```
/* BYZANTINE AGREEMENT PROTOCOL */
/* this files contains Cadence SMV part of the proof of probabilistic progress */

/*————————————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];
}

/* ——————————PROBABILISTIC PROGRESS PROOF———————————————*/

/* the prove probabilistic progress has the following structure */
/* consider an arbitrary party i that has not decided before round r+1 and show that */

/* (P1) if there is a concrete pre vote for v in round r+1 before the coin in r+1 is tossed, */
/* then if after the coin in round r+1 is tossed it equals v, then the party decides in round r+1 */

/* (P2) if there are no concrete pre votes in round r+1 before the coin in round r+1 is tossed, */
/* then either the party decides in round r+1 or if after the coins in round r+1 and round r+2 */
/* are tossed they are equal, then the party decides in round r+2 */

/* we consider round r+1 as opposed to round r because in round 0 no coin is tossed */
/* a concrete vote for 0 (1) implies pre_vote[r+1][1][0]>0 (pre_vote[r+1][0][1]>0) */
/* if c=0 or c=1 then we can use (c+1 mod 2) to denote the alternative value of the coin */

/* we therefore assume that the party has not decided before round r+1 */
/* that is the party reaches round r+2 */
```

```
/* and if the process does not decide in round r+1 then it reaches round r+2 */
/* note that this includes fairness requirements: */
/* to reach round r+2 there must be sufficient pre and main votes case in round r+1 */
forall (i in PROC) forall (r in ROUNDS) {
    progress_assumption1[i][r] : assert ( G ( ¬decide[r][i] ) ⇒ F ( round[i]=r+2 ∨ decide[r+1][i] ) );
    progress_assumption2[i][r] : assert ( G ( ¬decide[r][i] ∧ ¬decide[r+1][i] ) ⇒ F ( round[i]=r+3 ∨ decide[r+2][i] ) );
    assume progress_assumption1[i][r], progress_assumption2[i][r];
}

/*——————————————————————————-*/

/* sub for lemmas (P1) (concrete votes before the coin is tossed) */

/* if there is a concrete pre-vote for c then there are main votes for !c in the previous round */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
    inv1[r][c] : assert G ( pre_votes[r+1][c+1 mod 2][c]>0 ⇒ ( main_votes[r][c]>0 ∨ corrupted_main[r][c] ) );
    forall (i in PROC) {
        subcase inv1[r][c][i] of inv1[r][c] for i=history_pre_votes[r+1][c+1 mod 2][c];
        using lemma11[r+1][c+1 mod 2][c],
          lemma25[r][0],
          lemma25[r][1],
          lemma25[r][2],
          corrupted1[r][c],
          /* abstractions */
          VOTES⇒{0,M},
          ROUNDS⇒{r,r+1},
          /* free variables */
          coin//free,
          corrupted_main//free,
          corrupted_main[r][c],
          corrupted_pre//free,
          decide//free,
          main//free,
          main_votes//free,
          main_votes[r][c],
          pre//free,
          pre_proc//free,
          pre_proc_votes//free,
          pre_votes//free,
          pre_votes[r+1][c+1 mod 2][c],
          start//free
          prove inv1[r][c][i];
    }
}
/* if there is a concrete pre-vote for c then there are no concrete pre votes for !c */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
    inv2[r][c] : assert G ( pre_votes[r+1][c+1 mod 2][c]>0 ⇒
            ( pre_votes[r+1][c][c+1 mod 2]=0 ∧ ¬corrupted_pre[r+1][c][c+1 mod 2] ) );
    forall (i in PROC) {
        subcase inv2[r][c][i] of inv2[r][c] for i=history_pre_votes[r+1][c][c+1 mod 2];
        using inv1[r][c],
          lemma11[r+1][c+1 mod 2][c],
          lemma25[r][0],
          lemma25[r][1],
          lemma25[r][2],
          lemma40[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][c+1 mod 2],
            decide//free,
            main//free,
            main_votes//free,
            pre//free,
            pre_proc//free,
            pre_proc_votes//free,
            pre_votes//free,
            pre_votes[r+1][c][c+1 mod 2],
            start//free
            prove inv2[r][c][i];
    }
}
/* if there is a concrete pre-vote for c and the coin equals c then there are no main votes for !c */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
    inv3[r][c] : assert G ( (pre_votes[r+1][c+1 mod 2][c]>0 ∧ coin[r+1]=c) ⇒
                ( main_votes[r+1][c+1 mod 2]=0 ∧ ¬corrupted_main[r+1][c+1 mod 2] ) );
    forall (i in PROC) {
        subcase inv3[r][c][i] of inv3[r][c] for i=history_main_votes[r+1][c+1 mod 2];
        using inv2[r][c],
            lemma11[r+1][0][0],
            lemma11[r+1][0][1],
            lemma25[r+1][c+1 mod 2],
            corrupted1[r+1][c+1 mod 2],
            corrupted2[r+1][c],
            coin2[r+1],
