MODULE *ForceMove*

EXTENDS *Integers*, *TLC*, *Utils*

CONSTANTS

StartingTurnNumber,
NumParticipants,
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 checkpoints with a move that progresses the
channel

Alice must accept A. She must also accept *C* – indeed, she must accept any supported state that is recorded on chain, since she must have signed at least one "recent" state in its support, 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.

ASSUME

$$\begin{array}{ll} \land & StartingTurnNumber \in Nat \\ \land & \land\ NumParticipants \in Nat \\ & \land\ NumParticipants > 1 \end{array}$$

--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

$channel = [turnNumber \mapsto 0,\ mode \mapsto ChannelMode.OPEN]$,
$submittedTX = NULL$,

$Alice \in ParticipantIDXs \setminus \{ParticipantIDX(LatestTurnNumber + 1)\},$
$counter = 0$ 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
$Number \triangleq Nat \cup \{0\}$
$LatestTurnNumber \triangleq StartingTurnNumber + NumParticipants - 1$
$ParticipantIDXs \triangleq 1 \,..\, NumParticipants$
$ParticipantIDX(turnNumber) \triangleq 1 + ((turnNumber - 1)\%NumParticipants)$
$Signer(commitment) \triangleq ParticipantIDX(commitment.turnNumber)$

$MainHistoryTurnNumbers \triangleq 0 \,..\, (StartingTurnNumber + NumParticipants)$
$ValidCommitments \triangleq [turnNumber : Nat]$
$AlicesCommitments \triangleq \{c \in ValidCommitments :$
$\quad \wedge c.turnNumber \in MainHistoryTurnNumbers$
$\}$
$StoredCommitments \triangleq \{c \in AlicesCommitments : c.turnNumber \geq StartingTurnNumber\}$

$AlicesNextTurnNumber \triangleq$ CHOOSE $n \in (LatestTurnNumber + 1) \,..\, (LatestTurnNumber + NumParticipants)$
$TargetTurnNumbers \triangleq (LatestTurnNumber + 1) \,..\, (AlicesNextTurnNumber - 1)$
$EvesCommitments \triangleq \{c \in ValidCommitments : c.turnNumber \leq AlicesNextTurnNumber\}$

$challengeOngoing \triangleq channel.mode = ChannelMode.CHALLENGE$
$channelOpen \triangleq channel.mode = ChannelMode.OPEN$

$increasesTurnNumber(commitment) \triangleq commitment.turnNumber > channel.turnNumber$

$validCommitment(c) \triangleq c \in ValidCommitments$

$validTransition(c) \triangleq$
$\quad \wedge challengeOngoing$
$\quad \wedge c.turnNumber = channel.turnNumber + 1$

$AlicesGoalMet \triangleq channel.turnNumber \in TargetTurnNumbers$
end define ;

macro $validateCommitment(c, type)$
begin
if $\neg validCommitment(c)$ **then**
print $(\langle type, c \rangle)$**;**
assert FALSE**;**
end if ;
end macro ;

