```
/* BYZANTINE AGREEMENT PROTOCOL */
/* this files contains the proof of agreement */

/*————————————ASSUMPTIONS————————————————-*/

/* 1-4: there cannot be M votes for v and M votes for v' */
/* these follows from the fact if this was not true there would be */
/* M + M = (N − 2T) + (N − 2T) = N − T + (N − 3T) > N − T honest parties */
/* which is a contradiction since there are only N-T honest parties */
forall (r in ROUNDS) {
        assumption1[r] : assert G ¬( main_votes[r][2]=M ∧ main_votes[r][0]=M);
        assumption2[r] : assert G ¬( main_votes[r][2]=M ∧ main_votes[r][1]=M);
        assumption3[r] : assert G ¬( main_votes[r][0]=M ∧ main_votes[r][1]=M);
        assume assumption1[r], assumption2[r], assumption3[r];
}
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) {
        assumption4[r][c] : assert G ¬( pre_votes[r][c][0]=M ∧ pre_votes[r][c][1]=M);
        assume assumption4[r][c];
}


/*—————————ADDITIONAL INVARIANTS———————————————*/

/* these are proved in a separate file (lemmas.smv) */

forall (r in ROUNDS) forall (n in VOTES) {
        lemma5[r][n] : assert G ( pre[r]=n ⇒ G (pre[r]≥n) );
        assume lemma5[r][n];
}
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) forall (n in VOTES) {
        lemma9[r][c][v][n] : assert G ( pre_votes[r][c][v]≥n ⇒ G (pre_votes[r][c][v]≥n) );
        assume lemma9[r][c][v][n];
}
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) {
        lemma11[r][c][v] : assert G (pre_votes[r][c][v]>0 ⇒ G (pre_votes[r][c][v]>0) );
        assume lemma11[r][c][v];
}
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) {
        lemma16[r][c] : assert G ( pre_votes[r][c][1]=0 ⇒ pre_votes[r][c][0]=pre[r] );
        assume lemma16[r][c];
}
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) {
        lemma17[r][c] : assert G ( pre_votes[r][c][0]=0 ⇒ pre_votes[r][c][1]=pre[r] );
        assume lemma17[r][c];
}
forall (r in ROUNDS) forall (n in VOTES) {
        lemma19[r][n] : assert G ( main[r]≥n ⇒ G (main[r]≥n) );
        assume lemma19[r][n];
}
forall (r in ROUNDS) for(v = 0; v ≤ 2; v = v + 1) forall (n in VOTES) {
        lemma23[r][v][n] : assert G ( main_votes[r][v]≥n ⇒ G (main_votes[r][v]≥n) );
        assume lemma23[r][v][n];
}
forall (r in ROUNDS) for(v = 0; v ≤ 2; v = v + 1) {
        lemma25[r][v] : assert G ( main_votes[r][v]>0 ⇒ G (main_votes[r][v]>0) );
        assume lemma25[r][v];
}
forall (r in ROUNDS) {
        lemma31[r] : assert G ( (main_votes[r][1]=0 ∧ main_votes[r][2]=0) ⇒ main_votes[r][0]=main[r] );
```

```
        assume lemma31[r];
}
forall (r in ROUNDS) {
        lemma32[r] : assert G ( (main_votes[r][0]=0 ∧ main_votes[r][2]=0) ⇒ main_votes[r][1]=main[r] );
        assume lemma32[r];
}
forall (r in ROUNDS) {
        lemma33[r] : assert G ( (main_votes[r][0]=0 ∧ main_votes[r][1]=0) ⇒ main_votes[r][2]=main[r] );
        assume lemma33[r];
}
forall (r in ROUNDS) forall (i in PROC) {
        lemma37[r][i] : assert G ( decide[r][i] ⇒ G (decide[r][i]) );
        assume lemma37[r][i];
}
forall (r in ROUNDS) for (v = 0; v ≤ 2; v = v + 1) {
        corrupted1[r][v] : assert G ( corrupted_main[r][v] ⇒ G (corrupted_main[r][v]) );
        assume corrupted1[r][v];
}
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) for (v = 0; v ≤ 1; v = v + 1) {
        corrupted2[r][c][v] : assert G ( corrupted_pre[r][c][v] ⇒ G (corrupted_pre[r][c][v]) );
        assume corrupted2[r][c][v];
}
forall (r in ROUNDS) for (v = 0; v ≤ 2; v = v + 1) {
        corrupted5[r][v] : assert G ( main_votes[r][v]>0 ⇒ corrupted_main[r][v] );
        assume corrupted5[r][v];
}
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) for (v = 0; v ≤ 1; v = v + 1) {
        corrupted6[r][c][v] : assert G ( pre_votes[r][c][v]>0 ⇒ corrupted_pre[r][c][v] );
        assume corrupted6[r][c][v];
}
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) {
        coin2[r][c] : assert G ( coin[r]=c ⇒ G ( coin[r]=c ) );
                assume coin2[r][c];
}
forall (r in ROUNDS) {
        lemma40[r] : assert G ( (main_votes[r][0]>0 ∨ corrupted_main[r][0]) ⇒ ( main_votes[r][1]=0 ∧ ¬corrupted_main[r][1] ) );
        assume lemma40[r];
}
forall (r in ROUNDS) {
        lemma41[r] : assert G ( (main_votes[r][1]>0 ∨ corrupted_main[r][1]) ⇒ ( main_votes[r][0]=0 ∧ ¬corrupted_main[r][0] ) );
        assume lemma41[r];
}

/*————————SUBLEMMAS FOR AGREEMENT————————————————-*/

/* if a party decides on v then there are no main votes for !v and less than M for abstain */
forall (i in PROC) forall (r in ROUNDS) for(v = 0; v ≤ 1; v = v + 1) {
