MODULE *ReliableBroadcast*

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{
  variables
    bcast ∈ (SUBSET V) \ {{}} ;   the value(s) broadcast; multiple values model a malicious sender
    echo = [a ∈ Acceptor ↦ {}] ;
    ready = [a ∈ Acceptor ↦ [l ∈ Learner ↦ {}]] ;
  define {
    ProvenMalicious(a) ≜ ∃ v1, v2 ∈ V :
        ∧ v1 ≠ v2
        ∧ ∨ {v1, v2} ⊆ echo[a]
          ∨ ∃ l ∈ Learner : {v1, v2} ⊆ ready[a][l]
  }
  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
            with ( v ∈ V ) {
                when v ∈ bcast ∧ echo[self] = {} ;
                echo[self] := echo[self] ∪ {v} ;
            }
        or
            with ( v  ∈ V )
            with ( l  ∈ Learner )
            with ( Q ∈ LG.quorums[l] ) {
                when ready[self][l] = {} ;
                when ∀ a ∈ Q : v  ∈ echo[a] ;
                ready[self][l] := ready[self][l] ∪ {v} ;
            }
        or
            with ( v ∈ V )
            with ( l ∈ Learner )
            with ( readyForV = {a ∈ Acceptor : v ∈ ready[a][l]} ) {
                when ∀ Q ∈ LG.quorums[l] :
                    ∨  Q ∩ readyForV ≠ {}
                     TODO: this is wrong:
                    ∨  ∃ a ∈ Q : ProvenMalicious(a) ;
                ready[self][l] := ready[self][l] ∪ {v} ;
            }
    }
    process ( byzAcceptor ∈ B ) {
l0:     while ( TRUE ) {
            with ( v ∈ V )
                echo[self] := echo[self] ∪ {v} ;
            with ( rdy ∈ [Learner → V] ) {
                ready[self] := [l ∈ Learner ↦ ready[self][l] ∪ {rdy[l]}] ;
            }
        }
    }
}
```

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

Two learners must agree if one of their safe sets is fully well-behaved:
$Entangled(l1,\ l2) \triangleq \exists\, S \in LG.safeSets[\langle l1,\ l2\rangle] :$
$\quad S \cap B = \{\}$

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

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

$Liveness \triangleq$
$\quad \land \ Cardinality(bcast) = 1 \Rightarrow$
$\quad\quad\quad \forall\, l \in LiveLearner : \Diamond(pc[l] = \text{“Done”} \land bcast = \{output[l]\})$
This one is interesting (I think this is the best we can guarantee):
$\quad \land \ \forall\, l1,\ l2 \in LiveLearner : Entangled(l1,\ l2) \Rightarrow$
$\quad\quad\quad \Box(pc[l1] = \text{“Done”} \Rightarrow \Diamond(pc[l2] = \text{“Done”}))$

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