```
/* BYZANTINE AGREEMENT PROTOCOL */
/* this files contains the template of the byzantine agreement protocol */
/* gxn 7/12/2001 */
/* replaced the probabilistic choice of the coins by a non-deterministic choice */
/* and suppose that the coin in round r is flipped when at least n − 2t honest parties have read the main votes in round r */
/* note that the coin can never be tossed earlier than this since n − t shares are required before it can be revealed */
/* note that we have change the notations for the coins: for round r the coin is denoted coin[r + 1] */
/* this is to simplify the code: it allows the pre-votes for each round to be based on the coin for the previous round */
/* by setting the coin for round -1 (coin[0]) always to 0 */

/* instead of parties reading votes individually, we store the total number of honest votes for each preference */
/* and for corrupted parties we include only boolean variables indicating what corrupted parties can vote for */
/* recalling that different parties may see corrupted parties vote for different values */
/* based on these totals we then know what combination of votes each party can read */
/* that is what the adversary can make parties vote for */
/* note that sometimes there will be a non-deterministic choice as to what a party will vote for */
/* this is a choice of the adversary since it depends on which votes a party reads */

/* an honest party must get at least n − 2t votes from other honest parties */
/* and the remaining votes can come from either honest or corrupted parties */
/* if we can show: “if an honest party has a vote for a value then corrupted parties can also vote for this value” */
/* we need only count up to n − 2t honest party votes and use corrupted parties for the remaining votes */
/* this has been proved (properties corrupted5 and corrupted6 in lemmas.smv) */

/* note that to deal with the validity part of the proof and to keep things consistent the initial round is 0 instead of 1 */
/* this is because the round variable is an ordset and cannot specify round=1, but we can specify round=0 */

/* one further change is that we let the pre-votes be cast before they are cast in the actual protocol */
/* now they are cast when a party reads the main-votes of the previous round */
/* note that this can only give the adversary more power */
/* however, in the original protocol at the time when the pre-votes are now cast */
/* an honest party has read the main-votes it will use to cast a pre-vote */
/* therefore at this time the adversary already knows what pre-vote the party is going make */
/* and hence moving the casting of pre-votes to this point makes no difference to what the adversary knows */
/* there is one added complication: we need to store two different possible pre-votes which depend on */
/* what value of the corresponding coin becomes when it is tossed */

/*——————————————————————————————————————*/

/* CONSTANTS (n=10 & t=3) */
#define N 7 /* number of honest parties */
#define M 4 /* M = n − 2t number of honest pre and main votes that must be read */
#define K 4 /* K = 2t + 1 − t = t + 1 number of honest pre processing votes that must be read */

/* ABSTRACT DATA TYPES */
ordset PROC 1..N; /* set of parties */
ordset VOTES 0..M; /* honest pre and main votes */
