```
variables
    $channel = [turnNumber \mapsto [p \in ParticipantIDXs \mapsto 0],\ mode \mapsto ChannelMode.OPEN,\ challenge \mapsto NULL$
    $submittedTX = NULL,$
    $counter = 0$  Auxilliary variable used in some properties and invariants.
      We can't specify any properties that require any memory of the
      behaviour up to the certain point (ie. the behaviour has passed through state $X$ seven times in a row)
      we thus have to embed the "memory" of the behaviour in the state itself,
      if we want to check some property the depends on the history of the behaviour

define
$challengeOngoing \triangleq channel.mode = ChannelMode.CHALLENGE$
$channelOpen \triangleq channel.mode = ChannelMode.OPEN$
$progressesChannel(commitment) \triangleq commitment.turnNumber \geq channel.turnNumber[commitment.signer]$
$validCommitment(c) \triangleq c \in [turnNumber : Nat,\ signer : ParticipantIDXs]$
$validTransition(commitment) \triangleq$
    $\land\ commitment.turnNumber = channel.challenge.turnNumber + 1$
    $\land\ commitment.signer = ParticipantIDX(commitment.turnNumber)$
$AlicesGoalMet \triangleq$
    $\land\ channel.mode = ChannelMode.CHALLENGE$
    $\land\ channel.challenge.turnNumber = LatestTurnNumber$
end define ;

macro $clearChallenge(turnNumber)$
begin
assert $turnNumber \in Nat$ ;
$channel := [$
    $mode \mapsto ChannelMode.OPEN,$
    $turnNumber \mapsto [p \in ParticipantIDXs \mapsto Maximum(channel.turnNumber[p],\ turnNumber)],$
    $challenge \mapsto NULL$
$]$ ;
end macro ;

macro $respondWithMove(commitment)$
begin
if
    $\land\ challengeOngoing$
    $\land\ validTransition(commitment)$
 then $clearChallenge(commitment.turnNumber)$ ;
end if ;
end macro ;

macro $refute(turnNumber)$
```

```
begin
if
    $\land$ $challengeOngoing$
    $\land$ $ParticipantIDX(turnNumber) = channel.challenge.signer$
    $\land$ $turnNumber > channel.turnNumber[ParticipantIDX(turnNumber)]$
    $\land$ $turnNumber > channel.challenge.turnNumber$
 then
$channel :=$ [
    $mode \mapsto ChannelMode.OPEN$,
    $challenge \quad \mapsto NULL$,
    $turnNumber \mapsto [i \in \{ParticipantIDX(turnNumber)\} \mapsto turnNumber]$ @@ $channel.turnNumber$
     By switching to the following effect, we can see how Eve could infinitely grief
     with the previous version of the force-move protocol.
   $turnNumber \mapsto channel.turnNumber$
] ;
end if ;
end macro ;

macro $forceMove(commitment)$
begin
if
    $\land$ $channelOpen$
    $\land$ $progressesChannel(commitment)$
 then
    $channel := [mode \mapsto ChannelMode.CHALLENGE,\ challenge \mapsto commitment]$ @@ $channel$ ;
     By incrementing the number of forceMoves that have been called, we
     multiply the number of distinct states by a large amount, but we can specify properties like
     "Eve has not submitted 5 force moves"
   $counter := counter + 1$;
end if ;
end macro ;

fair process $adjudicator$ = "Adjudicator"
begin
This process records submitted transactions.
$Adjudicator$:
while $\neg AlicesGoalMet \lor submittedTX \neq NULL$ do
    if $submittedTX \neq NULL$ then
        if    $submittedTX.type\ = TX\_Type.FORCE\_MOVE$ then $forceMove(submittedTX.commitment)$ ;
         elsif $submittedTX.type = TX\_Type.REFUTE$       then $refute(submittedTX.turnNumber)$ ;
         elsif $submittedTX.type = TX\_Type.RESPOND$      then $respondWithMove(submittedTX.commitm$
         else assert FALSE ;
        end if ;
        $submittedTX := NULL$ ;
    end if ;
end while ;
end process ;
```

```
fair process $alice$ = "Alice"
begin
```

Alice has commitments $(n - numParticipants) \mathrel{..} (n - 1)$. She wants to end up with commitments $(n - numParticipants + 1) \mathrel{..} n$.

