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

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

/* VALIDITY ASSUMPTION: all parties start with the same value */
  for(v = 0; v ≤ 1; v = v + 1) forall (i in PROC){
        validity_assumption1[v][i] : assert G (start[i]=v);
        assume validity_assumption1[v][i];
}
/* also since all parties continue for one more round after they decide */
/* under fairness all parties enter round 1 */
forall (i in PROC) {
        validity_assumption2[i] : assert F (round[i]>0);
        assume validity_assumption2[i];
}

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

lemma3 : assert G ( pre_proc>0 ⇒ (pre_proc_votes[0]>0 ∨ pre_proc_votes[1]>0) );
assume lemma3;
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) {
        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) {
        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) for (v = 0; v ≤ 2; v = v + 1) {
        corrupted5[r][v] : assert G ( main_votes[r][v]>0 ⇒ corrupted_main[r][v]=1 );
        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]=1 );
        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];
}

/*—————————SUBLEMMAS FOR VALIDITY————————————-*/
```

```
/* show that if no party starts with an initial value then there are no pre processing vote for it */
/* we use the history variable for this */
/* no party starts with !v then there are no pre processing votes for !v */
for (v = 0; v ≤ 1; v = v + 1) {
        inv1[v] : assert G ( pre_proc_votes[v+1 mod 2]=0 );
        forall (i in PROC) {
        subcase inv1[v][i] of inv1[v] for i=history_pre_proc_votes[v+1 mod 2];
        using (inv1[v]),
                /* assumptions */
                validity_assumption1[v],
                /* abstractions */
                PRE_PROC_VOTES⇒{0}
                prove inv1[v][i];
        }
}
/* no preprocessing votes for !v means no pre votes for !v */
for (v = 0; v ≤ 1; v = v + 1) for (c = 0; c ≤ 1; c = c + 1) {
        inv2[v][c] : assert G ( pre_votes[0][c][v+1 mod 2]=0 ∧ ¬corrupted_pre[0][c][v+1 mod 2] );
        forall (i in PROC) {
                subcase inv2[v][c][i] of inv2[v][c] for i=history_pre_votes[0][c][v+1 mod 2];
                using inv1[v],
                  lemma3,
                  /* abstractions */
                  ROUNDS⇒{0},
                  VOTES⇒{0},
                  PRE_PROC_VOTES⇒{0,K},
                  /* free variables */
                  coin//free,
                  corrupted_main//free,
                  corrupted_pre//free,
                  corrupted_pre[0][c][v+1 mod 2],
                  decide//free,
                  main//free,
                  main_votes//free,
                  pre//free,
                  pre_votes//free,
                  pre_votes[0][c][v+1 mod 2],
                  start//free
                  prove inv2[v][c][i];
        }
}
/* no pre votes for !v means no main votes for !v or for abstain */
for (v = 0; v ≤ 1; v = v + 1) {
        inv3[v] : assert
          G ( main_votes[0][v+1 mod 2]=0 ∧ main_votes[0][2]=0 ∧ ¬corrupted_main[0][v+1 mod 2] ∧ ¬corrupted_main[0][2] );
        forall (i in PROC) forall (j in PROC) {
                subcase inv3[v][i][j] of inv3[v] for i=history_main_votes[0][v+1 mod 2] ∧ j=history_main_votes[0][2];
                using inv2[v],
                  lemma16[0],
                  lemma17[0],
                  corrupted6[0],
                  /* abstractions */
                  ROUNDS⇒{0},
                  VOTES⇒{0,M},
                  /* free variables */
                  corrupted_main//free,
                  corrupted_main[0][v+1 mod 2],
                  corrupted_main[0][2],
                  decide//free,
```

```
                    main_votes//free,
                    main_votes[0][v+1 mod 2],
                    main_votes[0][2],
                    pre//free,
                    pre_proc//free,
                    pre_proc_votes//free,
                    pre_votes//free
                    prove inv3[v][i][j];
            }
    }

    /*———————-VALIDITY PROOF———————*/

    /* now using these asumptions we can prove validity */
    for (v = 0; v ≤ 1; v = v + 1) forall (i in PROC) {
            valid[v][i] : assert F ( decide[0][i] ∧ decide_value[0][i]=v );
            using inv3[v],
                    lemma5[0][M],
                    lemma19[0][M],
                    lemma31[0],
                    lemma32[0],
                    corrupted5[0],
                    coin2[0],
                    /* assumptions */
                    validity_assumption2[i],
                    /* abstractions */
                    ROUNDS⇒{0},
                    VOTES⇒{0,M},
                    PRE_PROC_VOTES⇒{0,K}
                    prove valid[v][i];
    }

    /*————————-END OF PROOF————————-*/
}
```