```
/* consensus protocol [AH90] */
/* gxn 12/12/00*/
/* randomization replaced with non-deterministic choice */
/* note used values 1 and 2 not values 0,1 (use 0 to model bottom) */
/* THIS FILE CONTAINS THE PROOF OF LEMMA 6.12 */

/*-----------------------------------------------------------*/

/* CONSTANTS */
/* number of processes */
#define N 10
/* set of processes as ordset to use induction */
ordset PROC 1..N;
/* round numbers as ordset to use induction */
ordset NUM 0..;
/* local phases */
typedef PC {INITIAL, READ1, CHECK1, READ2, CHECK2, DECIDE, NIL};

/*-----------------------------------------------------------*/

module main(act){

/*----------------INPUTS----------------------*/

/* scheduler */
act : PROC;
/* initial values of processes */
start : array PROC of 1..2;

/* ----------------THE PROTOCOL--------------------- */

/* LOCAL VARIABLES */
/* phase */
pc : array PROC of PC;
/* values[i][j] choice of j when last read by i */
values : array PROC of array PROC of 0..2;
/* rounds[i][j] round number of j when last read by i */
rounds : array PROC of array PROC of NUM;
/* counter used for loop when reading */
count : array PROC of PROC;

/* GLOBAL VARIABLES (MEMORY) */
/* value[i] current choice of i */
value : array PROC of 0..2;
/* round[i] current round number of i */
round : array PROC of NUM;

/* INITIAL VALUES */
forall (i in PROC) {
    init(pc[i]) := INITIAL;
    forall (j in PROC) {
        init(rounds[i][j]) := 0;
        init(values[i][j]) := 0;
    }
    init(round[i]) := 0;
    init(value[i]) := 0;
    init(count[i]) := 1;
}
```

```
/* NEXT VALUES (based on the phase of the process) */
/* note only the process being scheduled (process act) moves */
switch (pc[act]) {
    INITIAL : {
        next(value[act]) := start[act];
        next(round[act]) := round[act]+1;
        next(pc[act]) := READ1;
    }
    READ1 : {
        next(pc[act]) := (count[act]=N) ? CHECK1 : READ1;
        next(rounds[act][count[act]]) := round[count[act]];
        next(values[act][count[act]]) := value[count[act]];
        next(count[act]) := count[act]=N ? count[act] : count[act]+1;
    }
    CHECK1 : {
        if (decide[act]) {
            /* all who disagree trail by two and I am a leader */
            next(pc[act]) := DECIDE;
        }
        else if (agree[act][1]) {
            /* all leaders agree on 1 */
            next(pc[act]) := READ1;
            next(count[act]) := 1;
            next(value[act]) := 1;
            next(round[act]) := round[act]+1;
        }
        else if (agree[act][2]) {
            /* all leaders agree on 2 */
            next(pc[act]) := READ1;
            next(count[act]) := 1;
            next(value[act]) := 2;
            next(round[act]) := round[act]+1;
        }
        else {
            next(pc[act]) := READ2;
            next(count[act]) := 1;
            next(value[act]) := 0;
        }
    }
    READ2 : {
        next(pc[act]) := count[act]=N ? CHECK2 : READ2;
        next(rounds[act][count[act]]) := round[count[act]];
        next(values[act][count[act]]) := value[count[act]];
        next(count[act]) := count[act]=N ? count[act] : count[act]+1;
    }
    CHECK2 : {
        if (agree[act][1]) {
            /* all leaders agree on 1 */
            next(pc[act]) := READ1;
            next(count[act]) := 1;
            next(value[act]) := 1;
            next(round[act]) := round[act]+1;
        }
        else if (agree[act][2]) {
            /* all leaders agree on 2 */
            next(pc[act]) := READ1;
            next(count[act]) := 1;
            next(value[act]) := 2;
            next(round[act]) := round[act]+1;
        }
```

```
        else {
            /* guess new value */
            next(pc[act]) := READ1;
            next(count[act]) := 1;
            next(value[act]) := {1,2};
            next(round[act]) := round[act]+1;
        }
    }
    DECIDE : {
        next(pc[act]) := NIL;
    }
};

/*————————————————END OF MAIN PROTOCOL————————————————-*/


/* ————————————FORMULAE WE NEED FOR CHECKING PHASES————————————- */

/* decide[i] true if according to i all that disagree trail by 2 and i is a leader */
