MODULE *ReliableBroadcastFD*

EXTENDS *LearnerGraph*, *FiniteSets*

CONSTANTS

LG, the learner graph

B, the set of malicious acceptors

W, the set of well-behaved acceptors, *i.e.* honest and available

V the set of values that can be broadcast

ASSUME $B \cap W = \{\}$

ASSUME $IsValidLearnerGraph(LG)$

$Learner \triangleq LG.learners$
$Acceptor \triangleq LG.acceptors$

$HonestAcceptor \triangleq Acceptor \setminus B$

Note that *HonestAcceptor* is not necessary equal to W

--algorithm *ReliableBroadcast* **{**

Global variables:

variables

$bcast \in (\text{SUBSET } V) \setminus \{\{\}\}$ **;** the value(s) broadcast; multiple values model a malicious sender

$echo = [a \in Acceptor \mapsto \{\}]$ **;** echo messages sent by the acceptors

ready messages sent by the acceptors; note that acceptors get ready per learner:

$ready = [a \in Acceptor \mapsto [l \in Learner \mapsto \{\}]]$ **;**

$fd = [a \in Acceptor \mapsto \{\}]$ **;** failure-detector output

define {

$NotEntangled(self, l1, l2) \triangleq$

$\land\ l1 \neq l2$ a learner is always entangled with itself

$l1$ and $l2$ are don't have to agree if none of their safe sets are completely well-behaved:

$\land\ \forall S \in LG.safeSets[\langle l1, l2 \rangle] :$

$\exists a \in S : a \in fd[self]$

}

Each process get outputs from a failure-detector that eventually identifies all malicious processes (we enforce this eventuality by making the failure-detector process below a fair process):

fair process ($fd \in \{$"failure-detector"$\}$ **)**

{

$l0$: **while (** $\exists a \in HonestAcceptor : \exists b \in B : b \notin fd[a]$ **)**

with ($a \in HonestAcceptor, b \in B$ **) {** NOTE notation idea: while some () {}

when $b \notin fd[a]$ **;**

$fd[a] := fd[a] \cup \{b\}$ **;**

}

}

```
    fair process ( learner ∈ Learner )
        variables
            output = ⟨⟩ ;
    {
l0:     with ( v ∈ V ) {
            when ∃ Q ∈ LG.quorums[self] :
                ∀ a ∈ Q : v ∈ ready[a][self] ;
            output := v ;
        }
    }
    process ( acceptor ∈ HonestAcceptor ) {
l0:     while ( TRUE )
        either   echo a value
            with ( v ∈ V ) {
                when v ∈ bcast ∧ echo[self] = {} ;
                echo[self] := echo[self] ∪ {v} ;
            }
        or   send ready when witnessing a quorum of echoes
            with ( v  ∈ V )
            with ( l  ∈ Learner )
            with ( Q ∈ LG.quorums[l] ) {
                when ready[self][l] = {} ;
                when ∀ a ∈ Q : v  ∈ echo[a] ;
                 check for conflicts (NOTE needed for safety):
                when ∀ l2 ∈ Learner : ∀ v2 ∈ V \ {v} :
                    v2 ∈ ready[self][l2] ⇒ NotEntangled(self, l, l2) ;
                ready[self][l] := ready[self][l] ∪ {v} ;
            }
        or   send ready when blocked by ready acceptors
            with ( v ∈ V )
            with ( l1 ∈ Learner, l2 ∈ Learner ) {
                when v ∈ bcast ;
                when ready[self][l1] = {} ;   no good . . . but no good without it either (liveness fails in both cases)
                when ∀ Q ∈ LG.quorums[l1] :
                    ∃ a2  ∈ Q : v ∈ ready[a2][l2] ;
                 check for conflicts:
                when ∀ l3 ∈ Learner \ {l1} : ∀ v2 ∈ V \ {v} :
                    v2 ∈ ready[self][l3] ⇒ NotEntangled(self, l1, l3) ;
                ready[self][l1] := ready[self][l1] ∪ {v} ;
            }
        }
```

```
    process ( byzAcceptor ∈ B ) {
l0:     while ( TRUE ) {
            either
            with ( v ∈ V )
                echo[self] := echo[self] ∪ {v}
            or
            with ( l ∈ Learner ) {
                with ( v ∈ V )
                    ready[self][l] := ready[self][l] ∪ {v} ;
             }
        }
    }
}
```

$TypeOK \triangleq$
$\land\ bcast \in (\text{SUBSET } V) \setminus \{\{\}\}$
$\land\ echo \in [Acceptor \to (\text{SUBSET } V)]$
$\land\ ready \in [Acceptor \to [Learner \to (\text{SUBSET } V)]]$
$\land\ output \in [Learner \to V \cup \{\langle\rangle\}]$

Two learners must agree if one of their safe sets is fully well-behaved:

$Entangled(l1,\ l2) \triangleq \exists\, S \in LG.safeSets[\langle l1,\ l2\rangle] :$
$S \cap B = \{\}$

$LiveLearner \triangleq \{l \in Learner :$
$\exists\, Q \in LG.quorums[l] : Q \subseteq W\}$

$Safety \triangleq$
$\land\ \forall\, l \in Learner :$
$\quad\land\ pc[l] = \text{“Done”}$
$\quad\land\ \exists\, Q \in LG.quorums[l] : Q \cap B = \{\}\ \ SafeLearner$
$\quad\Rightarrow output[l] \in bcast$
$\land\ \forall\, l1,\ l2 \in Learner :$
$\quad\land\ Entangled(l1,\ l2)$
$\quad\land\ pc[l1] = \text{“Done”}$
$\quad\land\ pc[l2] = \text{“Done”}$
$\quad\Rightarrow output[l1] = output[l2]$

$Liveness \triangleq$
$\land\ Cardinality(bcast) = 1 \Rightarrow$
$\qquad\forall\, l \in LiveLearner : \Diamond(pc[l] = \text{“Done”} \land bcast = \{output[l]\})$

This one is interesting (I think this is the best we can guarantee):

$\land\ \forall\, l1 \in Learner : \forall\, l2 \in LiveLearner : Entangled(l1,\ l2) \Rightarrow$
$\qquad\Box(pc[l1] = \text{“Done”} \Rightarrow \Diamond(pc[l2] = \text{“Done”}))$

$FairSpec \triangleq$

$\land$ $Spec$
$\land$ $\forall\, a \in W : \mathrm{WF}_{vars}(acceptor(a))$