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

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

/* there cannot be M votes for v and M votes for v' */
/* this 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) 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————————————————*/

/* PROPERTIES CONCERNING THE PREPROCESSING STEP */
/* total number of pre processing votes for a value not decreasing in one step */
for(v = 0; v ≤ 1; v = v + 1) {
        lemma1[v] : assert G ( pre_proc_votes[v]>0 ⇒ X (pre_proc_votes[v]>0) );
        forall (i in PROC) {
                subcase lemma1[v][i] of lemma1[v] for act=i;
                using
                  /* abstractions */
                  PRE_PROC_VOTES⇒{0,K},
                  /* free variables */
                  pc//free,
                  pre_proc_votes//free,
                  pre_proc_votes[v],
                  start//free
                  prove lemma1[v][i];
        }
}
/* total number of pre processing votes for a value not decreasing (need one step first) */
for(v = 0; v ≤ 1; v = v + 1) {
        lemma2[v] : assert G ( pre_proc_votes[v]>0 ⇒ G (pre_proc_votes[v]>0) );
        using
                (lemma2[v]), /* assume lemma2 at time t-1 */
                lemma1[v],
                /* abstractions */
                PRE_PROC_VOTES⇒{0,K},
                /* free variables */
                pre_proc_votes//free,
                pre_proc_votes[v]
                prove lemma2[v];
}
/* if pre_processing>0 then pre_proc_votes[v]>0 for some v */
/* to do this we need to know the first party that is scheduled */
/* we therefore introduce the following variable */
init_act : PROC;
init(init_act) := act;
next(init_act) := init_act;
/* using this we can prove the property as follows */
lemma3 : assert G ( pre_proc>0 ⇒ (pre_proc_votes[0]>0 ∨ pre_proc_votes[1]>0) );
forall(i in PROC) {
        subcase lemma3[i] of lemma3 for init_act=i;
        using
                lemma1[0],
                lemma1[1],
```

```
                /* abstractions */
                PRE_PROC_VOTES⇒{0},
                /* free variables */
                round//free,
                start//free
                prove lemma3[i];
}

/* PROPERTIES CONCERNING PRE */
/* total number of pre votes does not decrease in one step */
forall (r in ROUNDS) forall (n in VOTES) {
        lemma4[r][n] : assert G ( pre[r]≥n ⇒ X (pre[r]≥n) );
        forall (i in PROC) {
                subcase lemma4[r][n][i] of lemma4[r][n] for act=i;
                using
                  /* abstractions */
                  VOTES⇒{0,M},
                  ROUNDS⇒{r-1,r},
                  /* free variables */
                  corrupted_main//free,
                  corrupted_pre//free,
                  decide//free,
                  main//free,
                  main_votes//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free,
                  start//free
                  prove lemma4[r][n][i];
        }
}
/* total number of pre votes does not decrease (need one step first) */
forall (r in ROUNDS) forall (n in VOTES) {
        lemma5[r][n] : assert G ( pre[r]=n ⇒ G (pre[r]≥n) );
        using
                (lemma5[r][n]),
                lemma4[r][n],
                /* abstractions */
                VOTES⇒{M},
                /* free variables */
                main//free,
                pre_proc//free
                prove lemma5[r][n];
}
/* pre can only increase by at most 1 in one step */
forall (r in ROUNDS) forall (n in VOTES) {
        lemma6[r][n] : assert G ( (pre[r]=n ∧ n<M) ⇒ X (pre[r]≤n+1) );
        forall (i in PROC) {
                subcase lemma6[r][n][i] of lemma6[r][n] for act=i;
                using
                /* abstractions */
                VOTES⇒{n,n+1},
                ROUNDS⇒{r-1,r},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                decide//free,
                main//free,
                main_votes//free,
```

