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"
$\Box r = 2 \rightarrow$ "r2"
$\Box r = 3 \rightarrow$ "r3"

Now we give the two safety properties:

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

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

}

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

Now we give the specification of the algorithm:

fair process ($proc \in P$) {
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 ($VotedByMajority(self) \neq \{\}$)
with ($v \in VotedByMajority(self)$) 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 ($VotedByMajority(self) \neq \{\}$) if there is a majority for a value v, commit v:
with ($v \in VotedByMajority(self)$) the set is a singleton at this point
$output[self] := \langle$"commit", $v\rangle$
else if ($MostVotedFor(self) \neq \{\}$) otherwise, adopt a most voted value:

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

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

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:

$\mathit{Canary1} \triangleq \forall\, p \in P : \mathit{output}[p] = \mathit{Bot}$

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

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

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

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

* 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*