macro $clearChallenge(turnNumber)$

```
begin
assert turnNumber ∈ Nat ;
channel := [
    mode ↦ ChannelMode.OPEN,
    turnNumber ↦ turnNumber
] ;
end macro ;

macro respondWithMove(commitment)
begin
validateCommitment(commitment, "respond") ;
if validTransition(commitment)
 then clearChallenge(commitment.turnNumber) ;
end if ;
end macro ;

macro checkpoint(commitment)
begin
validateCommitment(commitment, "checkpoint") ;
if increasesTurnNumber(commitment)
 then clearChallenge(commitment.turnNumber) ;
end if ;
end macro ;

macro forceMove(commitment)
begin
validateCommitment(commitment, "forceMove") ;
if
    ∨ ∧ channelOpen
      ∧ commitment.turnNumber ≥ channel.turnNumber
    ∨ ∧ challengeOngoing
      ∧ commitment.turnNumber > channel.turnNumber
 then
    channel := [mode ↦ ChannelMode.CHALLENGE, turnNumber ↦ commitment.turnNumber] ;
     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* $\vee$ *submittedTX* $\neq$ *NULL* **do**
  if *submittedTX* $\neq$ *NULL* **then**
    if *submittedTX.type* = *ForceMoveAPI.FORCE_MOVE* **then** *forceMove*(*submittedTX.commitmen*
    elsif *submittedTX.type* = *ForceMoveAPI.RESPOND* **then** *respondWithMove*(*submittedTX.com*
    elsif *submittedTX.type* = *ForceMoveAPI.CHECKPOINT* **then** *checkpoint*(*submittedTX.commitmen*
    else assert FALSE;
    end if ;
    submittedTX := *NULL*;
  end if ;
end while ;
end process ;

fair + **process** *alice* = "Alice"
begin

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

She is allowed to: A. Call *submitForceMove* with any states that she currently has B. Call *respondWithMove* whenever there's an active challenge where
  it's her turn to move
C. Call checkpoint at any time.

A:
while $\neg$*AlicesGoalMet* **do**
  await *submittedTX* = *NULL*;
  if *challengeOngoing* **then with** *turnNumber* = *channel.turnNumber* **do**
    if *turnNumber* < *LatestTurnNumber* **then**
      Alice has signed commitments from *StartingTurnNumber* up to *LastTurnNumber*.
      She would therefore call *forceMove* with the latest commitment
      with *commitment* = CHOOSE $c \in$ *StoredCommitments* : *c.turnNumber* = *LatestTurnNumber* **do**
      submittedTX := [*type* $\mapsto$ *ForceMoveAPI.FORCE_MOVE*, *commitment* $\mapsto$ *commitment*] **; end with**
    end if ;
  end with ; **else**
    submittedTX := [
      commitment $\mapsto$ [*turnNumber* $\mapsto$ *LatestTurnNumber*],
      type $\mapsto$ *ForceMoveAPI.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
TODO: challenge with more commitments than this
with *commitment* ∈ *EvesCommitments*
do *forceMove*(*commitment*) **;**
end with ;
or if *challengeOngoing*
then either with *commitment* ∈ *EvesCommitments*
do *respondWithMove*(*commitment*) **; end with ;**
or with *commitment* ∈ *EvesCommitments*
do *checkpoint*(*commitment*) **; end with ;**
end either ;
end if ; end either ;
end while ;
end process ;

end algorithm ;

BEGIN TRANSLATION
VARIABLES *channel*, *submittedTX*, *Alice*, *counter*, *pc*

define statement
$Number \triangleq Nat \cup \{0\}$
$LatestTurnNumber \triangleq StartingTurnNumber + NumParticipants - 1$
$ParticipantIDXs \triangleq 1 \,..\, NumParticipants$
$ParticipantIDX(turnNumber) \triangleq 1 + ((turnNumber - 1)\%NumParticipants)$
$Signer(commitment) \triangleq ParticipantIDX(commitment.turnNumber)$

$MainHistoryTurnNumbers \triangleq 0 \,..\, (StartingTurnNumber + NumParticipants)$
$ValidCommitments \triangleq [turnNumber : Nat]$
$AlicesCommitments \triangleq \{c \in ValidCommitments :$