```
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove lemma6[r][n][i];
        }
}
/* dual of this property */
forall (r in ROUNDS) forall (n in VOTES) {
        lemma7[r][n] : assert G ( X(pre[r]=n ∧ n>0) ⇒ pre[r]≥n-1 );
        forall (i in PROC) {
                subcase lemma7[r][n][i] of lemma7[r][n] for act=i;
                using
                /* abstractions */
                VOTES⇒{n-1,n},
                ROUNDS⇒{r-1,r},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                decide//free,
                main//free,
                main_votes//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove lemma7[r][n];
        }
}
/* PROPERTIES CONCERNING PRE_VOTES */
/* total number of pre votes for a value does not decrease in one step */
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) forall (n in VOTES) {
        lemma8[r][c][v][n] : assert G ( pre_votes[r][c][v]≥n ⇒ X (pre_votes[r][c][v]≥n) );
        forall (i in PROC) {
                subcase lemma8[r][c][v][n][i] of lemma8[r][c][v][n] for act=i;
                using
                  /* abstractions */
                  VOTES⇒{M},
                  ROUNDS⇒{r-1,r},
                  /* free variables */
                  corrupted_main//free,
                  corrupted_pre//free,
                  main//free,
                  main_votes//free,
                  pc//free,
                  pre//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free,
                  pre_votes[r][c][v]
                  prove lemma8[r][c][v][n][i];
        }
}
/* total number of pre votes for a value does not decrease (need one step first) */
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) );
        using
                (lemma9[r][c][v][n]),
                lemma8[r][c][v][n],
```

```
            /* abstractions */
            VOTES⇒{M},
            /* free variables */
            corrupted_main//free,
            corrupted_pre//free,
            main//free,
            main_votes//free,
            pre//free,
            pre_proc_votes//free,
            pre_votes//free,
            pre_votes[r][c][v]
            prove lemma9[r][c][v][n];
}
/* variant on the total number of pre votes for a value does not decreasing in one step */
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) {
    lemma10[r][c][v] : assert G ( pre_votes[r][c][v]>0 ⇒ X (pre_votes[r][c][v]>0) );
    forall (i in PROC) {
        subcase lemma10[r][c][v][i] of lemma10[r][c][v] for act=i;
        using
          /* abstractions */
          VOTES⇒{0,M},
          ROUNDS⇒{r-1,r},
          /* free variables */
          corrupted_main//free,
          corrupted_pre//free,
          main//free,
          main_votes//free,
          pc//free,
          pre//free,
          pre_proc//free,
          pre_proc_votes//free,
          pre_votes//free,
          pre_votes[r][c][v],
          round//free,
          start//free
          prove lemma10[r][c][v][i];
    }
}
/* variant on the total number of pre votes for a value not decreasing (need one step first) */
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) );
    using
        (lemma11[r][c][v]),
        lemma10[r][c][v],
        VOTES⇒{0},
        coin//free,
        corrupted_main//free,
        corrupted_pre//free,
        main//free,
        main_votes//free,
        pc//free,
        pre//free,
        pre_proc//free,
        pre_proc_votes//free,
        pre_votes//free,
        pre_votes[r][c][v]
        prove lemma11[r][c][v];
}
/* pre_votes can increase by at most 1 in one step */
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) forall (n in VOTES) {
```

```
        lemma12[r][c][v][n] : assert G ( (pre_votes[r][c][v]=n ∧ n<M) ⇒ X (pre_votes[r][c][v]≤n+1) );
        forall (i in PROC) {
                subcase lemma12[r][c][v][n][i] of lemma12[r][c][v][n] for act=i;
                using
                /* abstractions */
                VOTES⇒{n,n+1},
                ROUNDS⇒{r-1,r},
                /* free variables */
                corrupted_main//free,
                decide//free,
                main//free,
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                pre_votes[r][c][v],
                start//free
                prove lemma12[r][c][v][n][i];
        }
}
/* dual of above property */
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) for(v = 0; v ≤ 1; v = v + 1) forall (n in VOTES) {
        lemma13[r][c][v][n] : assert G ( X(pre_votes[r][c][v]=n ∧ n>0) ⇒ pre_votes[r][c][v]≥n-1 );
        forall (i in PROC) {
                subcase lemma13[r][c][v][n][i] of lemma13[r][c][v][n] for act=i;
                using
                /* abstractions */
                VOTES⇒{n-1,n},
                ROUNDS⇒{r-1,r},
                /* free variables */
                corrupted_main//free,
                decide//free,
                main//free,
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                pre_votes[r][c][v],
                start//free
                prove lemma13[r][c][v][n][i];
        }
}