        inv1[i][r][v] : assert G ( (decide[r][i] ∧ decide_value[r][i]=v) ⇒
            (main_votes[r][v]=M ∧ main_votes[r][v+1 mod 2]=0 ∧ ¬corrupted_main[r][v+1 mod 2] ∧ main_votes[r][2]<M) );
        using lemma40[r],
                lemma41[r],
                lemma23[r][v][M],
                corrupted1[r],
                /* assumptions */
                assumption1[r],
                assumption2[r],
                /* abstractions */
                VOTES⇒{0,M},
                /* free variables */
                corrupted_main//free,
```

```
                corrupted_main[r][v+1 mod 2],
                corrupted_pre//free,
                main//free,
                main_votes//free,
                main_votes[r][v],
                pre//free,
                pre_votes//free,
                pre_proc//free,
                pre_proc_votes//free
                prove inv1[i][r][v];
}
/* if a party decides on v then no party has a pre_vote for !v in the next round */
forall (i in PROC) forall (r in ROUNDS) for(v = 0; v ≤ 1; v = v + 1) for (c = 0; c ≤ 1; c = c + 1){
        inv2[i][r][v][c] : assert G ( (decide[r][i] ∧ decide_value[r][i]=v) ⇒
            ( pre_votes[r+1][c][v+1 mod 2]=0 ∧ ¬corrupted_pre[r+1][c][v+1 mod 2] ) );
        forall (j in PROC) {
                subcase inv2[i][r][v][c][j] of inv2[i][r][v][c] for j=history_pre_votes[r+1][c][v+1 mod 2];
                using inv1[i][r][v],
                  lemma19[r][M],
                  lemma23[r][0][M],
                  lemma23[r][1][M],
                  lemma23[r][2][M],
                  lemma25[r][v+1 mod 2],
                  lemma25[r][2],
                  lemma31[r],
                  lemma32[r],
                  lemma33[r],
                  corrupted1[r],
                  /* abstractions */
                  VOTES⇒{0,M},
                  ROUNDS⇒{0,r,r+1},
                  /* free variables */
                  coin//free,
                  corrupted_main//free,
                  corrupted_pre//free,
                  corrupted_pre[r+1][c][v+1 mod 2],
                  decide//free,
                  decide[r][i],
                  main//free,
                  main_votes//free,
                  pc//free,
                  pc[j],
                  pre//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free,
                  pre_votes[r+1][c][v+1 mod 2],
                  round//free,
                  round[j]
                  prove inv2[i][r][v][c][j];
        }
}
/* if a party decides on 0, then there are no main_votes for 1 in the next round */
forall (i in PROC) forall (r in ROUNDS) for(v = 0; v ≤ 1; v = v + 1) {
        inv3[i][r][v] : assert G ( (decide[r][i] ∧ decide_value[r][i]=v) ⇒
            ( main_votes[r+1][v+1 mod 2]=0 ∧ ¬corrupted_main[r+1][v+1 mod 2] ) );
        forall (j in PROC) {
                subcase inv3[i][r][v][j] of inv3[i][r][v] for j=history_main_votes[r+1][v+1 mod 2];
                using (inv3[i][r][v][j]),
                  inv2[i][r][v],
```

```
                lemma9[r+1][0][0][M],
                lemma9[r+1][0][1][M],
                lemma9[r+1][1][0][M],
                lemma9[r+1][1][1][M],
                lemma25[r+1][v+1 mod 2],
                /* abstractions */
                VOTES⇒{0,M},
                ROUNDS⇒{r,r+1},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_main[r+1][v+1 mod 2],
                corrupted_pre//free,
                decide//free,
                decide[r][i],
                main//free,
                main_votes//free,
                main_votes[r+1][v+1 mod 2],
                pc//free,
                pc[j],
                pre//free,
                pre[r+1],
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove inv3[i][r][v][j];
        }
}
/* if a party decides on 0, then there are no main_votes for abstain in the next round */
forall (i in PROC) forall (r in ROUNDS) for(v = 0; v ≤ 1; v = v + 1) {
        inv4[i][r][v] : assert G ( (decide[r][i] ∧ decide_value[r][i]=v) ⇒ ( main_votes[r+1][2]=0 ∧ ¬corrupted_main[r+1][2] ) );
        forall (j in PROC) {
                subcase inv4[i][r][v][j] of inv4[i][r][v] for j=history_main_votes[r+1][2];
                using (inv4[i][r][v][j]),
                  inv2[i][r][v],
                  lemma9[r+1][0][0][M],
                  lemma9[r+1][1][0][M],
                  lemma9[r+1][0][1][M],
                  lemma9[r+1][1][1][M],
                  lemma11[r+1],
                  lemma16[r+1],
                  lemma17[r+1],
                  lemma25[r+1][2],
                  corrupted2[r+1],