ordset PRE_PROC_VOTES 0..K; /* honest preprocessing votes */
ordset ROUNDS 0..; /* round numbers */


/* DATA TYPES */
typedef PC {INITIAL , PRE_PROCESSING , MAIN , DECIDE , COIN , DONE };
/* INITIAL - make preprocessing vote (initial state) */
/* PRE_PROCESSING - read preprocessing votes and make initial pre-vote */
/* MAIN - read pre-votes and make main vote */
/* DECIDE - read main votes and make choice whether to decide or not and next pre-vote */
/* COIN - generate share of the coin */
/* DONE - finished */
```

```
/*————————————————————————————————————————*/

module main(act){

/*———————————————INPUTS——————————————————*/

act : PROC; /* SCHEDULER left unspecified */

/* in certain cases a party can cast different votes depending on which votes it reads */
/* we represent these choices using the following variables */
/* note since this is decided by the adversary we leave these variables unspecified */
/* pre votes */
nd_pre : {0,1}; /* non deterministic choice between voting for 0 and 1 */
nd_pre1 : {0,2}; /* non deterministic choice between voting for 0 and coin */
nd_pre2 : {1,2}; /* non deterministic choice between voting for 1 and coin */
nd_pre3 : {0,1,2}; /* non deterministic choice between voting for 1 and coin */
/* main votes */
nd_main1 : {0,2}; /* non deterministic choice between voting for 0 and 2 */
nd_main2 : {1,2}; /* non deterministic choice between voting for 1 and 2 */

start : array PROC of 0..1; /* initial values of parties */
forall (i in PROC) next(start[i]) := start[i]; /* initial values do not change (needed for validity) */

/* ——————————THE PROTOCOL FOR HONEST PARTIES——————————————- */

/* PHASE */
pc : array PROC of PC; /* pc[i] - phase of party i */

/* VALUES OF VOTES */
/* pre-processing-votes: 0 or 1 */
/* pre-votes: 0 or 1 */
/* main-votes: 0, 1 or 2 where 2 denotes abstain */

/* PRE_PROCESSING */
pre_proc : PRE_PROC_VOTES; /* pre_proc - number of pre processing votes */
pre_proc_votes : array 0..1 of PRE_PROC_VOTES; /* pre_proc_votes[v] - number of pre processing votes for v */

/* PRE_VOTE */
pre : array ROUNDS of VOTES; /* pre[r] - number of pre-votes in round r */
pre_votes : array ROUNDS of array 0..1 of array 0..1 of VOTES;
/* pre_votes[r][c][v] - number of pre-votes for v in round r if the coin in round r equals c */

/* MAIN_VOTE */
main : array ROUNDS of VOTES; /* main[r] - number of main-votes so far in round r */
main_votes : array ROUNDS of array 0..2 of VOTES; /* main_votes[r][v] - number of main-votes so far for v in round r */

/* ROUND */
round : array PROC of ROUNDS; /* round[i] - round party i is currently in */

/* DECIDING */
decide : array ROUNDS of array PROC of boolean; /* decide[r][i] party i decided in round r */
decide_value : array ROUNDS of array PROC of 0..1;
/* decide[r][i][v]=v what value party i decided on in round r (if the party decides) */

/* COIN */
coin : array ROUNDS of { not_tossed , 0 , 1 }; /* coin[r] value of the coin in round r */

/* INITIAL VALUES */
forall (i in PROC) {
  init(pc[i]) := INITIAL;
  init(round[i]) := 0;
}
/* set all totals to be zero */
init(pre_proc) := 0;
for(v = 0; v ≤ 1; v = v + 1) init(pre_proc_votes[v]) := 0;
forall(r in ROUNDS){
  init(pre[r]) := 0;
  init(main[r]) := 0;
  for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) init(pre_votes[r][c][v]) := 0;