/* PROPERTIES RELATING PRE AND PRE_VOTES */
/* if one of pre_votes[r][c][0] or pre_votes[r][c][1] is zero then the other equals pre[r] */
/* first we need a one step argument */
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) {
        lemma14[r][c] : assert G ( (pre_votes[r][c][1]=0 ⇒ pre_votes[r][c][0]=pre[r])
                        ⇒ X ( pre_votes[r][c][1]=0 ⇒ pre_votes[r][c][0]=pre[r]) );
        forall (n in VOTES) forall (i in PROC) {
                subcase lemma14[r][c][n][i] of lemma14[r][c] for pre_votes[r][c][0]=n ∧ act=i;
                  using
                  /* abstractions */
                  VOTES⇒{0,n,n+1,M},
                  ROUNDS⇒{r-1,r},
                  PRE_PROC_VOTES⇒{0,K},
                  /* free variables */
```

```
                corrupted_main//free,
                corrupted_pre//free,
                decide//free,
                main//free,
                main_votes//free,
                pre//free,
                pre[r],
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                pre_votes[r][c]
                prove lemma14[r][c][n][i];
        }
}
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) {
        lemma15[r][c] : assert G ( (pre_votes[r][c][0]=0 ⇒ pre_votes[r][c][1]=pre[r])
                        ⇒ X ( pre_votes[r][c][0]=0 ⇒ pre_votes[r][c][1]=pre[r]) );
        forall (n in VOTES) forall (i in PROC) {
                subcase lemma15[r][c][n][i] of lemma15[r][c] for pre_votes[r][c][1]=n ∧ act=i;
                  using
                  /* abstractions */
                  VOTES⇒{0,n,n+1,M},
                  ROUNDS⇒{r-1,r},
                  PRE_PROC_VOTES⇒{0,K},
                  /* free variables */
                  corrupted_main//free,
                  corrupted_pre//free,
                  decide//free,
                  main//free,
                  main_votes//free,
                  pre//free,
                  pre[r],
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free,
                  pre_votes[r][c]
                  prove lemma15[r][c][n][i];
        }
}
/* using the one step arguments we get what we want */
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] );
        forall (n in VOTES) forall (i in PROC) {
                subcase lemma16[r][c][n] of lemma16[r][c] for pre_votes[r][c][0]=n;
                using (lemma16[r][c]),
                  lemma5[r],
                  lemma6[r],
                  lemma7[r],
                  lemma9[r][c][0],
                  lemma12[r][c][0],
                  lemma13[r][c][0],
                  lemma14[r][c],
                  /* abstractions */
                  VOTES⇒{0,n-1,n,n+1,M},
                  /* free variables */
                  corrupted_main//free,
                  corrupted_pre//free,
                  main//free,
                  main_votes//free,
                  pre_proc//free,
```

```
                pre_proc_votes//free,
                pre_votes//free,
                pre_votes[r][c]
                prove lemma16[r][c][n];
        }
}
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] );
        forall (n in VOTES) forall (i in PROC) {
                subcase lemma17[r][c][n] of lemma17[r][c] for pre_votes[r][c][1]=n;
                using (lemma17[r][c]),
                  lemma5[r],
                  lemma6[r],
                  lemma7[r],
                  lemma9[r][c][1],
                  lemma12[r][c][1],
                  lemma13[r][c][1],
                  lemma15[r][c],
                  /* abstractions */
                  VOTES⇒{0,n-1,n,n+1,M},
                  /* free variables */
                  corrupted_main//free,
                  corrupted_pre//free,
                  main//free,
                  main_votes//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free,
                  pre_votes[r][c]
                  prove lemma17[r][c][n];
        }
}

/* PROPERTIES CONCERNING MAIN */
/* total number of main votes does not decrease in one step */
forall (r in ROUNDS) forall (n in VOTES) {
        lemma18[r][n] : assert G ( main[r]≥n ⇒ X (main[r]≥n) );
        forall (i in PROC) {
                subcase lemma18[r][n][i] of lemma18[r][n] for act=i;
                using
                  /* abstractions */
                  VOTES⇒{M},
                  /* free variables */
                  corrupted_main//free,
                  corrupted_pre//free,
                  decide//free,
                  main_votes//free,
                  pre//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  round//free,
                  start//free
                  prove lemma18[r][n][i];
        }
}
/* total number of main votes does not decrease (need one step first) */
forall (r in ROUNDS) forall (n in VOTES) {
        lemma19[r][n] : assert G ( main[r]≥n ⇒ G (main[r]≥n) );
        using
                (lemma19[r][n]),
```

