MODULE *CommitAdopt*

This is a specification of the commit-adopt algorithm of Gafni and Losa in the message-adversary model with dynamic participation. The specification is written in PlusCal and TLA+.

The message-adversary model with dynamic participation is like the sleepy model, except that processes never fail; instead, the adversary corrupts their messages. This has the same effect as processes being faulty but is cleaner to model.

Note that, to check this specification with the TLC model-checker, you must first translate the PlusCal algorithm to TLA+ using the TLA toolbox or the TLA+ VSCode extension.

EXTENDS $Naturals$, $FiniteSets$, $Utilities$

CONSTANTS
  P the set of processors
, V the set of possible values
, Bot the special value "bottom", indicating the absence of something
, $Lambda$ the failure notification "lambda"
, $NoCommit$ an indication that a processors didn't see a unanimous majority in round 1 of the algorithm

ASSUME $Distinct(\langle P,\ V,\ \{Bot\},\ \{Lambda\},\ \{NoCommit\}\rangle)$

--algorithm CA**{**
variables
  $input \in [P \rightarrow V]$ **;** the processors' inputs
  $sent = [p \in P \mapsto Bot]$ **;** messages sent in the current round
  message received by p from q in the current round; Bot means no message received:
  $received = [p \in P \mapsto [q \in P \mapsto Bot]]$ **;**
  $rnd = 1$ **;** the current round (1, 2, or 3); we end at 3 but nothing happens in round 3
  the processors' outputs (either Bot, $\langle$"commit", $v\rangle$, or $\langle$"adopt", $v\rangle$) for some v
  $output = [p \in P \mapsto Bot]$ **;**
define {
  $TypeOkay \triangleq$
    $\land\ input \in [P \rightarrow V]$
    $\land\ sent \in [P \rightarrow V \cup \{Bot,\ NoCommit\}]$
    $\land\ received \in [P \rightarrow [P \rightarrow V \cup \{Bot,\ NoCommit,\ Lambda\}]]$
    $\land\ rnd \in \{1,\ 2,\ 3\}$
    $\land\ output \in [P \rightarrow \{Bot\} \cup \{\langle ca,\ v\rangle : ca \in \{\text{"commit"},\ \text{"adopt"}\},\ v \in V\}]$
  the set of processors from which p received a message (*i.e.* heard of):
  $HeardOf(p) \triangleq \{q \in P : received[p][q] \neq Bot\}$
  the set of minority subsets of S:
  $Minority(S) \triangleq \{M \in \text{SUBSET } S : 2 * Cardinality(M) < Cardinality(S)\}$
  the number of votes for v that p received:
  $VoteCount(p,\ v) \triangleq Cardinality(\{q \in P : received[p][q] = v\})$
  the set of values v for which p received a strict majority of votes:
  $VotedByMajority(p) \triangleq \{v \in V : 2 * VoteCount(p,\ v) > Cardinality(HeardOf(p))\}$
  the set containing the value voted for most often, if any, according to p's received messages:

$MostVotedFor(p) \triangleq \{v \in V : \forall\, w \in V \setminus \{v\} : VoteCount(p, v) > VoteCount(p, w)\}$

for technical reasons, we need the program counter of a processor in round r:

$Pc(r) \triangleq$ CASE $r = 1 \rightarrow$ "r1"
$\quad \Box\, r = 2 \rightarrow$ "r2"
$\quad \Box\, r = 3 \rightarrow$ "r3"

Now we give the two safety properties:

$Agreement \triangleq \forall\, p, q \in P : output[p] \neq Bot \land output[q] \neq Bot \land output[p][1] =$ "commit"
$\quad \Rightarrow output[p][2] = output[q][2]$

$Validity \triangleq \forall\, p \in P : \forall\, v \in V :$
$\quad pc[p] =$ "Done" $\land (\forall\, q \in P : input[q] = v) \Rightarrow output[p] = \langle$"commit", $v\rangle$