decide : array PROC of boolean;
/* array_agree[i][v][j] true if i has read j implies according to i if j is a leader then j agrees on v */
array_agree : array PROC of array 1..2 of array PROC of boolean;
/* agree[i][v] true if according to i all leaders read by process i agree on v */
agree : array PROC of array 1..2 of boolean;
/* array_minus1_agree[i][v][j] true if i has read j then rounds[i][j] ≥ fill_maxr[i]-1 → values[i][j]=1 */
array_minus1_agree : array PROC of array 1..2 of array PROC of boolean;
/* minus1_agree[i][v][j] true if according to i all process with round ≥ fill_maxr[i]-1 read by process i agree on v */
minus1_agree : array PROC of array 1..2 of boolean;

/*Note that inv5 and inv7 (proved in invariants.smv) allow us to use fill_maxr in the definition of array_agree etc */

/* INITIAL VALUES */
forall (i in PROC) {
    init(decide[i]) := 0;
    for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
            init(array_agree[i][v][j]) := 1;
            init(array_minus1_agree[i][v][j]) := 1;
    }
}

forall (i in PROC) {
    next(decide[i]) := ( next(minus1_agree[i][1]) ∨ next(minus1_agree[i][2]) ) ∧ next(fill_maxr[i])=next(round[i]);
}

forall (i in PROC) {
    for(v = 1; v ≤ 2; v = v + 1) {
        forall (j in PROC) {
            if (next(pc[i])=INITIAL ∨ ((next(pc[i])=READ1 ∨ next(pc[i])=READ2) ∧ next(count[i])≤j)) {
            /* not read yet */
            next(array_agree[i][v][j]) := 1;
            next(array_minus1_agree[i][v][j]) := 1;
            }
            else {
            /* already read */
            next(array_agree[i][v][j]) := next(rounds[i][j])≥next(fill_maxr[i]) ⇒ next(values[i][j])=v;
            next(array_minus1_agree[i][v][j]) := next(rounds[i][j])≥next(fill_maxr[i])-1 ⇒ next(values[i][j])=v;
            }
        }
    }
}
```

```
/* conjuction of arrays */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    agree[i][v] := ∧[ array_agree[i][v][j] : j in PROC ];
    minus1_agree[i][v] := ∧[ array_minus1_agree[i][v][j]: j in PROC ];
}

/*————————-EXTRA PREDICATES NEEDED FOR AGREEMENT PROOF————————*/

/* global_maxr the maximum round */
global_maxr : NUM;
/* fill_maxr[i] round i thinks is the maximum round after fill */
fill_maxr : array PROC of NUM;

/* we use +1 for global_maxr and fill_maxr as opposed to next(history_round) as it simplifies proofs */
/* from inv1 and inv2 (proved in invariants.smv) global maxr is correct */
/* from inv4, inv5, inv6 and inv7 (proved in invariants.smv) fill_maxr is correct */

/* INITIAL VALUES */
init(global_maxr) := 0;
forall (i in PROC) init(fill_maxr[i]) := 0;

/* NEXT VALUES */
next(global_maxr) := next(history_round)>global_maxr ? global_maxr+1 : global_maxr;
forall (i in PROC) {
    if (next(pc[i])=INITIAL ∨ ((next(pc[i])=READ1 ∨ next(pc[i])=READ2) ∧ next(count[i])=1))
        /* not read any processes (obs=0) so use global_maxr */
        next(fill_maxr[i]) := next(history_round)>global_maxr ? global_maxr+1 : global_maxr;
    else if ((next(pc[i])=READ1 ∨ next(pc[i])=READ2) ∧ next(count[i])≤act) {
        /* not read process act but read some processes update fill_maxr */
        next(fill_maxr[i]) := next(history_round)>fill_maxr[i] ? fill_maxr[i]+1 : fill_maxr[i];
    }
}
/* array_fill_agree[i][v] true if according to i process j agrees on v after fill*/
array_fill_agree : array PROC of array 1..2 of array PROC of boolean;
/* fill_agree[i][v] true if according to i all leaders agree on v after fill*/
fill_agree : array PROC of array 1..2 of boolean;

/* INITIAL VALUES */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    init(array_fill_agree[i][v][j]) := 0;
}

/* NEXT VALUES */
forall (i in PROC) {
    for(v = 1; v ≤ 2; v = v + 1) {
        forall (j in PROC) {
            if (next(pc[i])=INITIAL ∨ ((next(pc[i])=READ1 ∨ next(pc[i])=READ2) ∧ next(count[i])≤j)) {
            /* not read yet so use global values */
            next(array_fill_agree[i][v][j]) := next(round[j])≥next(fill_maxr[i]) ⇒ next(value[j])=v;
            }
            else {
            /* already read */
            next(array_fill_agree[i][v][j]) := next(rounds[i][j])≥next(fill_maxr[i]) ⇒ next(values[i][j])=v;