```
                lemma18[r][n],
                VOTES⇒{M},
                /* free variables */
                pre//free,
                pre_proc//free
                prove lemma19[r][n];
}
/* main can increase by at most 1 in one step */
forall (r in ROUNDS) forall (n in VOTES) {
        lemma20[r][n] : assert G ( (main[r]=n ∧ n<M) ⇒ X (main[r]≤n+1) );
        forall (i in PROC) {
                subcase lemma20[r][n][i] of lemma20[r][n] for act=i;
                using
                /* abstractions */
                VOTES⇒{n,n+1},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                decide//free,
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                round//free,
                start//free
                prove lemma20[r][n][i];
        }
}
/* dual of above property */
forall (r in ROUNDS) forall (n in VOTES) {
        lemma21[r][n] : assert G ( X(main[r]=n ∧ n>0) ⇒ main[r]≥n-1 );
        forall (i in PROC) {
                subcase lemma21[r][n][i] of lemma21[r][n] for act=i;
                using
                /* abstractions */
                VOTES⇒{n-1,n},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                decide//free,
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                round//free,
                start//free
                prove lemma21[r][n];
        }
}

/* PROPERTIES CONCERNING MAIN_TOTAL */
/* total number of main votes for a value does not decrease in one step */
forall (r in ROUNDS) for(v = 0; v ≤ 2; v = v + 1) forall (n in VOTES) {
        lemma22[r][v][n] : assert G ( main_votes[r][v]≥n ⇒ X (main_votes[r][v]≥n) );
        forall (i in PROC) {
                subcase lemma22[r][v][n][i] of lemma22[r][v][n] for act=i;
```

```
                using
                  /* abstractions */
                  VOTES⇒{M},
                  /* free variables */
                  coin//free,
                  corrupted_main//free,
                  corrupted_pre//free,
                  main//free,
                  main_votes//free,
                  main_votes[r][v],
                  pc//free,
                  pre//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free,
                  round//free,
                  start//free
                  prove lemma22[r][v][n][i];
        }
}
/* total number of main votes for a value does not decrease (need one step first) */
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) );
        using
                (lemma23[r][v][n]),
                VOTES⇒{M},
                lemma22[r][v][n],
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                main//free,
                main_votes//free,
                main_votes[r][v],
                pc//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                round//free,
                start//free
                prove lemma23[r][v][n];
}
/* variant on the total number of main votes for a value not decreasing in one step */
forall (r in ROUNDS) for(v = 0; v ≤ 2; v = v + 1) {
        lemma24[r][v] : assert G ( main_votes[r][v]>0 ⇒ X (main_votes[r][v]>0) );
        forall (i in PROC) {
                subcase lemma24[r][v][i] of lemma24[r][v] for act=i;
                using
                  /* abstractions */
                  VOTES⇒{0,M},
                  /* free variables */
                  coin//free,
                  corrupted_main//free,
                  corrupted_pre//free,
                  main//free,
                  main_votes//free,
                  main_votes[r][v],
                  pc//free,
                  pre//free,
```

```
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                round//free,
                start//free
                prove lemma24[r][v][i];
        }
}
/* variant on the total number of main votes for a value not decreasing (need one step first) */
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) );
        using
                (lemma25[r][v]),
                lemma24[r][v],
                VOTES⇒{0},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                main//free,
                main_votes//free,
                main_votes[r][v],
                pc//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                round//free,
                start//free
                prove lemma25[r][v];
}
/* main_votes can increase by at most 1 in one step */
forall (r in ROUNDS) for(v = 0; v ≤ 2; v = v + 1) forall (n in VOTES) {
        lemma26[r][v][n] : assert G ( (main_votes[r][v]=n ∧ n<M) ⇒ X (main_votes[r][v]≤n+1) );
        forall (i in PROC) {
                subcase lemma26[r][v][n][i] of lemma26[r][v][n] for act=i;
                using
                /* abstractions */
                VOTES⇒{n,n+1},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                main//free,
                main_votes//free,
                main_votes[r][v],
                pc//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                round//free,
                start//free
                prove lemma26[r][v][n][i];
        }
}
/* dual of above property */
forall (r in ROUNDS) for(v = 0; v ≤ 2; v = v + 1) forall (n in VOTES) {
        lemma27[r][v][n] : assert G ( X(main_votes[r][v]=n ∧ n>0) ⇒ main_votes[r][v]≥n-1 );
        forall (i in PROC) {
```