            /* abstractions */
            VOTES⇒{0,M},
            ROUNDS⇒{r,r+1},
            /* free variables */
            coin//free,
            corrupted_main//free,
            corrupted_main[r+1][c+1 mod 2],
            corrupted_pre//free,
            decide//free,
            main//free,
            main_votes//free,
            main_votes[r+1][c+1 mod 2],
            pre//free,
            pre_proc//free,
            pre_proc_votes//free,
            pre_votes//free,
            start//free
            prove inv3[r][c][i];
    }
}
/* if there is a concrete pre-vote for c and the coin equals c then there are no main votes for abstain */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
    inv4[r][c] : assert G ( (pre_votes[r+1][c+1 mod 2][c]>0 ∧ coin[r+1]=c) ⇒
                ( main_votes[r+1][2]=0 ∧ ¬corrupted_main[r+1][2] ) );
    forall (i in PROC){
        subcase inv4[r][c][i] of inv4[r][c] for i=history_main_votes[r+1][2];
        using inv2[r][c],
            lemma11[r+1][c],
            lemma16[r+1][c],
            lemma17[r+1][c],
```

```
                lemma25[r+1][c+1 mod 2],
                corrupted2[r+1][c],
                corrupted6[r+1][c][c],
                coin2[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[r+1],
                main_votes//free,
                main_votes[r+1][2],
                pre//free,
                pre[r+1],
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove inv4[r][c][i];
        }
}

/*————————————————————————-*/

/* sub for lemmas (P2) (no concrete pre votes before the coin is tossed) */

/* if there are no concrete votes for a value then when the coin is tossed the are M pre votes for the coin */
/* first require a one step argument */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
        inv5[r][c] : assert G ( ( (coin[r+1]=not_tossed ⇒
                    (pre_votes[r+1][0][1]=0 ∧ pre_votes[r+1][1][0]=0)) ∧ X (coin[r+1]=c) ) ⇒ pre_votes[r+1][c][c]=M );
        forall (i in PROC) {
                subcase inv5[r][c][i] of inv5[r][c] for i=history_coin[r+1];
                using (inv5[r][c]),
                lemma16[r+1],
                lemma17[r+1],
                coin2[r+1],
                /* abstractions */
                VOTES⇒{0,M},
                ROUNDS⇒{r,r+1},
                /* free variables */
                coin//free,
                coin[r+1],
                corrupted_main//free,
                corrupted_pre//free,
                decide//free,
                main//free,
                main_votes//free,
                pre//free,
                pre[r+1],
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                pre_votes[r+1][c][0],
                pre_votes[r+1][c][1]
```

```
                prove inv5[r][c][i];
        }
}
/* then using the one step case we have what we want */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
        inv6[r][c] : assert G ( ( (coin[r+1]=not_tossed ⇒ (pre_votes[r+1][0][1]=0 ∧ pre_votes[r+1][1][0]=0)) ∧ F (coin[r+1]=c) )
                        ⇒ F (pre_votes[r+1][c][c]=M) );
        using inv5[r][c],
                coin2[r+1],
                /* abstractions */
                VOTES⇒{0,M},
                ROUNDS⇒{r,r+1},
                /* free variables */
                coin//free,
                coin[r+1],
                corrupted_main//free,
                corrupted_pre//free,
                main//free,
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free
                prove inv6[r][c];
}
/* if there are no concrete votes for a value before the coin is tossed then the are no main votes other than for the coin */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
        inv7[r][c] : assert G ( ( (coin[r+1]=not_tossed ⇒(pre_votes[r+1][0][1]=0 ∧ pre_votes[r+1][1][0]=0)) ∧ F (coin[r+1]=c) )
                        ⇒ (main_votes[r+1][c+1 mod 2]=0 ∧ ¬corrupted_main[r+1][c+1 mod 2]) );
        forall (i in PROC) {
                subcase inv7[r][c][i] of inv7[r][c] for i=history_main_votes[r+1][c+1 mod 2];
                using inv6[r][c],
                  lemma9[r+1][c][0][M],
                  lemma9[r+1][c][1][M],
                  lemma11[r+1][c][0],
                  lemma11[r+1][c][1],
                  lemma25[r+1][c+1],
                  coin2[r+1],
                  /* assumptions */
                  assumption4[r+1][c],
                  /* abstractions */
                  VOTES⇒{0,M},
                  ROUNDS⇒{r,r+1},
                  /* free variables */
                  coin//free,
                  coin[r+1],