  for(v = 0; v ≤ 2; v = v + 1) init(main_votes[r][v]) := 0;
}
/* initially undecided */
forall(r in ROUNDS) forall(i in PROC) init(decide[r][i]) := 0;

/* as explained the coin in round 0 is set to 0 then never changed */
/* the remaining coins initially are not tossed */
forall(r in ROUNDS) init(coin[r]) := (r=0) ? 0 : not_tossed;

/* NEXT VALUES */
switch (pc[act]) {
INITIAL : {
  next(pc[act]) := PRE_PROCESSING; /* move to pre processing */
  /* make pre processing vote based on the value of start[act] */
  next(pre_proc_votes[start[act]]) := (pre_proc_votes[start[act]])=K ? pre_proc_votes[start[act]] : pre_proc_votes[start[act]]+1;
  /* record that a pre processing vote has been cast */
  next(pre_proc) := pre_proc=K ? pre_proc : pre_proc+1;
}
PRE_PROCESSING : {
  if (pre_proc=K) { /* sufficient pre processing votes */
    next(pc[act]) := MAIN; /* move to MAIN */
    /* record that a pre vote has been cast */
    next(pre[round[act]]) := pre[round[act]]=M ? pre[round[act]] : pre[round[act]]+1;
    /* sufficient votes to have a pre vote for either value therefore the vote is the adversary's choice */
    if (pre_proc_votes[0]>0 ∧ pre_proc_votes[1]>0) {
      next(pre_votes[round[act]][0][nd_pre]) :=
            pre_votes[round[act]][0][nd_pre]=M ? pre_votes[round[act]][0][nd_pre] : pre_votes[round[act]][0][nd_pre]+1;
      next(pre_votes[round[act]][1][nd_pre]) :=
            pre_votes[round[act]][1][nd_pre]=M ? pre_votes[round[act]][1][nd_pre] : pre_votes[round[act]][1][nd_pre]+1;
    }
    /* pre vote for 0 only */
    else if (pre_proc_votes[0]>0) {
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]][c][0]) := pre_votes[round[act]][c][0]=M ? pre_votes[round[act]][c][0] : pre_votes[round[act]][c][0]+1;
    }
    /* pre vote for 1 only */
    else if (pre_proc_votes[1]>0) {
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]][c][1]) := pre_votes[round[act]][c][1]=M ? pre_votes[round[act]][c][1] : pre_votes[round[act]][c][1]+1;
    }
    /* the above should be the only cases but as a default suppose the adversary can choose */
    else {
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]][c][nd_pre]) :=
            pre_votes[round[act]][c][nd_pre]=M ? pre_votes[round[act]][c][nd_pre] : pre_votes[round[act]][c][nd_pre]+1;
    }
  }
}
MAIN : {
  /* read pre votes and cast main vote */
  /* to simplify the conditions have used the assumptions that there cannot be M votes for two distinct values in the same round */
  if ( pre[round[act]]=M ∧ ¬coin[round[act]]=not_tossed ) { /* sufficient pre votes and the coin has been tossed */
    next(pc[act]) := DECIDE; /* move to DECIDE */

    /* record that the party has made a main vote */
    next(main[round[act]]) := main[round[act]]=M ? main[round[act]] : main[round[act]]+1;
    /* main vote for 0 or abstain only */
    if (pre_votes[round[act]][coin[round[act]]][0]=M ∧ corrupted_pre[round[act]][coin[round[act]]][0] ∧
      (pre_votes[round[act]][coin[round[act]]][1]>0 ∨ corrupted_pre[round[act]][coin[round[act]]][1]) )
      next(main_votes[round[act]][nd_main1]) :=
              main_votes[round[act]][nd_main1]=M ? main_votes[round[act]][nd_main1] : main_votes[round[act]][nd_main1]+1;
    /* main vote for 1 or abstain only */
    else if (pre_votes[round[act]][coin[round[act]]][1]=M ∧ corrupted_pre[round[act]][coin[round[act]]][1] ∧