```
            subcase lemma27[r][v][n][i] of lemma27[r][v][n] for act=i;
            using
            /* abstractions */
            VOTES⇒{n-1,n},
            /* free variables */
            coin//free,
            corrupted_main//free,
            corrupted_pre//free,
            main//free,
            main_votes//free,
            main_votes[r][v],
            pc//free,
            pre//free,
            pre_proc//free,
            pre_proc_votes//free,
            pre_votes//free,
            round//free,
            start//free
            prove lemma27[r][v][n][i];
        }
}

/* PROPERTIES RELATING MAIN AND MAIN_TOTAL */
/* if two of main_votes[r][v] for v=0,1,2 are zero then the remaining one equals main[r] */
/* first we need a one step argument */
forall (r in ROUNDS) {
        lemma28[r] : assert G ( ((main_votes[r][1]=0 ∧ main_votes[r][2]=0) ⇒ main_votes[r][0]=main[r])
                                ⇒ X ( (main_votes[r][1]=0 ∧ main_votes[r][2]=0) ⇒ main_votes[r][0]=main[r]) );
        forall (n in VOTES) forall (i in PROC) {
            subcase lemma28[r][n][i] of lemma28[r] for main_votes[r][0]=n ∧ act=i;
              using lemma36[i][r],
              /* abstractions */
              VOTES⇒{0,n,n+1,M},
              /* free variables */
              coin//free,
              corrupted_main//free,
              corrupted_pre//free,
              decide//free,
              pre//free,
              pre_proc//free,
              pre_proc_votes//free,
              pre_votes//free,
              start//free
              prove lemma28[r][n][i];
        }
}
forall (r in ROUNDS) {
        lemma29[r] : assert G ( ((main_votes[r][0]=0 ∧ main_votes[r][2]=0) ⇒ main_votes[r][1]=main[r])
                                ⇒ X ( (main_votes[r][0]=0 ∧ main_votes[r][2]=0) ⇒ main_votes[r][1]=main[r]) );
        forall (n in VOTES) forall (i in PROC) {
            subcase lemma29[r][n][i] of lemma29[r] for main_votes[r][1]=n ∧ act=i;
              using lemma36[i][r],
              /* abstractions */
              VOTES⇒{0,n,n+1,M},
              /* free variables */
              coin//free,
              corrupted_main//free,
              corrupted_pre//free,
              decide//free,
              pre//free,
```

```
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove lemma29[r][n][i];
        }
}
forall (r in ROUNDS) {
        lemma30[r] : assert G ( ((main_votes[r][0]=0 ∧ main_votes[r][1]=0) ⇒ main_votes[r][2]=main[r])
                                ⇒ X ( (main_votes[r][0]=0 ∧ main_votes[r][1]=0) ⇒ main_votes[r][2]=main[r]) );
        forall (n in VOTES) forall (i in PROC) {
                subcase lemma30[r][n][i] of lemma30[r] for main_votes[r][2]=n ∧ act=i;
                using lemma36[i][r],
                /* abstractions */
                VOTES⇒{0,n,n+1,M},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                decide//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free
                prove lemma30[r][n][i];
        }
}
/* using the one step arguments we get what we want */
forall (r in ROUNDS) {
        lemma31[r] : assert G ( (main_votes[r][1]=0 ∧ main_votes[r][2]=0) ⇒ main_votes[r][0]=main[r] );
        forall (n in VOTES) forall (i in PROC) {
                subcase lemma31[r][n] of lemma31[r] for main_votes[r][0]=n;
                using (lemma31[r]),
                lemma18[r],
                lemma20[r],
                lemma21[r],
                lemma22[r][0],
                lemma26[r][0],
                lemma27[r][0],
                lemma28[r],
                /* abstractions */
                VOTES⇒{0,n-1,n,n+1,M},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove lemma31[r][n];
        }
}
forall (r in ROUNDS) {
        lemma32[r] : assert G ( (main_votes[r][0]=0 ∧ main_votes[r][2]=0) ⇒ main_votes[r][1]=main[r] );
        forall (n in VOTES) forall (i in PROC) {
                subcase lemma32[r][n] of lemma32[r] for main_votes[r][1]=n;
                using (lemma32[r]),
                lemma18[r],
```