            }
        }
    }
}
/* conjuction of arrays */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    fill_agree[i][v] := ∧[ array_fill_agree[i][v][j] : j in PROC ];
```

```
}

/*————EXTRA PREDICATES NEEDED FOR PROBABILISTIC PROGRESS PROOF————————-*/

/* array_fillr_agree[i][r][v][j] true if according to i after fill j has round greater than r imples value[i][j]=v */
array_fillr_agree : array PROC of array NUM of array 1..2 of array PROC of boolean;
/* fill_agree[i][r][v] true if according to i after fill all processes with round greater than r agree on v */
fillr_agree : array PROC of array NUM of array 1..2 of boolean;

/* INITIAL VALUES */
forall (i in PROC) forall (r in NUM) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    init(array_fillr_agree[i][r][v][j]) := 0;
}
/* NEXT VALUES */
forall (i in PROC) forall (r in NUM) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    if (next(pc[i])=INITIAL ∨ ((next(pc[i])=READ1 ∨ next(pc[i])=READ2) ∧ next(count[i])≤j)) {
        /* not read yet so use global values */
        next(array_fillr_agree[i][r][v][j]) := next(round[j])>r ⇒ next(value[j])=v;
    }
    else {
        /* already read */
        next(array_fillr_agree[i][r][v][j]) := next(rounds[i][j])>r ⇒ next(values[i][j])=v;
    }
}
forall (i in PROC) forall (r in NUM) for(v = 1; v ≤ 2; v = v + 1) {
    fillr_agree[i][r][v] := ∧[ array_fillr_agree[i][r][v][j] : j in PROC ];
}

/* to simplify proofs we need the condition of the invariant 7.6 as a single variable */
inv76 : array NUM of boolean;
forall (r in NUM) {
    inv76[r] := ∧[ (round[i]=r ⇒ ¬fill_agree[i][2]) : i in PROC ] ∧ ∧[ fillr_agree[i][r][1] : i in PROC] ∧
                ∧[ (round[i]>r ⇒ value[i]=1) : i in PROC ];
}

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

/* records the current round of the process being scheduled */
history_round : NUM;
init(history_round) := 0;
next(history_round) := next(round[act]);

/* records the process with the global maximum round */
history_maxr : PROC;
init(history_maxr) := act;
next(history_maxr) := next(history_round)>global_maxr ? act : history_maxr;

/* records the process j with round[j] or rounds[i][j] equal to fill_maxr[i] */
history_fill_maxr : array PROC of PROC;
forall (i in PROC) {
    init(history_fill_maxr[i]) := act;
    if (next(pc[i])=INITIAL ∨ ((next(pc[i])=READ1 ∨ next(pc[i])=READ2) ∧ next(count[i])=1))
        /* not read any processes (obs=0) so use global_maxr */
        next(history_fill_maxr[i]) := next(history_round)>global_maxr ? act : history_maxr;
    else if ((next(pc[i])=READ1 ∨ next(pc[i])=READ2) ∧ next(count[i])≤act) {
        /* not read process act but read some processes update fill_maxr */
        next(history_fill_maxr[i]) := next(history_round)>fill_maxr[i] ? act : history_fill_maxr[i];
    }
}
```

```
/*————————————————THE PROOF———————————————————*/

/* THE PROOF OF LEMMA 6.12 */
/* if process j moves or process i moves (not from CHECK1 or CHECK2 to READ1 or READ2) then */
/* if fill_maxr[i]=round[i] & fill_maxr[i][1] & minus1_agree[i][1] holds in the next state */
/* then fill_maxr[i]=round[i] & fill_maxr[i][1] & minus1_agree[i][1] holds in the current state as well */

/*extra invariants*/
forall (i in PROC) forall (j in PROC) {
    inv4[i][j] : assert G ( (count[i]≤j ∧ (pc[i]=READ1 ∨ pc[i]=READ2)) ⇒ fill_maxr[i]≥round[j]);
    assume inv4[i][j];
}
forall (i in PROC) {
    inv8[i] : assert G ( round[i]≤fill_maxr[i] );
    assume inv8[i];
}
forall (r in NUM) forall (i in PROC) {
    inv11[r][i] : assert G ( fill_maxr[i]=r ⇒ G (fill_maxr[i]≥r) );
    assume inv11[r][i];
}
forall (i in PROC) {
    inv13[i] : assert G ( pc[i]=INITIAL ⇒ value[i]=0 );
    assume inv13[i];
}
/* extra variables we need as witness for proving by contradiction */
/* fill_agree */
y1 : array PROC of array 1..2 of PROC;
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    y1[i][v] := { j : j in PROC , ¬(array_fill_agree[i][v][j]) };
}
/* minus1_agree */
y2 : array PROC of array 1..2 of PROC;
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    y2[i][v] := { j : j in PROC , ¬(array_minus1_agree[i][v][j]) };