      (pre_votes[round[act]][coin[round[act]]][0]>0 ∨ corrupted_pre[round[act]][coin[round[act]]][0]) )
      next(main_votes[round[act]][nd_main2]) :=
              main_votes[round[act]][nd_main2]=M ? main_votes[round[act]][nd_main2] : main_votes[round[act]][nd_main2]+1;
    /* main vote for 0 only */
    else if (pre_votes[round[act]][coin[round[act]]][0]=M ∧ corrupted_pre[round[act]][coin[round[act]]][0])
      next(main_votes[round[act]][0]) := main_votes[round[act]][0]=M ? main_votes[round[act]][0] : main_votes[round[act]][0]+1;
    /* main vote for 1 only */
    else if (pre_votes[round[act]][coin[round[act]]][1]=M ∧ corrupted_pre[round[act]][coin[round[act]]][1])
      next(main_votes[round[act]][1]) := main_votes[round[act]][1]=M ? main_votes[round[act]][1] : main_votes[round[act]][1]+1;
    /* left with only a main vote for abstain */
    else
      next(main_votes[round[act]][2]) := main_votes[round[act]][2]=M ? main_votes[round[act]][2] : main_votes[round[act]][2]+1;
  }

}
DECIDE : {
  /* to simplify the model we allow decide_value to be set without decide being set to true */
  /* in the case the value of decide_value does not matter */
  /* and again use the assumptions that there cannot be M votes for two distinct values in the same round */
  if (main[round[act]]=M) { /* sufficient main votes */
    next(pc[act]) := COIN;
    /* record that a pre vote has been cast */
    next(pre[round[act]+1]) := pre[round[act]+1]=M ? pre[round[act]+1] : pre[round[act]+1]+1;
    /* must decide on 0 and cast a pre-vote for 0 */
    if (main_votes[round[act]][0]=M ∧ corrupted_main[round[act]][0] ∧ main_votes[round[act]][1]=0 ∧
          ¬corrupted_main[round[act]][1] ∧ main_votes[round[act]][2]=0 ∧ ¬corrupted_main[round[act]][2] ) {
      next(decide[round[act]][act]) := 1;
      next(decide_value[round[act]][act]) := 0;
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][0]) :=
              pre_votes[round[act]+1][c][0]=M ? pre_votes[round[act]+1][c][0] : pre_votes[round[act]+1][c][0]+1;
    }
    /* must decide on 1 and cast a pre-vote for 1 */
    else if (main_votes[round[act]][1]=M ∧ corrupted_main[round[act]][1] ∧ main_votes[round[act]][0]=0 ∧
          ¬corrupted_main[round[act]][0] ∧ main_votes[round[act]][2]=0 ∧ ¬corrupted_main[round[act]][2] ) {
      next(decide[round[act]][act]) := 1;
      next(decide_value[round[act]][act]) := 1;
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][1]) :=
              pre_votes[round[act]+1][c][1]=M ? pre_votes[round[act]+1][c][1] : pre_votes[round[act]+1][c][1]+1;
    }
    /* may decide on 0 and must cast a pre-vote for 0 */
    else if (main_votes[round[act]][0]=M ∧ corrupted_main[round[act]][0] ∧
          main_votes[round[act]][1]=0 ∧ ¬corrupted_main[round[act]][1] ) {
      next(decide[round[act]][act]) := {0,1};
      next(decide_value[round[act]][act]) := 0;
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][0]) :=
              pre_votes[round[act]+1][c][0]=M ? pre_votes[round[act]+1][c][0] : pre_votes[round[act]+1][c][0]+1;
    }
    /* may decide on 1 and must cast a pre-vote for 1 */
    else if (main_votes[round[act]][1]=M ∧ corrupted_main[round[act]][1] ∧
          main_votes[round[act]][0]=0 ∧ ¬corrupted_main[round[act]][0] ) {
      next(decide[round[act]][act]) := {0,1};
      next(decide_value[round[act]][act]) := 1;