$\quad \wedge\, c.turnNumber \in MainHistoryTurnNumbers$
$\}$
$StoredCommitments \triangleq \{c \in AlicesCommitments : c.turnNumber \geq StartingTurnNumber\}$

$AlicesNextTurnNumber \triangleq$ CHOOSE $n \in (LatestTurnNumber + 1) \,..\, (LatestTurnNumber + NumParticipants)$
$TargetTurnNumbers \triangleq (LatestTurnNumber + 1) \,..\, (AlicesNextTurnNumber - 1)$
$EvesCommitments \triangleq \{c \in ValidCommitments : c.turnNumber \leq AlicesNextTurnNumber\}$

$challengeOngoing \triangleq channel.mode = ChannelMode.CHALLENGE$
$channelOpen \triangleq channel.mode = ChannelMode.OPEN$

$increasesTurnNumber(commitment) \triangleq commitment.turnNumber > channel.turnNumber$

$validCommitment(c) \triangleq c \in ValidCommitments$

$validTransition(c) \triangleq$
$\quad \land challengeOngoing$
$\quad \land c.turnNumber = channel.turnNumber + 1$

$AlicesGoalMet \triangleq channel.turnNumber \in TargetTurnNumbers$

$vars \triangleq \langle channel,\ submittedTX,\ Alice,\ counter,\ pc \rangle$

$ProcSet \triangleq \{\text{“Adjudicator”}\} \cup \{\text{“Alice”}\} \cup \{\text{“Eve”}\}$

$Init \triangleq$ Global variables
$\land channel = [turnNumber \mapsto 0,\ mode \mapsto ChannelMode.OPEN]$
$\land submittedTX = NULL$
$\land Alice \in ParticipantIDXs \setminus \{ParticipantIDX(LatestTurnNumber + 1)\}$
$\land counter = 0$
$\land pc = [self \in ProcSet \mapsto \text{CASE } self = \text{“Adjudicator”} \to \text{“Adjudicator”}$
$\qquad\qquad\qquad\qquad\qquad\qquad \square \quad self = \text{“Alice”} \to \text{“A”}$
$\qquad\qquad\qquad\qquad\qquad\qquad \square \quad self = \text{“Eve”} \to \text{“E”}]$

$Adjudicator \triangleq \land pc[\text{“Adjudicator”}] = \text{“Adjudicator”}$
$\land \text{IF } \neg AlicesGoalMet \lor submittedTX \neq NULL$
$\text{THEN } \land \text{IF } submittedTX \neq NULL$
$\text{THEN } \land \text{IF } submittedTX.type = ForceMoveAPI.FORCE\_MOVE$
$\text{THEN } \land \text{IF } \neg validCommitment((submittedTX.commitment))$
$\text{THEN } \land PrintT((\langle \text{“forceMove”},\ (submittedTX.com$
$\land Assert(\text{FALSE},$
$\text{“Failure of assertion at line 109, co$
$\text{ELSE } \land \text{TRUE}$
$\land \text{IF } \lor \land channelOpen$
$\land (submittedTX.commitment).turnNumber \geq ch$
$\lor \land challengeOngoing$
$\land (submittedTX.commitment).turnNumber > ch$
$\text{THEN } \land channel' = [mode \mapsto ChannelMode.CHA1$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } channel$
$\text{ELSE } \land \text{IF } submittedTX.type = ForceMoveAPI.RESPOND$
$\text{THEN } \land \text{IF } \neg validCommitment((submittedTX.com$
$\text{THEN } \land PrintT((\langle \text{“respond”},\ (submit$
$\land Assert(\text{FALSE},$
$\text{“Failure of assertion a$

ELSE ∧ TRUE
∧ IF *validTransition*((*submittedTX*.*commit*
THEN ∧ *Assert*(((*submittedTX*.*comm*
"Failure of assertion a
∧ *channel*′ = [
mode ↦ *Chan*
turnNumber ↦
]
ELSE ∧ TRUE
∧ UNCHANGED *channel*
ELSE ∧ IF *submittedTX*.*type* = *ForceMoveAPI*.*C*
THEN ∧ IF ¬*validCommitment*((*subm*
THEN ∧ *PrintT*(((⟨"check
∧ *Assert*(FALSE,
"Failure c
ELSE ∧ TRUE
∧ IF *increasesTurnNumber*((*su*
THEN ∧ *Assert*(((*submit*
"Failure c
∧ *channel*′ =
m
tu
]
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 ⟨*Alice*, *counter*⟩

adjudicator ≜ *Adjudicator*

A ≜ ∧ *pc*[“Alice”] = “A”
∧ IF ¬*AlicesGoalMet*
THEN ∧ *submittedTX* = *NULL*
∧ IF *challengeOngoing*
THEN ∧ LET *turnNumber* ≜ *channel*.*turnNumber*IN
IF *turnNumber* < *LatestTurnNumber*
THEN ∧ LET *commitment* ≜ CHOOSE *c* ∈ *StoredCommitments* : *c*.*tur*