                  corrupted6[r+1],
                  /* abstractions */
                  VOTES⇒{0,M},
                  ROUNDS⇒{r,r+1},
                  /* free variables */
                  coin//free,
                  corrupted_main//free,
                  corrupted_main[r+1][2],
                  corrupted_pre//free,
                  decide//free,
                  main//free,
                  main_votes//free,
                  main_votes[r+1][2],
                  pc//free,
                  pc[j],
```

```
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                round//free,
                round[j]
                prove inv4[i][r][v][j];
        }
}
/*————————-AGREEMENT PROOF———————————————*/


/* if two parties decide in the same round they decide on the same value */
forall (i in PROC) forall (r in ROUNDS) forall (j in PROC) {
        agree1[i][r][j] : assert G ( (decide[r][i] ∧ decide[r][j]) ⇒ decide_value[r][j]=decide_value[r][i] );
        using inv1[i][r],
                lemma25[r],
                corrupted1[r],
                /* abstractions */
                VOTES⇒{0,M},
                /* free variables */
                coin//free,
                corrupted_pre//free,
                corrupted_main//free,
                main//free,
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove agree1[i][r][j];
}
/* if a party decides in round r and other party decides in round r+1, then they must decide on the same value */
forall (i in PROC) forall (r in ROUNDS) forall (j in PROC) {
        agree2[i][r][j] : assert G ( (decide[r][i] ∧ decide[r+1][j]) ⇒ decide_value[r+1][j]=decide_value[r][i] );
        using inv3[i][r],
                lemma25[r+1],
                corrupted1[r+1],
                /* abstractions */
                VOTES⇒{0,M},
                ROUNDS⇒{r,r+1},
                /* free variables */
                coin//free,
                corrupted_pre//free,
                corrupted_main//free,
                decide//free,
                decide[r][i],
                decide[r+1][j],
                main//free,
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove agree2[i][r][j];
}
/* if party i decides in round r and is the first party to decide, */
/* then any other party (j) must decide in round r or round r+1 */
```

```
    /* to prove this property we require some assumptions which we cannot prove because we are using ordset */

    /* the premise is that no party has decided before round r */
    /* noting that a party continues for one more round after it decides under fairness we have */
    /* all parties reaches round r+1 */
    /* and any party that does not decide in round r reaches round r+2 */
    /* (note that therefore these assumptions includes fairness requirements) */
    forall (i in PROC) forall (r in ROUNDS) {
        agreement_assumption1[i][r] : assert F ( round[i]=r+1 );
        agreement_assumption2[i][r] : assert G ( ¬decide[r][i] ) ⇒ F ( round[i]=r+2 );
        assume agreement_assumption1[i][r], agreement_assumption2[i][r];
    }
    /* the proof is then given by */
    forall (i in PROC) forall (r in ROUNDS) forall (j in PROC) {
        agree3[i][r][j] : assert G ( (decide[r][i]) ⇒ F (decide[r][j] ∨ decide[r+1][j]) );
        using inv1[i][r],
              inv3[i][r],
              inv4[i][r],
              lemma5[r+1][M],
              lemma19[r][M],
              lemma19[r+1][M],
              lemma23[r][0][M],
              lemma23[r+1][0][M],
              lemma23[r][1][M],
              lemma23[r+1][1][M],
              lemma25[r],
              lemma25[r+1],
              lemma31[r+1],
              lemma32[r+1],
              lemma37[r],
              lemma37[r+1],
              corrupted1[r+1],
              corrupted5[r+1],
              coin2[r],
              coin2[r+1],
              /* assumed properties */
              agreement_assumption1[j][r],
              agreement_assumption2[j][r],
              /* abstractions */
              VOTES⇒{0,M},
              ROUNDS⇒{r-1,r,r+1},
              /* free variables */
              coin//free,
              corrupted_pre//free,
              corrupted_main//free,
              decide//free,
              decide[r][j],
              decide[r][i],
              decide[r+1][j],
              main//free,
              main_votes//free,
              pre//free,
              pre_proc//free,
              pre_proc_votes//free,
              pre_votes//free,
              round//free,
              round[j]
              prove agree3[i][r][j];
    }
}
```