She is allowed to :
 – Call *submitForceMove* with any states that she currently has
- Call refute with any state that she has
- Call *respondWithMove* whenever there's an active challenge where it's her turn to move

```
$A$:
while $\neg AlicesGoalMet$ do
    await $submittedTX = NULL$;
    if $challengeOngoing$ then with $turnNumber = channel.challenge.turnNumber$ do
        if $turnNumber < StartingTurnNumber$ then
            Alice has signed commitments from StartingTurnNumber up to LastTurnNumber.
            She can therefore call refute with exactly one commitment, according to
            the channel's current turnNumber.
          with $refutation = \text{CHOOSE } n \in AlicesCommitments : ParticipantIDX(n) = channel.challenge.signe$
           do $submittedTX := [type \mapsto TX\_Type.REFUTE,\ turnNumber \mapsto refutation]$; end with ;
         elsif $turnNumber < LatestTurnNumber$ then
            with $response = turnNumber + 1$,
                  $commitment = [turnNumber \mapsto response,\ signer \mapsto ParticipantIDX(response)]$
             do
                assert $response \in AlicesCommitments$;
                $submittedTX := [type \mapsto TX\_Type.RESPOND,\ commitment \mapsto commitment]$;
            end with ;
         else skip;   Alice has run out of allowed actions.
        end if ;
    end with ; else
         $submittedTX := [$
             $commitment \mapsto [turnNumber \mapsto LatestTurnNumber,\ signer \mapsto AlicesIDX]$,
             $type \mapsto TX\_Type.FORCE\_MOVE$
         $]$;
    end if ;
end while ;
end process ;
```

```
fair process $eve$ = "Eve"
begin
```

Eve can do almost anything.

a. She can sign any data with any private key, except she cannot sign a commitment with *Alice*'s private key when the turn number is greater than or equal to *StartingTurnNumber*

b. She can call any *adjudicator* function, at any time *c*. She can front-run any transaction an arbitrary number of times: if
anyone else calls an *adjudicator* function in a transaction tx, she can then choose to submit any transaction before tx is mined.

d. She can choose not to do anything, thus causing any active challenge to expire.