```
                lemma20[r],
                lemma21[r],
                lemma22[r][1],
                lemma26[r][1],
                lemma27[r][1],
                lemma29[r],
                /* abstractions */
                VOTES⇒{0,n-1,n,n+1,M},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove lemma32[r][n];
        }
}
forall (r in ROUNDS) {
        lemma33[r] : assert G ( (main_votes[r][0]=0 ∧ main_votes[r][1]=0) ⇒ main_votes[r][2]=main[r] );
        forall (n in VOTES) forall (i in PROC) {
                subcase lemma33[r][n] of lemma33[r] for main_votes[r][2]=n;
                using (lemma33[r]),
                lemma18[r],
                lemma20[r],
                lemma21[r],
                lemma22[r][2],
                lemma26[r][2],
                lemma27[r][2],
                lemma30[r],
                /* abstractions */
                VOTES⇒{0,n-1,n,n+1,M},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove lemma33[r][n];
        }
}

/* PROPERTIES CONCERNING DECIDE PHASE */
/* once a party has decided it stays decided */
forall (r in ROUNDS) forall (i in PROC) {
        lemma37[r][i] : assert G ( decide[r][i] ⇒ G (decide[r][i]) );
        using
                /* abstraction */
                VOTES⇒{M},
                ROUNDS⇒{r-1,r},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_pre//free,
                main//free,
```

```
                main_votes//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove lemma37[r][i];
}
/* PROPERTIES OF PARTIES */
/* the party that first increases pre[r] has a round greater than or equal to r */
forall (r in ROUNDS) {
        lemma39[r] : assert G ( pre[r]>0 ⇒ round[history_pre[r]]≥r );
        forall (i in PROC) forall (r1 in ROUNDS) {
                subcase lemma39[r][i][r1] of lemma39[r] for i=history_pre[r] ∧ round[i]=r;
                using (lemma39[r][i]),
                  /* abstractions */
                  VOTES⇒{0},
                  /* free variables */
                  coin//free,
                  corrupted_main//free,
                  corrupted_pre//free,
                  decide//free,
                  main//free,
                  main_votes//free,
                  pre//free,
                  pre[r],
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free,
                  start//free
                  prove lemma39[r][i][r];
        }
}

/* PROPERTIES RELATING MAIN_VOTES AND PRE_VOTES */
/* if there are main votes for 0 then there are no main votes for 1 */
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]));
        forall (i in PROC) forall (j in PROC) {
                subcase lemma40[r][i][j] of lemma40[r] for i=history_main_votes[r][0] ∧ j=history_main_votes[r][1];
                using
                  lemma9[r],
                  coin2[r],
                  /* assumptions */
                  assumption4[r],
                  /* abstractions */
                  VOTES⇒{0,M},
                  /* free variables */
                  coin//free,
                  corrupted_main//free,
                  corrupted_main[r][0],
                  corrupted_main[r][1],
                  corrupted_pre//free,
                  decide//free,
                  main//free,
                  main_votes//free,
                  main_votes[r][0],
                  main_votes[r][1],
                  pc//free,
                  pre//free,
                  pre_proc//free,
```

```
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove lemma40[r][i][j];
        }
}
/* if there are main votes for 1 then there are no main votes for 0 */
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] ) );
        forall (i in PROC) forall (j in PROC) {
                subcase lemma41[r][i][j] of lemma41[r] for i=history_main_votes[r][1] ∧ j=history_main_votes[r][0];
                using
                  lemma9[r],
                  coin2[r],
                  /* assumptions */
                  assumption4[r],
                  /* abstractions */
                  VOTES⇒{0,M},
                  /* free variables */
                  coin//free,
                  corrupted_main//free,
                  corrupted_main[r][0],
                  corrupted_main[r][1],
                  corrupted_pre//free,
                  decide//free,
                  main//free,
                  main_votes//free,
                  main_votes[r][0],
                  main_votes[r][1],
                  pc//free,
                  pre//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free,
                  start//free
                  prove lemma41[r][i][j];
        }
}

