MODULE $AB2P$

This is a version of specification $AB2$ modified so it implements the fairness requirement of the high-level AB specification in module $ABSpec$, which asserts that new values keep being sent and received. For $AB2$ to satisfy that fairness requirement, when a process keeps sending messages to the other process, at least one of those messages must not be corrupted.

It seems to be impossible to express this requirement by adding fairness conditions on subactions of the next-state action of $AB2$. To allow the requirement to be expressed with fairness conditions, the current spec adds two variables $AtoBgood$ and $BtoAgood$. The value of $AtoBgood$ controls which messages in $AtoBgood$ may be corrupted, and the value of $BtoAgood$ does the same for messages in $BtoAgood$.

The value of $AtoBgood$ is a sequence of Boolean values having the same length as $AtoB2$. A value is appended to the end of $AtoBgood$ whenever a message is appended to the end of $AtoB2$; and a message is removed from the head of $AtoBgood$ whenever a message or Bad is removed from the head of $AtoBgood$. If $AtoBgood[i]$ equals TRUE, then message number i of $AtoB2$ cannot be corrupted. So if TRUE is appended to $AtoBgood$ when a message is appended to $AtoB2$, then that message cannot be corrupted. Similarly, $BtoAgood$ controls whether messages in $BtoA2$ can be corrupted.

The following EXTENDS statement imports all the constant and variable declarations and all the definitions from module $AB2$ (with no renaming).

EXTENDS $AB2$

VARIABLES $AtoBgood$, $BtoAgood$

$varsP \triangleq \langle vars, AtoBgood, BtoAgood \rangle$

The definitions of the type-correctness invariant, initial predicate, and actions of the sender and receiver in the current spec are obtained in a straightforward way by conjoining conditions on the variables $AtoBgood$ and $BtoAgood$ to the corresponding definitions from module $AB2$ (which are imported to the current module by the EXTENDS statement).

$$\begin{aligned} TypeOKP \triangleq\ & \land\ TypeOK \\ & \land\ AtoBgood \in Seq(\text{BOOLEAN}) \\ & \land\ BtoAgood \in Seq(\text{BOOLEAN}) \end{aligned}$$

$$\begin{aligned} InitP \triangleq\ & \land\ Init \\ & \land\ AtoBgood = \langle\rangle \\ & \land\ BtoAgood = \langle\rangle \end{aligned}$$

$$\begin{aligned} ASndP \triangleq\ & \land\ ASnd \\ & \land\ \exists\, b \in \text{BOOLEAN} : AtoBgood' = Append(AtoBgood,\ b) \\ & \land\ \text{UNCHANGED}\ BtoAgood \end{aligned}$$

$$\begin{aligned} ARcvP \triangleq\ & \land\ ARcv \\ & \land\ BtoAgood' = Tail(BtoAgood) \\ & \land\ \text{UNCHANGED}\ AtoBgood \end{aligned}$$

$$\begin{aligned} BSndP \triangleq\ & \land\ BSnd \\ & \land\ \exists\, b \in \text{BOOLEAN} : BtoAgood' = Append(BtoAgood,\ b) \\ & \land\ \text{UNCHANGED}\ AtoBgood \end{aligned}$$

$$BRcvP \triangleq \land BRcv \\ \land AtoBgood' = Tail(AtoBgood) \\ \land \text{UNCHANGED } BtoAgood$$

The *CorruptMsg* action of module *AB* is modified by adding an enabling condition that allows a message in $AtoB2$ or $BtoA2$ to be corrupted only if the corresponding element of *AtoBgood* or *BtoAgood* equals FALSE; and by requiring *AtoBgood* and *BtoAgood* to be unchanged.

$$\begin{aligned} CorruptMsgP \triangleq \; & \land \lor \land \exists\, i \in 1 \,..\, Len(AtoB2) : \\ & \qquad \land \neg AtoBgood[i] \\ & \qquad \land AtoB2' = [AtoB2 \text{ EXCEPT } ![i] = Bad] \\ & \quad \land BtoA2' = BtoA2 \\ & \lor \land \exists\, i \in 1 \,..\, Len(BtoA2) : \\ & \qquad \land \neg BtoAgood[i] \\ & \qquad \land BtoA2' = [BtoA2 \text{ EXCEPT } ![i] = Bad] \\ & \quad \land AtoB2' = AtoB2 \\ & \land \text{UNCHANGED } \langle AVar,\, BVar,\, AtoBgood,\, BtoAgood \rangle \end{aligned}$$

The next-state action and safety spec are named *NextP* and *SpecP*.

$$NextP \triangleq ASndP \lor ARcvP \lor BSndP \lor BRcvP \lor CorruptMsgP$$

$$SpecP \triangleq InitP \land \Box[NextP]_{varsP}$$

It's clear that every assignment of values to the variables of module $AB2$ that satisfies *InitP* also satisfies the initial predicate *Init* of $AB2$, and every change to the variables of $AB2$ allowed by *NextP* is also allowed by the next-state relation Next of $AB2$. Hence *SpecP* implements the specification *Spec* of $AB2$.

THEOREM $SpecP \Rightarrow Spec$

Since *Spec* implements the specification $ABS!Spec$ of module *ABSpec*, we deduce the following theorem from $SpecP \Rightarrow Spec$. (The definition of $ABS!Spec$ is imported into the current module by the EXTENDS statement, along with all the other definitions from module $AB2$.)

THEOREM $SpecP \Rightarrow ABS!Spec$

We now obtain the spec *FairSpecP* by conjoining fairness conditions to *SpecP*. Because messages are not deleted, weak fairness conditions on the receive actions ensure that every sent message or its corrupted *Bad* replacement is eventually received. To ensure that an uncorrupted version of every message eventually is received, we add fairness conditions not for the sending actions *ASndP* and *BSndP*, but for those sending actions that append TRUE to *AtoBgood* or *BtoAgood*.

Note that a subaction of the next-state action *NextP* is any formula that implies *NextP*. It doesn't have to be a disjunct in the definition of *NextP*. Thus the two actions

$ASndP \land AtoBgood'[Len(AtoBgood')]$

$BSndP \land BtoAgood'[Len(BtoAgood')]$

are subactions of *NextP*, just like the actions *ARcvP* and *BRcvP*.

$$FairSpecP \triangleq \land SpecP \\ \land \text{WF}_{vars}(ARcvP) \\ \land \text{WF}_{vars}(BRcvP)$$

$$\land \mathrm{WF}_{vars}(ASndP \land AtoBgood'[Len(AtoBgood')])$$
$$\land \mathrm{WF}_{vars}(BSndP \land BtoAgood'[Len(BtoAgood')])$$

The following theorem asserts that *FairSpecP* implements specification *FairSpec* of module *ABSpec* under the expected refinement mapping. *TLC* can check this theorem.

THEOREM $FairSpecP \Rightarrow ABS\,!\,FairSpec$

* Modification History
* Last modified Sat *Jun* 11 21:42:45 *CST* 2022 by *wengjialin*
* Last modified *Fri Jan* 26 16:30:35 *PST* 2018 by *lamport*
* Created Sun *Jan* 21 16:11:54 *PST* 2018 by *lamport*