(d) is emulated by behaviours where execution is either *Alice* → *Adjudicator* or *Adjudicator* → *Alice*

```
$E$:
while $\neg AlicesGoalMet$ do
    either
        with $n \in NumParticipants \mathrel{..} LatestTurnNumber$,
              $idx \in ParticipantIDXs \setminus \{AlicesIDX\}$
         do
            $forceMove([turnNumber \mapsto n,\ signer \mapsto idx])$ ;
        end with ;
    or if $challengeOngoing$
         then either with
            $turnNumber = channel.challenge.turnNumber + 1$,
            $commitment = [turnNumber \mapsto turnNumber,\ signer \mapsto ParticipantIDX(turnNumber)]$
         do $respondWithMove(commitment)$ ; end with ;
        or  with $turnNumber \in \{\}$
                Eve can refute with any state she has. Alice has seen all of these states.
               $\cup\ 0 \mathrel{..} LatestTurnNumber$
                Since Eve can sign arbitrary data with any private key other than Alice's,
                she can also refute with arbitrarily states, as long as it's not Alice's
                turn in that state.
               $\cup\ \{n \in Nat : \neg AlicesMove(n)\}$
         do $refute(turnNumber)$ ; end with ;
        end either ;
        end if ;
    end either ;
end while ;
end process ;

end algorithm ;
 BEGIN TRANSLATION
VARIABLES $channel$, $submittedTX$, $counter$, $pc$

 define statement
$challengeOngoing \triangleq channel.mode = ChannelMode.CHALLENGE$
$channelOpen \triangleq channel.mode = ChannelMode.OPEN$
$progressesChannel(commitment) \triangleq commitment.turnNumber \geq channel.turnNumber[commitment.signer]$
$validCommitment(c) \triangleq c \in [turnNumber : Nat,\ signer : ParticipantIDXs]$
$validTransition(commitment) \triangleq$
    $\land\ commitment.turnNumber = channel.challenge.turnNumber + 1$
```

```
      ∧ commitment.signer = ParticipantIDX(commitment.turnNumber)
AlicesGoalMet ≜
      ∧ channel.mode = ChannelMode.CHALLENGE
      ∧ channel.challenge.turnNumber = LatestTurnNumber


vars ≜ ⟨channel, submittedTX, counter, pc⟩

ProcSet ≜ {"Adjudicator"} ∪ {"Alice"} ∪ {"Eve"}

Init ≜  Global variables
        ∧ channel = [turnNumber ↦ [p ∈ ParticipantIDXs ↦ 0], mode ↦ ChannelMode.OPEN, challenge ↦
        ∧ submittedTX = NULL
        ∧ counter = 0
        ∧ pc = [self ∈ ProcSet ↦ CASE self = "Adjudicator" → "Adjudicator"
                                  □  self = "Alice" → "A"
                                  □  self = "Eve" → "E"]

Adjudicator ≜ ∧ pc["Adjudicator"] = "Adjudicator"
              ∧ IF ¬AlicesGoalMet ∨ submittedTX ≠ NULL
                   THEN ∧ IF submittedTX ≠ NULL
                             THEN ∧ IF submittedTX.type = TX_Type.FORCE_MOVE
                                       THEN ∧ IF ∧ channelOpen
                                                 ∧ progressesChannel((submittedTX.commitment))
                                                 THEN ∧ channel′ = [mode ↦ ChannelMode.CHAL
                                                 ELSE ∧ TRUE
                                                      ∧ UNCHANGED channel
                                       ELSE ∧ IF submittedTX.type = TX_Type.REFUTE
                                                 THEN ∧ IF ∧ challengeOngoing
                                                           ∧ ParticipantIDX((submittedTX.turn
                                                           ∧ (submittedTX.turnNumber) > chan
                                                           ∧ (submittedTX.turnNumber) > chan
                                                           THEN ∧ channel′ =          [
                                                                           mode ↦ Chan
                                                                           challenge    ↦
                                                                           turnNumber ↦


                                                                        ]
                                                           ELSE ∧ TRUE
                                                                ∧ UNCHANGED channel
                                                 ELSE ∧ IF submittedTX.type = TX_Type.RESPO
                                                           THEN ∧ IF ∧ challengeOngoing
                                                                     ∧ validTransition((submit
                                                                     THEN ∧ Assert(((submit
                                                                                 "Failure o
```

```
                                                                                                              ∧ channel′ =
                                                                                                                                 m
                                                                                                                                 tu
                                                                                                                                 ch
                                                                                                                            ]
                                                                                                      ELSE   ∧ TRUE
                                                                                                              ∧ UNCHANGED ch
                                                                                       ELSE   ∧ Assert(FALSE,
                                                                                                          “Failure of assertion a
                                                                                               ∧ UNCHANGED channel