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][1]) :=
              pre_votes[round[act]+1][c][1]=M ? pre_votes[round[act]+1][c][1] : pre_votes[round[act]+1][c][1]+1;
    }
    /* may decide on 0 and can cast a pre-vote for 0 or 1 */
    else if (main_votes[round[act]][0]=M ∧ corrupted_main[round[act]][0]) {
      next(decide[round[act]][act]) := {0,1};
      next(decide_value[round[act]][act]) := 0;
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][0]) :=
              pre_votes[round[act]+1][c][nd_pre]=M ? pre_votes[round[act]+1][c][nd_pre] : pre_votes[round[act]+1][c][nd_pre]+1;
    }
    /* may decide on 1 and can cast a pre-vote for 0 or 1 */
    else if (main_votes[round[act]][1]=M ∧ corrupted_main[round[act]][1]) {
      next(decide[round[act]][act]) := {0,1};
      next(decide_value[round[act]][act]) := 1;
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][0]) :=
              pre_votes[round[act]+1][c][nd_pre]=M ? pre_votes[round[act]+1][c][nd_pre] : pre_votes[round[act]+1][c][nd_pre]+1;
    }
    /* cannot decide and must cast a pre-vote for 0 */
    else if ( (main_votes[round[act]][0]>0 ∨ corrupted_main[round[act]][0]) ∧ main_votes[round[act]][1]=0 ∧
          ¬corrupted_main[round[act]][1] ∧ (main_votes[round[act]][2]<M ∨ ¬corrupted_main[round[act]][2]) ) {
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][0]) :=
              pre_votes[round[act]+1][c][0]=M ? pre_votes[round[act]+1][c][0] : pre_votes[round[act]+1][c][0]+1;
    }
    /* cannot decide and must cast a pre-vote for 1 */
    else if ( (main_votes[round[act]][1]>0 ∨ corrupted_main[round[act]][1]) ∧ main_votes[round[act]][0]=0 ∧
          ¬corrupted_main[round[act]][0] ∧ (main_votes[round[act]][2]<M ∨ ¬corrupted_main[round[act]][2]) ) {
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][1]) :=
              pre_votes[round[act]+1][c][1]=M ? pre_votes[round[act]+1][c][1] : pre_votes[round[act]+1][c][1]+1;
    }
    /* cannot decide and may cast a pre-vote for 0 or 1 */
    else if ( (main_votes[round[act]][0]>0 ∨ corrupted_main[round[act]][0]) ∧
          (main_votes[round[act]][1]>0 ∨ corrupted_main[round[act]][1]) ∧
          (main_votes[round[act]][2]<M ∨ ¬corrupted_main[round[act]][2]) ) {
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][1]) :=
              pre_votes[round[act]+1][c][nd_pre]=M ? pre_votes[round[act]+1][c][nd_pre] : pre_votes[round[act]+1][c][nd_pre]+1;
    }
    /* cannot decide and may vote for 0, 1 or coin */
    else if ( (main_votes[round[act]][0]>0 ∨ corrupted_main[round[act]][0]) ∧
          (main_votes[round[act]][1]>0 ∨ corrupted_main[round[act]][1]) ) {
      if (nd_pre3=0) { /* concrete vote for 0 */
        for(c = 0; c ≤ 1; c = c + 1)
          next(pre_votes[round[act]+1][c][0]) :=
              pre_votes[round[act]+1][c][0]=M ? pre_votes[round[act]+1][c][0] : pre_votes[round[act]+1][c][0]+1;
    }
      else if (nd_pre3=1) { /* concrete vote for 1 */
        for(c = 0; c ≤ 1; c = c + 1)
```

```
        next(pre_votes[round[act]+1][c][1]) :=
            pre_votes[round[act]+1][c][1]=M ? pre_votes[round[act]+1][c][1] : pre_votes[round[act]+1][c][1]+1;
      }
      else { /* coin */
        for(c = 0; c ≤ 1; c = c + 1)
          next(pre_votes[round[act]+1][c][c]) :=