}
/*case when process j ≠ i moves*/
/*——fill_agree——*/
/* proof first for one element of the array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    lemma1[i][v][j] : assert G ( (¬(act=i) ∧ X (fill_agree[i][v] ∧ fill_maxr[i]=round[i]))
                                        ⇒ (array_fill_agree[i][v][j] ∧ fill_maxr[i]=round[i]) );
    forall (r in NUM) {
        subcase lemma1[i][v][j][r] of lemma1[i][v][j]
        for fill_maxr[i]=r;
        using
            NUM⇒{0,r},
            inv4[i][j],
            inv8[i],
            inv11[r][i],
            inv13[i],
            inv13[j],
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_agree//free,
```

```
            fill_agree[i][v],
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            minus1_agree//free,
            round//free,
            round[i],
            round[j],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            value[j],
            values//free,
            values[i][j]
            prove lemma1[i][v][j][r];
    }
}
/* then prove by contradiction it holds for the whole array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma2[i][v] : assert G ( ( ¬(act=i) ∧ X ( fill_agree[i][v] ∧ fill_maxr[i]=round[i]) ) ⇒ ( fill_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase lemma2[i][v][j][r] of lemma2[i][v]
        for j=y1[i][v] ∧ fill_maxr[i]=r;
        using
            lemma1[i][v][j],
            inv4[i][j],
            inv8[i],
            inv11[r][i],
            inv13[i],
            inv13[j],
            NUM⇒{0,r},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            fill_agree//free,
            fill_agree[i][v],
            history_round//free,
            global_maxr//free,
            minus1_agree//free,
            round//free,
            round[i],
            round[j],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            value[j],
            values//free,
            values[i][j]
            prove lemma2[i][v][j][r];
    }
}
```

```
/*—-minus1_agree—-*/
/* proof first for for one element of the array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    lemma3[i][v][j] : assert G ( ( ¬(act=i) ∧ X (minus1_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( array_minus1_agree[i][v][j] ∧ fill_maxr[i]=round[i] ) );
    forall (r in NUM) {
        subcase lemma3[i][v][j][r] of lemma3[i][v][j]
        for fill_maxr[i]=r;
        using
            NUM⇒{r-1..r},
            inv8[i],
            inv11[r][i],
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            history_round//free,
            fill_maxr[i],
            global_maxr//free,
            minus1_agree//free,
            minus1_agree[i][v],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove lemma3[i][v][j][r];
    }
}
/* then prove by contradiction it holds for the whole array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma4[i][v] : assert G ( ( ¬(act=i) ∧ X ( minus1_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( minus1_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase lemma4[i][v][j][r] of lemma4[i][v]
        for j=y2[i][v] ∧ fill_maxr[i]=r;
        using
            lemma3[i][v][j],
            inv8[i],
            inv11[r][i],
            NUM⇒{r-1..r},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
```

```
            minus1_agree[i][v],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove lemma4[i][v][j][r];
    }
}
/* combining lemma2 and lemma4 we have */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma612a[i][v] : assert G ( ( ¬(act=i) ∧ X ( fill_agree[i][v] ∧ minus1_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( fill_agree[i][v] ∧ minus1_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (r in NUM) {
        subcase lemma612a[i][v][r] of lemma612a[i][v]
            for fill_maxr[i]=r;
            using
            lemma2[i][v],
            lemma4[i][v],
            inv8[i],
            inv11[r][i],
            NUM⇒{0,r-1..r}
            prove lemma612a[i][v][r];
    }
}
/* case when process i moves and is READ1 or READ2 */
/*—–fill_agree—–*/
