MODULE $AB2$

This is a modification of spec AB in which instead of losing messages, messages are detectably "corrupted"–represented by being changed to the value Bad. The to communication channels are represented by the variables $AtoB2$ and $BtoA2$.

EXTENDS $Integers$, $Sequences$, TLC

CONSTANT $Data$, Bad

ASSUME $Bad \notin (Data \times \{0, 1\}) \cup \{0, 1\}$

We need to asssume that Bad is different from any of the legal messsages.

VARIABLES $AVar$, $BVar$, The same as in module $ABSpec$
$AtoB2$, The sequence of data messages in transit from sender to receiver
$BtoA2$ The sequence of ack messages in transit from receiver to sender
Messages are sent by appending them to the end of the sequence and received by removing them from the head of the sequence.

$vars \triangleq \langle AVar, BVar, AtoB2, BtoA2 \rangle$

$$\begin{aligned} TypeOK \triangleq\ & \land AVar \in Data \times \{0, 1\} \\ & \land BVar \in Data \times \{0, 1\} \\ & \land AtoB2 \in Seq((Data \times \{0, 1\}) \cup \{Bad\}) \\ & \land BtoA2 \in Seq(\{0, 1, Bad\}) \end{aligned}$$

$$\begin{aligned} Init \triangleq\ & \land AVar \in Data \times \{1\} \\ & \land BVar = AVar \\ & \land AtoB2 = \langle\rangle \\ & \land BtoA2 = \langle\rangle \end{aligned}$$

The action of the sender sending a data message by appending $AVar$ to the end of the message queue $AtoB2$. It will keep sending the same message until it receives an acknowledgment for it from the receiver.

$$\begin{aligned} ASnd \triangleq\ & \land AtoB2' = Append(AtoB2, AVar) \\ & \land \text{UNCHANGED } \langle AVar, BtoA2, BVar\rangle \end{aligned}$$

The action of the sender receiving an ack message. If that ack is for the value it is sending, then it chooses another message to send and sets $AVar$ to that message. If the ack is for the previous value it sent, it ignores the message. In either case, it removes the message from $BtoA2$. Note that Bad cannot equal $AVar[2]$, which is in $\{0, 1\}$.

$$\begin{aligned} ARcv \triangleq\ & \land BtoA2 \neq \langle\rangle \\ & \land \text{IF } Head(BtoA2) = AVar[2] \\ & \quad \text{THEN } \exists\, d \in Data : AVar' = \langle d, 1 - AVar[2]\rangle \\ & \quad \text{ELSE } AVar' = AVar \\ & \land BtoA2' = Tail(BtoA2) \\ & \land \text{UNCHANGED } \langle AtoB2, BVar\rangle \end{aligned}$$

The action of the receiver sending an acknowledgment message for the last data item it received.

$$\begin{aligned} BSnd \triangleq\ & \land BtoA2' = Append(BtoA2, BVar[2]) \\ & \land \text{UNCHANGED } \langle AVar, BVar, AtoB2\rangle \end{aligned}$$

The action of the receiver receiving a data message. It ignores a *Bad* message. Otherwise, it sets *BVar* to the message if it's not for the data item it has already received.

$$
\begin{array}{l}
BRcv \triangleq \land\ AtoB2 \neq \langle\rangle \\
\qquad \land\ \text{IF } (Head(AtoB2) \neq Bad) \land (Head(AtoB2)[2] \neq BVar[2]) \\
\qquad\qquad \text{THEN } BVar' = Head(AtoB2) \\
\qquad\qquad \text{ELSE } BVar' = BVar \\
\qquad \land\ AtoB2' = Tail(AtoB2) \\
\qquad \land\ \text{UNCHANGED } \langle AVar,\ BtoA2\rangle
\end{array}
$$

CorruptMsg is the action that changes an arbitrary message in *AtoB*2 or *BtoA*2 to *Bad*. (We don't bother testing if the message in *AtoB*2 already equals *Bad*, since setting to *Bad* a message that already equals *Bad* is just a stuttering step.)