            pre_votes[round[act]+1][c][c]=M ? pre_votes[round[act]+1][c][c] : pre_votes[round[act]+1][c][c]+1;
      }
    }
    /* cannot decide and may vote for 0 or coin */
    else if (main_votes[round[act]][0]>0 ∨ corrupted_main[round[act]][0]) {
      if (nd_pre1=0) { /* concrete vote for 0 */
        for(c = 0; c ≤ 1; c = c + 1)
          next(pre_votes[round[act]+1][c][0]) :=
            pre_votes[round[act]+1][c][0]=M ? pre_votes[round[act]+1][c][0] : pre_votes[round[act]+1][c][0]+1;
    }
      else { /* coin */
        for(c = 0; c ≤ 1; c = c + 1)
          next(pre_votes[round[act]+1][c][c]) :=
            pre_votes[round[act]+1][c][c]=M ? pre_votes[round[act]+1][c][c] : pre_votes[round[act]+1][c][c]+1;
      }
    }
    /* cannot decide and may vote for 1 or coin */
    else if (main_votes[round[act]][1]>0 ∨ corrupted_main[round[act]][1]) {
      if (nd_pre2=1) { /* concrete vote for 1 */
        for(c = 0; c ≤ 1; c = c + 1)
          next(pre_votes[round[act]+1][c][0]) :=
            pre_votes[round[act]+1][c][1]=M ? pre_votes[round[act]+1][c][1] : pre_votes[round[act]+1][c][1]+1;
    }
      else { /* coin */
        for(c = 0; c ≤ 1; c = c + 1)
          next(pre_votes[round[act]+1][c][c]) :=
            pre_votes[round[act]+1][c][c]=M ? pre_votes[round[act]+1][c][c] : pre_votes[round[act]+1][c][c]+1;
      }
    }
    else {
    /* left with cannot decide and vote for coin */
      for(c = 0; c ≤ 1; c = c + 1)
        next(pre_votes[round[act]+1][c][c]) :=
            pre_votes[round[act]+1][c][c]=M ? pre_votes[round[act]+1][c][c] : pre_votes[round[act]+1][c][c]+1;
    }
  }
}
COIN : {
  /* if decided in previous round then done else continue */
  next(pc[act]) := (round[act]>0 ∧ decide[round[act]-1][act]) ? DONE : MAIN;
  next(round[act]) := (round[act]>0 ∧ decide[round[act]-1][act]) ? round[act] : round[act]+1; /* increase round */
  /* toss coin if at least M honest parties have reached this stage and the coin has not been tossed */
  next(coin[round[act]+1]) := pre[round[act]+1]=M ∧ coin[round[act]+1]=not_tossed ? {0,1} : coin[round[act]+1];
}

}

/* ————————-CORRUPTED PARTIES—————————————— */

/* corrupted_pre[r][c][v] is true if a corrupted party can have a pre vote for v in round r given the coin equals c */
corrupted_pre : array ROUNDS of array 0..1 of array 0..1 of boolean;
/* corrupted_main[r][v] is true if a corrupted party can have a main vote for v in round r */
corrupted_main : array ROUNDS of array 0..2 of boolean;
```

```
/* INITIAL VALUES */
/* initially cannot vote for either */
forall(r in ROUNDS){
for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) init(corrupted_pre[r][c][v]) := 0;
for(v = 0; v ≤ 2; v = v + 1) init(corrupted_main[r][v]) := 0;
}
/* NEXT VALUES */
/* pre votes: for round 0 need at least one preprocessing vote from an honest party for a value to vote for that value */
/* in general to vote v either need a main vote for v (from either an honest or corrupted party) in the previous round */
/* or is able to vote for the coin, that is, there are need sufficient abstain main votes in the previous round */
forall (r in ROUNDS) {
if (r=0) {
  for(v = 0; v ≤ 1; v = v + 1) next(corrupted_pre[r][0][v]) := (next(pre_proc_votes[v])>0);
  for(v = 0; v ≤ 1; v = v + 1) next(corrupted_pre[r][1][v]) := (next(pre_proc_votes[v])>0);
}
else {
  next(corrupted_pre[r][0][0]) :=