/* proof first for one element of the array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    lemma5[i][v][j] : assert G ( ( (act=i) ∧ (pc[i]=READ1 ∨ pc[i]=READ2) ∧ X (fill_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( array_fill_agree[i][v][j] ∧ fill_maxr[i]=round[i] ) );
    forall (r in NUM) {
        subcase lemma5[i][v][j][r] of lemma5[i][v][j]
        for fill_maxr[i]=r;
        using
            inv4[i][j],
            inv8[i],
            inv13[i],
            inv13[j],
            inv11[r][i],
            NUM⇒{0,r},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_agree//free,
            fill_agree[i][v],
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            round//free,
```

```
            round[i],
            round[j],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            value[j],
            values//free,
            values[i][j]
            prove lemma5[i][v][j][r];
    }
}
/* then prove by contradiction it holds for the whole array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma6[i][v] : assert G ( ( (act=i) ∧ (pc[i]=READ1 ∨ pc[i]=READ2) ∧ X ( fill_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( fill_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase lemma6[i][v][j][r] of lemma6[i][v]
        for j=y1[i][v] ∧ fill_maxr[i]=r;
        using
            lemma5[i][v][j],
            inv13[i],
            inv13[j],
            inv8[i],
            inv4[i][j],
            inv11[r][i],
            NUM⇒{0,r},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_agree//free,
            fill_agree[i][v],
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            round//free,
            round[i],
            round[j],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            value[j],
            values//free,
            values[i][j]
            prove lemma6[i][v][j][r];
    }
}
/*—-minus1_agree—-*/
/* proof first for one element of the array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    lemma7[i][v][j] : assert G ( ( (act=i) ∧ (pc[i]=READ1 ∨ pc[i]=READ2) ∧ X (minus1_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( array_minus1_agree[i][v][j] ∧ fill_maxr[i]=round[i] ) );
```

```
    forall (r in NUM) {
        subcase lemma7[i][v][j][r] of lemma7[i][v][j]
        for fill_maxr[i]=r;
        using
            NUM⇒{r-1..r},
            PROC⇒{i,j,N},
            inv8[i],
            inv11[r][i],
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            minus1_agree[i][v],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove lemma7[i][v][j][r];
    }
}
/* then prove by contradiction it holds for the whole array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma8[i][v] : assert G ( ( (act=i) ∧ (pc[i]=READ1 ∨ pc[i]=READ2) ∧ X ( minus1_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( minus1_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase lemma8[i][v][j][r] of lemma8[i][v]
        for j=y2[i][v] ∧ fill_maxr[i]=r;
        using
            NUM⇒{r-1..r},
            inv8[i],
            inv11[r][i],
            lemma7[i][v][j],
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            minus1_agree[i][v],
            round//free,
            round[i],
            rounds//free,
```

```
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove lemma8[i][v][j][r];
    }
}
/* combining lemma6 and lemma8 we have */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma612b[i][v] : assert G ( ( act=i ∧ (pc[i]=READ1 ∨ pc[i]=READ2) ∧
                                        X (fill_agree[i][v] ∧ minus1_agree[i][v] ∧ fill_maxr[i]=round[i]))
                                        ⇒ ( fill_agree[i][v] ∧ minus1_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (r in NUM) {
        subcase lemma612b[i][v][r] of lemma612b[i][v]
        for fill_maxr[i]=r;
        using
            lemma6[i][v],
            lemma8[i][v],
            inv8[i],
            inv13[i],
            inv11[r][i],
            NUM⇒{0,r-1..r}
            prove lemma612b[i][v][r];
    }
}
/*case when process i moves to DECIDE*/
/*---fill_agree---*/