```
                                                    submittedTX' = [type ↦ ForceMoveAPI.FORCE_MOVE, c
                                        ELSE  ∧ TRUE
                                              ∧ UNCHANGED submittedTX
                                  ELSE  ∧ submittedTX' =                [
                                                          commitment ↦ [turnNumber ↦ LatestTurnNumber],
                                                          type ↦ ForceMoveAPI.FORCE_MOVE
                                                        ]
                                        ∧ pc' = [pc EXCEPT ![“Alice”] = “A”]
                                  ELSE  ∧ pc' = [pc EXCEPT ![“Alice”] = “Done”]
                                        ∧ UNCHANGED submittedTX
                              ∧ UNCHANGED ⟨channel, Alice, counter⟩

alice ≜ A

E ≜  ∧ pc[“Eve”] = “E”
     ∧ IF ¬AlicesGoalMet
          THEN  ∧ ∨ ∧ ∃ commitment ∈ EvesCommitments :
                      ∧ IF ¬validCommitment(commitment)
                           THEN  ∧ PrintT((⟨“forceMove”, commitment⟩))
                                 ∧ Assert(FALSE,
                                          “Failure of assertion at line 109, column 5 of macro called at
                           ELSE  ∧ TRUE
                      ∧ IF  ∨ ∧ channelOpen
                              ∧ commitment.turnNumber ≥ channel.turnNumber
                            ∨ ∧ challengeOngoing
                              ∧ commitment.turnNumber > channel.turnNumber
                           THEN  ∧ channel' = [mode ↦ ChannelMode.CHALLENGE, turnNumber ↦
                           ELSE  ∧ TRUE
                                 ∧ UNCHANGED channel
                  ∨ ∧ IF challengeOngoing
                        THEN  ∧ ∨ ∧ ∃ commitment ∈ EvesCommitments :
                                    ∧ IF ¬validCommitment(commitment)
                                         THEN  ∧ PrintT((⟨“respond”, commitment⟩))
                                               ∧ Assert(FALSE,
                                                        “Failure of assertion at line 109, column 5 o
                                         ELSE  ∧ TRUE
                                    ∧ IF validTransition(commitment)
                                         THEN  ∧ Assert((commitment.turnNumber) ∈ Nat,
                                                        “Failure of assertion at line 115, column 1 o
                                               ∧ channel' =              [
                                                              mode ↦ ChannelMode.OPEN,
                                                              turnNumber ↦ (commitment.turnN
                                                            ]
                                         ELSE  ∧ TRUE
                                               ∧ UNCHANGED channel
```

$$
\begin{array}{l}
\lor\ \land \exists\, commitment \in EvesCommitments : \\
\quad \land \text{IF } \neg validCommitment(commitment) \\
\quad\quad \text{THEN } \land PrintT((\langle \text{“checkpoint”}, commitment\rangle)) \\
\quad\quad\quad \land Assert(\text{FALSE}, \\
\quad\quad\quad\quad \text{“Failure of assertion at line 109, column 5 o} \\
\quad\quad\quad \text{ELSE } \land \text{TRUE} \\
\quad \land \text{IF } increasesTurnNumber(commitment) \\
\quad\quad \text{THEN } \land Assert((commitment.turnNumber) \in Nat, \\
\quad\quad\quad\quad \text{“Failure of assertion at line 115, column 1 o} \\
\quad\quad\quad \land channel' = [\\
\quad\quad\quad\quad mode \mapsto ChannelMode.OPEN, \\
\quad\quad\quad\quad turnNumber \mapsto (commitment.turnN \\
\quad\quad\quad\quad] \\
\quad\quad \text{ELSE } \land \text{TRUE} \\
\quad\quad\quad \land \text{UNCHANGED } channel \\
\quad\quad \text{ELSE } \land \text{TRUE} \\
\quad\quad\quad \land \text{UNCHANGED } channel \\
\quad\quad \land pc' = [pc \text{ EXCEPT } ![\text{“Eve”}] = \text{“E”}] \\
\quad \text{ELSE } \land pc' = [pc \text{ EXCEPT } ![\text{“Eve”}] = \text{“Done”}] \\
\quad\quad \land \text{UNCHANGED } channel \\
\land \text{UNCHANGED } \langle submittedTX,\ Alice,\ counter\rangle
\end{array}
$$

$eve \triangleq E$

Allow infinite stuttering to prevent deadlock on termination.

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

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

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

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

END TRANSLATION

$AllowedTransactions \triangleq [type : Range(ForceMoveAPI),\ commitment : ValidCommitments]$
$AllowedChannels \triangleq [mode : Range(ChannelMode),\ turnNumber : Number]$

Safety & liveness properties

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

$AliceCanProgressChannel \triangleq \Diamond\Box(channel.turnNumber \in TargetTurnNumbers)$

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

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

$$TurnNumberIncrements \triangleq \Box[$$
$$channel'.turnNumber \geq channel.turnNumber$$
$$]_{\langle channel\rangle}$$

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.

$$EveCanGrieveAlice \triangleq counter < 5$$

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.

$$EveCannotFrontRun \triangleq \Box[$$
$$\quad \land submittedTX \neq NULL$$
$$\quad \land submittedTX' = NULL$$
$$\Rightarrow$$
$$\quad \lor channel' \neq channel$$

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

$$\quad \lor counter \leq 3$$
$$]_{\langle submittedTX,\, channel\rangle}$$

* Modification History
* Last modified *Wed Oct* 30 10:01:58 *NDT* 2019 by *andrewstewart*
* Created *Tue Aug* 06 14:38:11 *MDT* 2019 by *andrewstewart*