      (next(main[r-1])=M ∧ ( (next(main_votes[r-1][0])>0 ∨ next(corrupted_main[r-1][0])) ∨
                      (next(main_votes[r-1][2])=M ∧ next(corrupted_main[r-1][2])) )) ? 1 : 0;
  next(corrupted_pre[r][0][1]) :=
      (next(main[r-1])=M ∧ (next(main_votes[r-1][1])>0 ∨ next(corrupted_main[r-1][1]))) ? 1 : 0;
  next(corrupted_pre[r][1][0]) :=
      (next(main[r-1])=M ∧ (next(main_votes[r-1][0])>0 ∨ next(corrupted_main[r-1][0]))) ? 1 : 0;
  next(corrupted_pre[r][1][1]) :=
      (next(main[r-1])=M ∧ ( (next(main_votes[r-1][1])>0 ∨ next(corrupted_main[r-1][1])) ∨
                      (next(main_votes[r-1][2])=M ∧ next(corrupted_main[r-1][2])) )) ? 1 : 0;
}
}
/* main votes: to vote for v (0 or 1) need M pre votes for v from honest parties also pre votes for v from corrupted parties */
/* to vote for abstain need a pre vote for 0 and a pre vote for 1 (from either honest or corrupted parties) */
forall (r in ROUNDS) {
if (next(coin[r])=0) {
  for(v = 0; v ≤ 1; v = v + 1) next(corrupted_main[r][v]) :=
                      (next(pre[r])=M ∧ next(pre_votes[r][0][v])=M ∧ next(corrupted_pre[r][0][v]) ) ? 1 : 0;
  next(corrupted_main[r][2]) :=
      ( next(pre[r])=M ∧ (next(pre_votes[r][0][0])>0 ∨ next(corrupted_pre[r][0][0]))
           ∧ (next(pre_votes[r][0][1])>0 ∨ next(corrupted_pre[r][0][1])) ) ? 1 : 0;
}
else if (next(coin[r])=1) {
  for(v = 0; v ≤ 1; v = v + 1) next(corrupted_main[r][v]) :=
                      (next(pre[r])=M ∧ next(pre_votes[r][1][v])=M ∧ next(corrupted_pre[r][1][v]) ) ? 1 : 0;
  next(corrupted_main[r][2]) :=
      ( next(pre[r])=M ∧ (next(pre_votes[r][1][0])>0 ∨ next(corrupted_pre[r][1][0]))
           ∧ (next(pre_votes[r][1][1])>0 ∨ next(corrupted_pre[r][1][1])) ) ? 1 : 0;
}
}

/* ————————END OF PROTOCOL———————————— */

/*————————————-HISTORY VARIABLES——————————————*/

/* for each round records the first party to have a main-vote for each value */
history_main_votes : array ROUNDS of array 0..2 of PROC;
forall (r in ROUNDS) for(v = 0; v ≤ 2; v = v + 1) {
  init(history_main_votes[r][v]) := act;
  next(history_main_votes[r][v]) := next(main_votes[r][v])>0 ∧ main_votes[r][v]=0 ? act : history_main_votes[r][v];
}
/* for each round records the first party to have a pre-vote for each value */
history_pre_votes : array ROUNDS of array 0..1 of array 0..1 of PROC;
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) {
  init(history_pre_votes[r][c][v]) := act;
  next(history_pre_votes[r][c][v]) := next(pre_votes[r][c][v])>0 ∧ pre_votes[r][c][v]=0 ? act : history_pre_votes[r][c][v];
}
/* for each round records the first party to have a pre-vote */
history_pre : array ROUNDS of PROC;
forall (r in ROUNDS) {
  init(history_pre[r]) := act;
  next(history_pre[r]) := next(pre[r])>0 ∧ pre[r]=0 ? act : history_pre[r];
}
/* records the first party to have a pre-processing-vote for each value */
history_pre_proc_votes : array 0..1 of PROC;
for(v = 0; v ≤ 1; v = v + 1) {
  init(history_pre_proc_votes[v]) := act;
  next(history_pre_proc_votes[v]) := next(pre_proc_votes[v])>0 ∧ pre_proc_votes[v]=0 ? act : history_pre_proc_votes[v];
}
/* records the party which flips the coin */
history_coin : array ROUNDS of PROC;
forall (r in ROUNDS) {
  init(history_coin[r]) := act;
  next(history_coin[r]) := !next(coin[r])=not_tossed ∧ coin[r]=not_tossed ? act : history_coin[r];
}

}
```