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*

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

$Distinct(s) \triangleq \forall\, i, j \in \text{DOMAIN } s : i \neq j \Rightarrow s[i] \cap s[j] = \{\}$
ASSUME $Distinct(\langle P,\ V,\ \{Bot\},\ \{Lambda\},\ \{NoCommit\}\rangle)$

--algorithm *CA*{
variables
$input \in [P \to V]$; 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
$output = [p \in P \mapsto Bot]$; the processors' outputs
define {
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 of values v that were voted for the most often according to p:
$MostVotedFor(p) \triangleq \{v \in V : \forall\, w \in V \setminus \{v\} : VoteCount(p,\ v) \geq VoteCount(p,\ w)\}$
for technical reasons, we need the program counter of a processor in round r:
$Pc(r) \triangleq$ CASE $r = 1 \to$ "r1"
$\square\, r = 2 \to$ "r2"
$\square\, r = 3 \to$ "r3"
Now we give the two safety properties:

```
    Agreement ≜ ∀ p, q ∈ P : output[p] ≠ Bot ∧ output[q] ≠ Bot ∧ output[p][1] = "commit"
        ⇒ output[p][2]   = output[q][2]
    Validity ≜ ∀ p ∈ P : ∀ v ∈ V :
        pc[p] = "Done" ∧ (∀ q ∈ P : input[q] = v) ⇒ output[p] = ⟨"commit", v⟩
}
macro broadcast( v ) {
    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( participanting, corrupted ) {
    with ( ByzMsg ∈ [P → [corrupted → V ∪ {Bot, Lambda, NoCommit}]] ) {
        we assert the properties of the no-equivocation model:
        when ∀ p1, p2 ∈ P : ∀ q ∈ corrupted :
                ByzMsg[p1][q] ∈ V ⇒ ByzMsg[p2][q] ∈ {ByzMsg[p1][q], Lambda};
        received := [p ∈ P ↦ [q ∈ P ↦
            IF q ∈ corrupted
            THEN ByzMsg[p][q]  p receives a corrupted message
            ELSE IF q ∈ participating
                THEN sent[q]  p receives what q sent
                ELSE Bot  p receives nothing
    };
}
Now we give the specification of the algorithm:
fair process ( proc ∈ P ) {
    in round 1, vote for input[self]:
r1: broadcast(input[self]);
r2: await rnd = 2;
    if there is a majority for a value v, propose to commit v:
    if ( VotedByMajority(self) ≠ {} )
        with ( v ∈ VotedByMajority(self) )  the set is a singleton at this point
        broadcast(v)
    else
        broadcast(NoCommit);
r3: await rnd = 3;  in round 3 we just produce an output
    if ( VotedByMajority(self) ≠ {} )  if there is a majority for a value v, commit v:
        with ( v ∈ VotedByMajority(self) )  the set is a singleton at this point
        output[self] := ⟨"commit", v⟩
    else if ( MostVotedFor(self) ≠ {} )  otherwise, adopt a most voted value:
        with ( v ∈ MostVotedFor(self) )  there can be multiple values in the set
        output[self] := ⟨"adopt", v⟩
    else  if no value was voted for, adopt input:
        output[self] := ⟨"adopt", input[self]⟩
}
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 ∈ {"adversary"} ) {
adv:    while ( rnd < 3 ) {
            await ∀ p ∈ P : pc[p] = Pc(rnd + 1) ;
             pick a participating set and a set of corrupted processors:
            with ( Participating ∈ SUBSET P \ {{}} )
            with ( Corrupted ∈ Minority(participating[rnd]) )
                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:

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

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

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

* 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*