$$
\begin{array}{l}
CorruptMsg \triangleq \land \lor \land \exists\, i \in 1 \,..\, Len(AtoB2) : \\
\qquad\qquad\qquad\qquad AtoB2' = [AtoB2 \text{ EXCEPT } ![i] = Bad] \\
\qquad\qquad\qquad \land\ BtoA2' = BtoA2 \\
\qquad\qquad \lor \land \exists\, i \in 1 \,..\, Len(BtoA2) : \\
\qquad\qquad\qquad\qquad BtoA2' = [BtoA2 \text{ EXCEPT } ![i] = Bad] \\
\qquad\qquad\qquad \land\ AtoB2' = AtoB2 \\
\qquad\qquad \land\ \text{UNCHANGED } \langle AVar,\ BVar\rangle
\end{array}
$$

$$Next \triangleq ASnd \lor ARcv \lor BSnd \lor BRcv \lor CorruptMsg$$

$$Spec \triangleq Init \land \Box[Next]_{vars}$$

$$ABS \triangleq \text{INSTANCE } ABSpec$$

THEOREM $Spec \Rightarrow ABS!Spec$

FairSpec is the analogue of formula *FairSpec* of module *AB*2. That is, it is obtained by conjoining to formula *Spec* the fairness conditions that correspond to the ones in module *AB*2. However, specification *FairSpec* of this module does not implement $ABS!FairSpec$. You can use *TLC* to find a behavior in which no new values are ever sent.

$$
\begin{array}{l}
FairSpec \triangleq \\
\quad Spec \land \mathrm{SF}_{vars}(ARcv) \land \mathrm{SF}_{vars}(BRcv) \land \mathrm{WF}_{vars}(ASnd) \land \mathrm{WF}_{vars}(BSnd)
\end{array}
$$

A little thought reveals that, since messages are corrupted but not deleted, strong fairness of *ARcv* and *BRcv* is equivalent to weak fairness of those actions. The shortest counterexample showing that *FairSpec* does not implement $ABS!FairSpec$, which is probably the one found by *TLC*, is a behavior in which a message is sent on an empty message channel, but is always corrupted before it can received. This suggests that in addition to weak fairness of *ARcv* and *BRcv*, we want strong fairness of those actions when the head of the queue is not corrupt. That leads to the following spec.

$$
\begin{array}{l}
FairSpec2 \triangleq \\
\quad Spec \land \mathrm{WF}_{vars}(ARcv) \land \mathrm{WF}_{vars}(BRcv) \land \mathrm{WF}_{vars}(ASnd) \land \mathrm{WF}_{vars}(BSnd) \\
\qquad \land\ \mathrm{SF}_{vars}(ARcv \land Head(BtoA2) \neq Bad) \\
\qquad \land\ \mathrm{SF}_{vars}(BRcv \land Head(AtoB2) \neq Bad)
\end{array}
$$

Running *TLC* shows that *FairSpec*2 also does not implement $ABS!FairSpec$. In fact, I believe that it is impossible to obtain a specification that implements $ABS!FairSpec$ by conjoining to *Spec* fairness conditions on subactions of *Next*. Module $AB2P$ shows how we can modify the $AB2$ specification to obtain a specification that implements $ABS!Spec$.

We define *RemoveBad* so that $RemoveBad(seq)$ is the value obtained by removing from the sequence *seq* all elements that equal *Bad*.

RECURSIVE $RemoveBad(\_)$

$$RemoveBad(seq) \triangleq$$
IF $seq = \langle\rangle$
THEN $\langle\rangle$
ELSE $(\text{IF } Head(seq) = Bad \text{ THEN } \langle\rangle \text{ ELSE } \langle Head(seq)\rangle)$
$\circ\ RemoveBad(Tail(seq))$

There's an easy way to define *RemoveBad* using the *SelectSeq* operator of the *Sequences* module. Here's the alternative definition.

$$RemoveBadAlt(seq) \triangleq \text{LET } Test(elt) \triangleq elt \neq Bad \text{ IN } SelectSeq(seq, Test)$$

The following statement defines $AB!Spec$ to be the specification *Spec* of module *AB* with $RemoveBad(AtoB2)$ substituted for *AtoB* and $RemoveBad(BtoA2)$ substituted for *BtoA*.

$$AB \triangleq \text{INSTANCE } AB \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2),\ BtoA \leftarrow RemoveBad(BtoA2)$$

The following theorem asserts that the specification *Spec* of this module implements the specification *Spec* of module *AB* under the refinement mapping that substitutes $RemoveBad(AtoB2)$ for *AtoB* and substitutes for every other variable and every constant of module *AB* the variable or constant of the same name. This theorem is checked by having *TLC* check that the temporal property $AB!Spec$ is satisfied by the specification *Spec*.

THEOREM $Spec \Rightarrow AB!Spec$

---

* Modification History
* Last modified Sat *Jun* 11 21:33:05 *CST* 2022 by *wengjialin*
* Last modified *Wed Jan* 24 16:33:07 *PST* 2018 by *lamport*
* Created *Wed Mar* 25 11:53:40 *PDT* 2015 by *lamport*