/*————————-PROPERTIES OF CORRUPTED PARTIES————————————————*/

/* if an honest party has a vote for a value, then corrupted parties can also vote for this value */
/* first consider the following simple properties */
/* if a corrupted party can make a main vote for v it can always do this */
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]) );
        using corrupted2[r],
                coin2[r],
                lemma5[r][M],
                lemma9[r][0][v][M],
                lemma9[r][1][v][M],
                lemma11[r],
                /* abstractions */
                VOTES⇒{0,M},
                /* free variables */
                corrupted_main//free,
                corrupted_main[r][v],
                corrupted_pre//free,
                main//free,
                main_votes//free,
```

```
                pre//free,
                pre_proc//free,
                pre_proc_votes//free
                prove corrupted1[r][v];

}
/* if a corrupted party can make a pre vote for v it can always do this */
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]) );
        using corrupted1[r-1],
                lemma2[v],
                lemma19[r-1][M],
                lemma23[r-1][1][M],
                lemma23[r-1][2][M],
                lemma23[r-1][3][M],
                lemma25[r-1],
                /* abstractions */
                ROUNDS⇒{0,r-1,r},
                VOTES⇒{0,M},
                PRE_PROC_VOTES⇒{0,K},
                /* free variables */
                corrupted_pre//free,
                corrupted_pre[r][c][v],
                corrupted_main//free,
                corrupted_main[r-1][v],
                main//free,
                main_votes//free,
                pre//free,
                pre_proc_votes//free,
                pre_votes//free
                prove corrupted2[r][c][v];
}

/* using these we have what we want */
/* for main votes */
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] );
        forall (i in PROC) {
                subcase corrupted5[r][v][i] of corrupted5[r][v] for i=history_main_votes[r][v];
                using (corrupted5[r][v][i]),
                corrupted2[r],
                corrupted6[r],
                coin2[r],
                lemma5[r][M],
                lemma9[r][0][0][M],
                lemma9[r][0][1][M],
                lemma9[r][1][0][M],
                lemma9[r][1][1][M],
                lemma11[r],
                lemma16[r],
                lemma17[r],
                /* abstractions */
                VOTES⇒{0,M},
                PRE_PROC_VOTES⇒{0,K},
                /* free variables */
                coin//free,
                corrupted_main//free,
                corrupted_main[r][v],
                corrupted_pre//free,
                decide//free,
```

```
                main//free,
                main_votes//free,
                main_votes[r][v],
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                round//free
                prove corrupted5[r][v][i];
        }
}
/* for pre votes */
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] );
        forall (i in PROC) {
                subcase corrupted6[r][c][v][i] of corrupted6[r][c][v] for i=history_pre_votes[r][c][v];
                using (corrupted6[r][c][v][i]),
                corrupted1[r-1],
                corrupted5[r-1],
                lemma2,
                lemma3,
                lemma19[r-1][M],
                lemma23[r-1][0][M],
                lemma23[r-1][1][M],
                lemma23[r-1][2][M],
                lemma25[r-1],
                lemma31[r-1],
                lemma32[r-1],
                lemma33[r-1],
                /* abstractions */
                ROUNDS⇒{0,r-1,r},
                VOTES⇒{0,M},
                PRE_PROC_VOTES⇒{0,K},
                /* free variables */
                corrupted_main//free,
                corrupted_pre//free,
                corrupted_pre[r][c][v],
                decide//free,
                history_pre_votes//free,
                history_pre_votes[r][c][v],
                main//free,
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                pre_votes[r][c][v]
                prove corrupted6[r][c][v][i];
        }
}

/* ————————PROPERTIES OF THE COINS————————-*/

/* once a coin has been tossed it does not change value on step first */
forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) {
        coin1[r][c] : assert G ( coin[r]=c ⇒ X ( coin[r]=c ) );
        forall (i in PROC) {
                subcase coin1[r][c][i] of coin1[r][c] for act=i;
                using
                  /* abstractions */
```

```
                    ROUNDS⇒{0,r-1},
                    /* free variables */
                    decide//free,
                    main//free,
                    main_votes//free,
                    pre//free,
                    pre[r],
                    pre_proc//free,
                    pre_proc_votes//free,
                    pre_votes//free,
                    start//free
                    prove coin1[r][c][i];
            }
    }
    /* once a coin has been tossed it does not change value */
    forall (r in ROUNDS) for(c = 0; c ≤ 1; c = c + 1) {
            coin2[r][c] : assert G ( coin[r]=c ⇒ G ( coin[r]=c ) );
                    using (coin2[r][c]), coin1[r][c] prove coin2[r][c];
    }

}
```