                                                  ∧ submittedTX′ = NULL
                                          ELSE   ∧ TRUE
                                                  ∧ UNCHANGED ⟨channel, submittedTX⟩
                                   ∧ pc′ = [pc EXCEPT ![“Adjudicator”] = “Adjudicator”]
                           ELSE   ∧ pc′ = [pc EXCEPT ![“Adjudicator”] = “Done”]
                                   ∧ UNCHANGED ⟨channel, submittedTX⟩
                    ∧ UNCHANGED counter

adjudicator ≜ Adjudicator

A ≜ ∧ pc[“Alice”] = “A”
        ∧ IF ¬AlicesGoalMet
               THEN   ∧ submittedTX = NULL
                       ∧ IF challengeOngoing
                              THEN   ∧ LET turnNumber ≜ channel.challenge.turnNumberIN
                                          IF turnNumber < StartingTurnNumber
                                               THEN   ∧ LET refutation ≜ CHOOSE n ∈ AlicesCommitments : Particip
                                                             submittedTX′ = [type ↦ TX_Type.REFUTE, turnNumber
                                               ELSE   ∧ IF turnNumber < LatestTurnNumber
                                                              THEN   ∧ LET response ≜ turnNumber + 1IN
                                                                           LET commitment ≜ [turnNumber ↦ response,
                                                                              ∧ Assert(response ∈ AlicesCommitments,
                                                                                         “Failure of assertion at line 187, colu
                                                                              ∧ submittedTX′ = [type ↦ TX_Type.RESPO
                                                              ELSE   ∧ TRUE
                                                                      ∧ UNCHANGED submittedTX
                              ELSE   ∧ submittedTX′ =                       [
                                                          commitment ↦ [turnNumber ↦ LatestTurnNumber, sign
                                                          type ↦ TX_Type.FORCE_MOVE
                                                     ]
                       ∧ pc′ = [pc EXCEPT ![“Alice”] = “A”]
               ELSE   ∧ pc′ = [pc EXCEPT ![“Alice”] = “Done”]
                       ∧ UNCHANGED submittedTX
        ∧ UNCHANGED ⟨channel, counter⟩

alice ≜ A
```

```
$E \triangleq \land pc[\text{“Eve”}] = \text{“E”}$
      $\land$ IF $\neg AlicesGoalMet$
            THEN $\land \lor \land \exists\, n \in NumParticipants \,..\, LatestTurnNumber :$
                          $\exists\, idx \in ParticipantIDXs \setminus \{AlicesIDX\} :$
                            IF $\land channelOpen$
                               $\land progressesChannel(([turnNumber \mapsto n, signer \mapsto idx]))$
                               THEN $\land channel' = [mode \mapsto ChannelMode.CHALLENGE, challenge \mapsto ([tu$
                               ELSE $\land$ TRUE
                                    $\land$ UNCHANGED $channel$
                      $\lor \land$ IF $challengeOngoing$
                            THEN $\land \lor \land$ LET $turnNumber \triangleq channel.challenge.turnNumber + 1$IN
                                          LET $commitment \triangleq [turnNumber \mapsto turnNumber, signer \mapsto Pa$
                                            IF $\land challengeOngoing$
                                               $\land validTransition(commitment)$
                                               THEN $\land Assert((commitment.turnNumber) \in Nat,$
                                                                  “Failure of assertion at line 93, column 1 of
                                                    $\land channel' =$            [
                                                                    $mode \mapsto ChannelMode.OPEN,$
                                                                    $turnNumber \mapsto [p \in ParticipantIDX$
                                                                    $challenge \mapsto NULL$
                                                                  ]
                                               ELSE $\land$ TRUE
                                                    $\land$ UNCHANGED $channel$
                                  $\lor \land \exists\, turnNumber \in$                    $\{\}$

                                                   $\cup\, 0 \,..\, LatestTurnNumber$


                                                   $\cup\, \{n \in Nat : \neg AlicesMove(n)\} :$
                                          IF $\land challengeOngoing$
                                             $\land ParticipantIDX(turnNumber) = channel.challenge.signer$
                                             $\land turnNumber > channel.turnNumber[ParticipantIDX(turnN$
                                             $\land turnNumber > channel.challenge.turnNumber$
                                             THEN $\land channel' =$            [
                                                                  $mode \mapsto ChannelMode.OPEN,$
                                                                  $challenge \quad \mapsto NULL,$
                                                                  $turnNumber \mapsto [i \in \{ParticipantIDX$


                                                                ]
                                             ELSE $\land$ TRUE
                                                  $\land$ UNCHANGED $channel$
                            ELSE $\land$ TRUE
                                 $\land$ UNCHANGED $channel$
                $\land pc' = [pc$ EXCEPT $![\text{“Eve”}] = \text{“E”}]$
```