                  corrupted_main//free,
                  corrupted_main[r+1][c+1 mod 2],
                  corrupted_pre//free,
                  decide//free,
                  main//free,
                  main_votes//free,
                  main_votes[r+1][c+1 mod 2],
                  pre//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free
                  prove inv7[r][c][i];
        }
}
```

```
/* if there are no concrete votes for a value before the coin is tossed then the are no pre votes other than for the coin */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
    inv8[r][c] : assert G ( ( (coin[r+1]=not_tossed ⇒(pre_votes[r+1][0][1]=0 ∧ pre_votes[r+1][1][0]=0)) ∧ F (coin[r+1]=c) )
              ⇒ (pre_votes[r+2][c][c+1 mod 2]=0 ∧ ¬corrupted_pre[r+2][c][c+1 mod 2]) );
    forall (i in PROC) {
        subcase inv8[r][c][i] of inv8[r][c] for i=history_pre_votes[r+2][c][c+1 mod 2];
        using inv7[r][c],
          lemma25[r+1][c+1 mod 2],
          corrupted2[r+2][c][c+1 mod 2],
          coin2[r+1],
          /* abstractions */
          VOTES⇒{0,M},
          ROUNDS⇒{0,r,r+1,r+2},
          /* free variables */
          coin//free,
          coin[r+1],
          corrupted_main//free,
          corrupted_pre//free,
          corrupted_pre[r+2][c][c+1 mod 2],
          decide//free,
          main//free,
          main_votes//free,
          main_votes[r+1][c+1 mod 2],
          pre//free,
          pre_proc//free,
          pre_proc_votes//free,
          pre_votes//free,
          pre_votes[r+2][c][c+1 mod 2]
          prove inv8[r][c][i];
    }
}
/* if there are no concrete votes for a value before the coin is tossed in round r+1 */
/* and the coins in rounds r+1 and r+2 are equal then */
/* there are no main votes other than for the value of the coins */
/* first for concrete votes */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
    inv9[r][c] : assert G ( ( (coin[r+1]=not_tossed ⇒(pre_votes[r+1][0][1]=0 ∧ pre_votes[r+1][1][0]=0)) ∧ F (coin[r+1]=c)
              ∧ F (coin[r+2]=c) ) ⇒ (main_votes[r+2][c+1 mod 2]=0 ∧ ¬corrupted_main[r+2][c+1 mod 2]) );
    forall (i in PROC) {
        subcase inv9[r][c][i] of inv9[r][c] for i=history_main_votes[r+2][c+1 mod 2];
        using inv8[r][c],
          lemma11[r+2][c+1 mod 2],
          lemma25[r+2][c+1 mod 2],
          corrupted1[r+2][c+1 mod 2],
          coin2[r+1],
          coin2[r+2],
          /* abstractions */
          VOTES⇒{0,M},
          ROUNDS⇒{r,r+1,r+2},
          /* free variables */
          coin//free,
          coin[r+1],
          coin[r+2],
          corrupted_main//free,
          corrupted_main[r+2][c+1 mod 2],
          corrupted_pre//free,
          decide//free,
          main//free,
          main_votes//free,
          main_votes[r+2][c+1 mod 2],
```

```
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free
                prove inv9[r][c][i];
        }
}
/* now for abstain votes */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) {
        inv10[r][c] : assert G ( ( (coin[r+1]=not_tossed ⇒(pre_votes[r+1][0][1]=0 ∧ pre_votes[r+1][1][0]=0)) ∧ F (coin[r+1]=c)
                    ∧ F (coin[r+2]=c) ) ⇒ (main_votes[r+2][2]=0 ∧ ¬corrupted_main[r+2][2]) );
        forall (i in PROC) {
                subcase inv10[r][c][i] of inv10[r][c] for i=history_main_votes[r+2][2];
                using inv8[r][c],
                  lemma5[r+2][M],
                  lemma9[r+2][c][c][M],
                  lemma11[r+2][c+1 mod 2],
                  lemma16[r+2][c],
                  lemma17[r+2][c],
                  lemma25[r+2][2],
                  corrupted1[r+2][2],
                  corrupted6[r+2][c][c],
                  coin2[r+1],
                  coin2[r+2],
                  /* abstractions */
                  VOTES⇒{0,M},
                  ROUNDS⇒{r,r+1,r+2},
                  /* free variables */
                  coin//free,
                  coin[r+1],
                  coin[r+2],
                  corrupted_main//free,
                  corrupted_main[r+2][2],
                  corrupted_pre//free,
                  decide//free,
                  main//free,
                  main_votes//free,
                  main_votes[r+2][2],
                  pre//free,
                  pre_proc//free,
                  pre_proc_votes//free,
                  pre_votes//free
                  prove inv10[r][c][i];
        }
}

/*——————————————MAIN PROOF——————————————————————*/

/* proof of (P1): */
/* If there is a concrete pre vote for v in round r+1 before the coin in r+1 is tossed, */