/* proof first for one element of the array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    lemma9[i][v][j] : assert G ( ( (act=i) ∧ X ((pc[i]=DECIDE ∨ pc[i]=NIL) ∧ fill_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( array_fill_agree[i][v][j] ∧ fill_maxr[i]=round[i] ) );
    forall (r in NUM) {
        subcase lemma9[i][v][j][r] of lemma9[i][v][j]
        for fill_maxr[i]=r;
        using
            NUM⇒{0,r},
            inv13[i],
            inv13[j],
            inv8[i],
            inv4[i][j],
            inv11[r][i],
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_agree//free,
            fill_agree[i][v],
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            round//free,
            round[i],
            round[j],
```

```
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            value[j],
            values//free,
            values[i][j]
            prove lemma9[i][v][j][r];
    }
}
/* then prove by contradiction it holds for the whole array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma10[i][v] : assert G ( ( (act=i) ∧ X ((pc[i]=DECIDE ∨ pc[i]=NIL) ∧ fill_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( fill_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase lemma10[i][v][j][r] of lemma10[i][v]
        for j=y1[i][v] ∧ fill_maxr[i]=r;
        using
            lemma9[i][v][j],
            inv13[i],
            inv13[j],
            inv8[i],
            inv4[i][j],
            inv11[r][i],
            NUM⇒{0,r},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_agree//free,
            fill_agree[i][v],
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            round//free,
            round[i],
            round[j],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            value[j],
            values//free,
            values[i][j]
            prove lemma10[i][v][j][r];
    }
}
/*—-minus1_agree—-*/
/* proof first for one element of the array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    lemma11[i][v][j] : assert G ( ( (act=i) ∧ X ((pc[i]=DECIDE ∨ pc[i]=NIL) ∧ minus1_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                        ⇒ ( array_minus1_agree[i][v][j] ∧ fill_maxr[i]=round[i] ) );
    forall (r in NUM) {
        subcase lemma11[i][v][j][r] of lemma11[i][v][j]
```

```
        for fill_maxr[i]=r;
        using
            NUM⇒{r-1..r},
            PROC⇒{i,j,N},
            inv8[i],
            inv11[r][i],
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            minus1_agree[i][v],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove lemma11[i][v][j][r];
    }
}
/* then prove by contradiction it holds for the whole array */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma12[i][v] : assert G ( ( (act=i) ∧ X ((pc[i]=DECIDE ∨ pc[i]=NIL) ∧ minus1_agree[i][v] ∧ fill_maxr[i]=round[i]) )
                                                ⇒ ( minus1_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase lemma12[i][v][j][r] of lemma12[i][v]
        for j=y2[i][v] ∧ fill_maxr[i]=r;
        using
            NUM⇒{r-1..r},
            inv8[i],
            inv11[r][i],
            lemma11[i][v][j],
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            minus1_agree[i][v],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
```

```
            value//free,
            values//free,
            values[i][j]
            prove lemma12[i][v][j][r];
    }
}
/* combining lemma10 and lemma12 we have */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    lemma612c[i][v] : assert G ( (act=i ∧ X ((pc[i]=DECIDE ∨ pc[i]=NIL) ∧ fill_agree[i][v] ∧ minus1_agree[i][v] ∧ fill_maxr[i]=round[i]))
                                        ⇒ ( fill_agree[i][v] ∧ minus1_agree[i][v] ∧ fill_maxr[i]=round[i] ) );
    forall (r in NUM) {
        subcase lemma612c[i][v][r] of lemma612c[i][v]
        for fill_maxr[i]=r;
        using
            lemma10[i][v],
            lemma12[i][v],
            inv13[i],
            inv8[i],
            inv11[r][i],
            NUM⇒{0,r-1..r}
            prove lemma612c[i][v][r];
    }
}

/*————————————————-END————————————————————*/

}
```