}

macro *broadcast*(v) **{**
$\quad sent := [sent$ EXCEPT $![self] = v]$
}

The following macro is used to deliver messages to the processors. It includes message corruptions by the adversary:

macro *deliver_msgs*(*participating*, *corrupted*) **{**
$\quad$ **with** ($ByzMsg \in [P \rightarrow [corrupted \rightarrow V \cup \{Bot, Lambda, NoCommit\}]]$) **{**
$\quad\quad$ we assert the properties of the no-equivocation model:
$\quad\quad$ **when** $\forall\, p1, p2 \in P : \forall\, q \in corrupted :$
$\quad\quad\quad ByzMsg[p1][q] \in V \Rightarrow ByzMsg[p2][q] \in \{ByzMsg[p1][q], Lambda\}$ **;**
$\quad\quad received := [p \in P \mapsto [q \in P \mapsto$
$\quad\quad\quad$ IF $q \in corrupted$
$\quad\quad\quad$ THEN $ByzMsg[p][q]$ $\quad$ p receives a corrupted message
$\quad\quad\quad$ ELSE IF $q \in participating$
$\quad\quad\quad\quad$ THEN $sent[q]$ $\quad$ p receives what q sent
$\quad\quad\quad\quad$ ELSE $Bot]]$ $\quad$ p receives nothing
$\quad$ **}**
}

Now we give the specification of the algorithm:

fair process ($proc \in P$) **{**
$\quad$ in round 1, vote for $input[self]$:

$r1$: $\quad broadcast(input[self])$ **;**

$r2$: $\quad$ **await** $rnd = 2$ **;**
$\quad$ if there is a majority for a value v, propose to commit v:
$\quad$ **if** ($VotedByMajority(self) \neq \{\}$)
$\quad\quad$ **with** ($v \in VotedByMajority(self)$) $\quad$ the set is a singleton at this point
$\quad\quad broadcast(v)$
$\quad$ **else**
$\quad\quad broadcast(NoCommit)$ **;**

$r3$: $\quad$ **await** $rnd = 3$ **;** $\quad$ in round 3 we just produce an output
$\quad$ **if** ($VotedByMajority(self) \neq \{\}$) $\quad$ if there is a majority for a value v, commit v:
$\quad\quad$ **with** ($v \in VotedByMajority(self)$) $\quad$ the set is a singleton at this point
$\quad\quad output[self] := \langle$"commit", $v\rangle$
$\quad$ **else if** ($MostVotedFor(self) \neq \{\}$) $\quad$ otherwise, adopt a most voted value:

with ($v \in MostVotedFor(self)$) there can be multiple values in the set
$output[self] := \langle$ "adopt", $v \rangle$
else if no value was voted for, adopt an arbitrary value:
with ($v \in V$)
$output[self] := \langle$ "adopt", $v \rangle$
}

Below we specify the behavior of the adversary. The no-equivocation model guarantees that if a processor receives v from p, then all receive v or $Lambda$.

fair process ($adversary \in \{$ "adversary" $\}$) **{**
adv: **while** ($rnd < 3$) **{**
await $\forall p \in P : pc[p] = Pc(rnd + 1)$ **;**
pick a participating set and a set of corrupted processors:
with ($Participating \in \text{SUBSET } P \setminus \{\{\}\}$)
with ($Corrupted \in Minority(Participating)$)
$deliver\_msgs(Participating, Corrupted)$ **;**
$rnd := rnd + 1$ **;**
}
}
}

Canary invariants that should break (this is to make sure that the specification reaches expected states):

To find a state in which some process outputs:

$Canary1 \triangleq \forall p \in P : output[p] = Bot$

To find a state in which some process commits while another adopts:

$Canary2 \triangleq \forall p, q \in P :$
$\land\, output[p] \neq Bot$
$\land\, output[q] \neq Bot$
$\Rightarrow \neg(output[p][1] = \text{"commit"} \land output[q][1] = \text{"adopt"})$

To find a state in which two processes adopt different values:

$Canary3 \triangleq \forall p, q \in P :$
$\land\, output[p] \neq Bot$
$\land\, output[q] \neq Bot$
$\Rightarrow \neg(output[p][1] = \text{"adopt"} \land output[q][1] = \text{"adopt"} \land output[p][2] \neq output[q][2])$

\ * Modification History
\ * Last modified Sun *Jan* 01 16:07:58 *PST* 2023 by *nano*
\ * Created *Thu Dec* 29 09:54:34 *PST* 2022 by *nano*