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
$input \in [P \rightarrow V]$ **;** the processors' inputs
$sent = [p \in P \mapsto Bot]$ **;** messages sent in the current round
$received = [p \in P \mapsto [q \in P \mapsto Bot]]$ **;** message received by p from q in the current round
$rnd = 1$ **;** 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
$participating = [r \in \{1, 2\} \mapsto \{\}]$ **;** the set of participating processors in round r
$corrupted = [r \in \{1, 2\} \mapsto \{\}]$ **;** the set of corrupted processors in round r
define {
first we make some auxiliary definitions
the set of processors from which p received a message:
$HeardOf(rcvd) \triangleq \{p \in P : rcvd[p] \neq Bot\}$
the set of minority subsets of S:
$Minority(S) \triangleq \{M \in \text{SUBSET } S : 2 * Cardinality(M) < Cardinality(S)\}$
$VoteCount(rcvd, v) \triangleq Cardinality(\{p \in P : rcvd[p] = v\})$
$VotedByMajority(rcvd) \triangleq \{v \in V : 2 * VoteCount(rcvd, v) > Cardinality(HeardOf(rcvd))\}$
$MostVotedFor(rcvd) \triangleq \{v \in V : \forall w \in V \setminus \{v\} : VoteCount(rcvd, v) \geq VoteCount(rcvd, w)\}$
for technical reasons, we need the program counter of a processor in round r:
$Pc(r) \triangleq \text{CASE } r = 1 \rightarrow$ "r1"
$\Box\, r = 2 \rightarrow$ "r2"
$\Box\, r = 3 \rightarrow$ "r3"
Now 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"

$\Rightarrow output[p][2] \quad = output[q][2]$

$Validity \triangleq \forall p \in P : \forall v \in V :$

$pc[p] = \text{“Done”} \wedge (\forall q \in P : input[q] = v) \Rightarrow output[p] = \langle \text{“commit”}, v \rangle$

}

macro $broadcast(\ v\ )$ **{**

$sent := [sent \text{ EXCEPT } ![self] = v]$

}

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

macro $deliver\_msgs(\ )$ **{**

with ($ByzMsg \in [P \rightarrow [corrupted[rnd] \rightarrow V \cup \{Bot, Lambda, NoCommit\}]]$ **) {**

we assert the properties of the no-equivocation model:

when $\forall p1, p2 \in P : \forall q \in corrupted[rnd] :$

$ByzMsg[p1][q] \in V \Rightarrow ByzMsg[p2][q] \in \{ByzMsg[p1][q], Lambda\}$ **;**

$received := [p \in P \mapsto [q \in P \mapsto$

$\text{IF } q \in corrupted[rnd]$

$\text{THEN } ByzMsg[p][q]$

$\text{ELSE } sent[q]]]$ **;**

} ;

}

Now the specification of the algorithm:

fair process ($proc \in P$ **) {**

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 ($VotedByMajority(received[self]) \neq \{\}$ **)**

with ($v \in VotedByMajority(received[self])$ **)** the set is a singleton at this point

$broadcast(v)$

else

$broadcast(NoCommit)$ **;**

$r3$: **await** $rnd = 3$ **;**

if ($VotedByMajority(received[self]) \neq \{\}$ **)** if there is a majority for a value v, commit v:

with ($v \in VotedByMajority(received[self])$ **)** the set is a singleton at this point

$output[self] := \langle \text{“commit”}, v \rangle$

else if ($MostVotedFor(received[self]) \neq \{\}$ **)** otherwise, adopt a most voted value:

with ($v \in MostVotedFor(received[self])$ **)** there can be multiple values in the set

$output[self] := \langle \text{“adopt”}, v \rangle$

else if no value was voted for, adopt input:

$output[self] := \langle \text{“adopt”}, input[self] \rangle$

}

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 ($adversary \in \{\text{“adversary”}\}$ **) {**

adv: **while (** $rnd < 3$ **) {**

await $\forall p \in P : pc[p] = Pc(rnd + 1)$ **;**

pick a participating set:

with ($Participating \in \text{SUBSET } P$) **{**

  when $Participating \neq \{\}$ **;**

  $participating[rnd] := Participating$ **;**

} **;**

pick a set of corrupted processors:

with ($Corrupted \in Minority(participating[rnd])$)

  $corrupted[rnd] := Corrupted$ **;**

$deliver\_msgs()$ **;**

$rnd := rnd + 1$ **;**

}

}

}

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

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

$Canary2 \triangleq \forall p, q \in P :$

$\quad \land output[p] \neq Bot$

$\quad \land output[q] \neq Bot$

$\quad \Rightarrow \neg(output[p][1] = \text{"commit"} \land output[q][1] = \text{"adopt"})$

$Canary3 \triangleq \forall p, q \in P :$

$\quad \land output[p] \neq Bot$

$\quad \land output[q] \neq Bot$

$\quad \Rightarrow \neg(output[p][1] = \text{"adopt"} \land output[q][1] = \text{"adopt"} \land output[p][2] \neq output[q][2])$

\ * 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*