$$\begin{array}{l}
\qquad\qquad \text{ELSE} \quad \land\, pc' = [pc \text{ EXCEPT } ![\text{“Eve”}] = \text{“Done”}] \\
\qquad\qquad\qquad\quad\; \land\, \text{UNCHANGED } channel \\
\qquad \land\, \text{UNCHANGED } \langle submittedTX,\, counter \rangle
\end{array}$$

$$eve \triangleq E$$

Allow infinite stuttering to prevent deadlock on termination.

$$\begin{array}{l}
Terminating \triangleq \land\, \forall\, self \in ProcSet : pc[self] = \text{“Done”} \\
\qquad\qquad\qquad\;\; \land\, \text{UNCHANGED } vars
\end{array}$$

$$\begin{array}{l}
Next \triangleq adjudicator \lor alice \lor eve \\
\qquad\quad \lor\, Terminating
\end{array}$$

$$\begin{array}{l}
Spec \triangleq \land\, Init \land \Box[Next]_{vars} \\
\qquad\quad\; \land\, \mathrm{WF}_{vars}(adjudicator) \\
\qquad\quad\; \land\, \mathrm{WF}_{vars}(alice) \\
\qquad\quad\; \land\, \mathrm{WF}_{vars}(eve)
\end{array}$$

$$Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”})$$

END TRANSLATION

$$\begin{array}{l}
AllowedTurnNumbers \triangleq 0 \,..\, (StartingTurnNumber + NumParticipants) \\
AllowedCommitments \triangleq [turnNumber : AllowedTurnNumbers,\, signer : ParticipantIDXs]
\end{array}$$

$$\begin{array}{l}
AllowedTransactions \triangleq \{NULL\} \\
\quad \cup\, [type : \{TX\_Type.FORCE\_MOVE,\, TX\_Type.RESPOND\},\, commitment : AllowedCommitments] \\
\quad \cup\, [type : \{TX\_Type.REFUTE\},\, turnNumber : AllowedTurnNumbers]
\end{array}$$

$$AllowedChannels \triangleq [turnNumber : [ParticipantIDXs \rightarrow Nat],\, mode : Range(ChannelMode),\, challenge : Allou$$

Safety & liveness properties

$$\begin{array}{l}
TypeOK \triangleq \\
\quad \land\, channel \in AllowedChannels \\
\quad \land\, submittedTX \in AllowedTransactions
\end{array}$$

$$\begin{array}{l}
AliceCanProgressChannel \triangleq \Diamond\Box(\\
\quad \land\, channel.mode = ChannelMode.CHALLENGE \\
\quad \land\, channel.challenge.turnNumber = LatestTurnNumber \\
)
\end{array}$$

We can verify that *Alice* can never directly modify the channel with this property, with the exception that she can finalize the channel.

$$\begin{array}{l}
AliceMustSubmitTransactions \triangleq \Box[\\
\qquad\quad \land\, pc[\text{“Alice”}] = \text{“AliceTakesAction”} \\
\qquad\quad \land\, pc'[\text{“Alice”}] = \text{“AliceMoves”} \\
\quad\; \Rightarrow \text{UNCHANGED } channel \\
]_{\langle pc,\, channel \rangle}
\end{array}$$

$$
\begin{array}{l}
\mathit{TurnNumberIncrements} \triangleq \Box[\\
\quad \forall\, p \in \mathit{ParticipantIDXs} : \mathit{channel}'.\mathit{turnNumber}[p] \geq \mathit{channel}.\mathit{turnNumber}[p] \\
]_{\langle \mathit{channel} \rangle}
\end{array}
$$

It's useful to specify the following invariants or properties, since we can inspect the trace of behaviours that violate them to verify that the model checker is working as intended.

$$\mathit{EveCanGrieveAlice} \triangleq \mathit{counter} < 5$$

Behaviours that violate this property exhibit *Eve*'s ability to front-run: *Alice* always submits a transaction that would change the channel state, if it took effect immediately. Therefore, if the channel state is not changed when a pending transaction is processed, *Eve* must have called a function already.

$$
\begin{array}{l}
\mathit{EveCannotFrontRun} \triangleq \Box[\\
\qquad\quad \land\ \mathit{submittedTX} \neq \mathit{NULL} \\
\qquad\quad \land\ \mathit{submittedTX}' = \mathit{NULL} \\
\quad \Rightarrow \\
\qquad\quad \lor\ \mathit{channel}' \neq \mathit{channel}
\end{array}
$$

By uncommenting the following line, one can inspect traces where *Eve* might have front-run *Alice* multiple times

$$
\begin{array}{l}
\qquad \lor\ \mathit{counter} \leq 3 \\
]_{\langle \mathit{submittedTX},\, \mathit{channel} \rangle}
\end{array}
$$

\* Modification History

\* Last modified *Tue Sep* 10 18:47:34 *MDT* 2019 by *andrewstewart*

\* Created *Tue Aug* 06 14:38:11 *MDT* 2019 by *andrewstewart*