/* then if after the coin in round r+1 is tossed it equals v, then the party decides in round r+1 */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) forall (i in PROC) {
        progress1[r][c][i] : assert
             G ( ( G (¬decide[r][i]) ∧ pre_votes[r+1][c+1 mod 2][c]>0 ∧ coin[r+1]=not_tossed ∧ X (¬coin[r+1]=not_tossed) )
                    ⇒ ( F( coin[r+1]=c) ⇒ F ( decide[r+1][i] ) ) );
        using inv3[r][c],
                inv4[r][c],
                lemma11[r+1],
                lemma25[r+1],
                lemma31[r+1],
```

```
                lemma32[r+1],
                corrupted1[r+1],
                corrupted5[r+1],
                coin2[r+1],
                /* assumptions */
                progress_assumption1[i][r],
                /* abstractions */
                VOTES⇒{0,M},
                ROUNDS⇒{r,r+1},
                /* free variables */
                coin//free,
                coin[r+1],
                corrupted_main//free,
                corrupted_pre//free,
                decide//free,
                decide[r-1][i],
                decide[r][i],
                decide[r+1][i],
                main//free,
                main[r+1],
                main_votes//free,
                pre//free,
                pre_proc//free,
                pre_proc_votes//free,
                pre_votes//free,
                start//free
                prove progress1[r][c][i];
}
/* proof of (P2): */
/* if there are no concrete pre votes in round r+1 before the coin in round r+1 is tossed, */
/* then either the party decides in round r+1 or if after the coins in round r+1 and round r+2 are tossed they are equal, */
/* then the party decides in round r+2 */
forall (r in ROUNDS) for (c = 0; c ≤ 1; c = c + 1) forall (i in PROC) {
     progress2[r][c][i] : assert
G ( ( G (¬decide[r][i]) ∧ coin[r+1]=not_tossed ∧ X (¬coin[r+1]=not_tossed) ∧ pre_votes[r+1][0][1]=0 ∧ pre_votes[r+1][1][0]=0 )
          ⇒ F ( decide[r+1][i] ∨ ( (coin[r+1]=c ∧ coin[r+2]=c) ⇒ decide[r+2][i]) ) );
                using inv9[r][c],
                  inv10[r][c],
                  lemma11[r+1][0][1],
                  lemma11[r+1][1][0],
                  lemma19[r+2][M],
                  lemma23[r+2][c][M],
                  lemma25[r+2],
                  lemma31[r+2],
                  lemma32[r+2],
                  coin2[r+1],
                  coin2[r+2],
                  corrupted5[r+2][c],
                  /* assumptions */
                  progress_assumption1[i][r],
                  progress_assumption2[i][r],
                  /* abstractions */
                  VOTES⇒{0,M},
                  ROUNDS⇒{0,r,r+1,r+2},
                  /* free variables */
                  coin//free,
                  coin[r+1],
                  coin[r+2],
                  corrupted_main//free,
                  corrupted_pre//free,
```

```
                    decide//free,
                    decide[r+1][i],
                    decide[r+2][i],
                    main//free,
                    main[r+2],
                    main_votes//free,
                    main_votes[r+2][c],
                    pre//free,
                    pre_proc//free,
                    pre_proc_votes//free,
                    pre_votes//free
                    prove progress2[r][c][i];
    }


    /* we also require the following property concerning when the coins are tossed (P3) */
    /* whereas property (P4) is given by inv2 in this file and the remaining properties (P5) and (P6) */
    /* can be found in the additional invariants file (lemma9 and coin2) */

    /* if the coin in round r+1 is not tossed then neither is the coin in round r+2 */
    forall (r in ROUNDS) {
        progress3[r] : assert G (coin[r+1]=not_tossed ⇒ coin[r+2]=not_tossed);
        forall (i in PROC) {
            subcase progress3[r][i] of progress3[r] for i=history_coin[r+2];
            using coin2[r+1],
              coin2[r+2],
              /* abstractions */
              ROUNDS⇒{r..r+2},
              /* free variables */
              coin//free,
              coin[r+1],
              coin[r+2],
              corrupted_main//free,
              corrupted_pre//free,
              decide//free,
              main//free,
              main_votes//free,
              pre//free,
              pre_proc//free,
              pre_proc_votes//free,
              pre_votes//free,
              start//free
              prove progress3[r][i];
        }
    }

    /*————————————END OF FILE———————————————*/

}
```