```
/* consensus protocol [AH90] */
/* gxn 05/12/00*/
/* randomization replaced with non-deterministic choice */
/* this file contains the agreement proof (no two processes decide on different values) */
/* note used values 1 and 2 not values 0,1 (use 0 to model bottom) */
/* THE FILE CONTAINS THE PROOFS OF THE EXTRA INVARIANTS WE NEED */

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

/* 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];
    }
}
```

```
/*————————————————FAIRNESS————————————————-*/

/* we need fairness for some properties */
forall (i in PROC) {
    fair[i] : assert G F (act=i);
    assume fair[i];
}

/*————————————————THE PROOF————————————————-*/

/* properties concerning global_maxr */
/* global_maxr is the global maximum round (since use +1 not next(history_round) in the definition) */
/* this is proved by inv1 and inv2 */
forall (i in PROC) {
    inv1[i] : assert G ( round[i]≤global_maxr );
    forall (r in NUM) {
        subcase inv1[i][r] of inv1[i]
            for round[i]=r;
            using
            (inv1[i]),
            NUM⇒{r-1..r},
            agree//free,
            decide//free,
            start//free,
            value//free
            prove inv1[i][r];
    }
}
forall (i in PROC) {
    inv2[i] : assert G ( (i=history_maxr) ⇒ global_maxr=round[i] );
    forall (r in NUM) {
        subcase inv2[i][r] of inv2[i]
            for round[i]=r;
            using inv1[i] prove inv2[i][r];
    }
}
/* global max always increases */
forall (r in NUM) {
    inv3[r] : assert G ( global_maxr=r ⇒ G (global_maxr≥r) );
}
/* next consider properties of fill_maxr */
/* prove fill_maxr is correct (since use +1 not next(history_round) in the definition) */
/* proved by inv4, inv5, inv6 and inv7 */
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]);
    forall (r in NUM) forall (c in PROC) {
        subcase inv4[i][j][r][c] of inv4[i][j]
        for fill_maxr[i]=r ∧ count[i]=c;
        using (inv4[i][j]),
            inv1[j],
            inv10[i],
            PROC⇒{i,j,N},
            agree//free,
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            pc//free,
```

```
            pc[i],
            round//free,
            round[j]
            prove inv4[i][j][r][c];
    }
}
forall (i in PROC) forall (j in PROC) {
    inv5[i][j] : assert G ( (count[i]>j ∧ (pc[i]=CHECK1 ∨ pc[i]=CHECK2)) ⇒ fill_maxr[i]≥rounds[i][j]);
    forall (r in NUM) forall (c in PROC) {
        subcase inv5[i][j][r][c] of inv5[i][j]
        for fill_maxr[i]=r ∧ count[i]=c;
        using (inv5[i][j]),
            inv1[j],
            inv4[i][j],
            inv10[i],
            PROC⇒{i,j,N},
            agree//free,
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            pc//free,
            pc[i],
            round//free,
            round[j],
            rounds//free,
            rounds[i][j],
            value//free,
            values//free
            prove inv5[i][j][r][c];
    }
}
forall (i in PROC) forall (j in PROC) {
    inv6[i][j] : assert G ( (j=history_fill_maxr[i]) ⇒ ( ((pc[i]=READ1 ∨ pc[i]=READ2) ∧ count[i]≤j) ⇒ fill_maxr[i]=round[j] ));
    forall (r1 in NUM) forall (c in PROC) {
        subcase inv6[i][j][r1][c] of inv6[i][j]
            for round[j]=r1 ∧ count[i]=c;
            using
            (inv6[i][j]),
            inv1[i],
            inv2[j],
            inv4[i][j],
            inv9[i],
            PROC⇒{i,j,c,N},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            round[j],
            rounds//free,
            value//free,
```

```
            values//free
            prove inv6[i][j][r1][c];
    }
}
forall (i in PROC) forall (j in PROC) {
    inv7[i][j] : assert G ( (j=history_fill_maxr[i]) ⇒ ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ count[i]>j) ⇒ fill_maxr[i]=rounds[i][j] ));
    forall (r1 in NUM) forall (c in PROC) {
        subcase inv7[i][j][r1][c] of inv7[i][j]
            for rounds[i][j]=r1 ∧ count[i]=c;
            using
            (inv7[i][j]),
            inv1[i],
            inv2[j],
            inv4[i][j],
            inv6[i][j],
            inv9[i],
            PROC⇒{i,j,c,N},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            round[j],
            rounds//free,
            rounds[i][j],
            value//free,
            values//free
            prove inv7[i][j][r1][c];
    }
}
/* additional invs needed concerning fill_maxr */
/* needed for validity and agreement */
forall (i in PROC) {
    inv8[i] : assert G ( round[i]≤fill_maxr[i] );
    forall (r in NUM) {
        subcase inv8[i][r] of inv8[i]
            for round[i]=r;
            using (inv8[i]),
            inv1[i],
            NUM⇒{r-1..r},
            agree//free,
            decide//free,
            start//free,
            value//free
            prove inv8[i][r];
    }
}
forall (i in PROC) {
    inv9[i] : assert G ( fill_maxr[i]≤global_maxr );
    forall (r in NUM) {
        subcase inv9[i][r] of inv9[i]
            for global_maxr=r;
            using
```

```
            (inv9[i]),
            agree//free,
            decide//free,
            round//free,
            start//free,
            value//free
            prove inv9[i][r];
    }
}
forall (i in PROC) {
    inv10[i] : assert G ( (count[i]=1 ∧ (pc[i]=READ1 ∨ pc[i]=READ2)) ⇒ fill_maxr[i]=global_maxr );
    forall (r in NUM) {
        subcase inv10[i][r] of inv10[i]
            for global_maxr=r;
            using
            (inv10[i]),
            PROC⇒{1,i,N},
            NUM⇒{r-1..r},
            agree//free,
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            pc//free,
            pc[i],
            round//free,
            start//free,
            value//free
            prove inv10[i][r];
    }
}
/* fill_max always increases */
forall (r in NUM) forall (i in PROC) {
    inv11[r][i] : assert G ( fill_maxr[i]=r ⇒ G (fill_maxr[i]≥r) );
    using inv9[i] prove inv11[r][i];
}
/* show agree[i][1] and agree[i][2] cannot both be true */
/* this is needed to prove cases concerning agree[i][2] since we need agree[i][1] to be false */
forall (i in PROC) {
    inv12[i] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2) ⇒ ¬( agree[i][1]∧ agree[i][2] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase inv12[i][j][r] of inv12[i]
            for history_fill_maxr[i]=j ∧ fill_maxr[i]=r;
            using
            inv7[i][j],
            agree//free,
            agree[i][1],
            agree[i][2],
            array_agree//free,
            array_agree[i][1][j],
            array_agree[i][2][j],
            array_minus1_agree//free,
            count//free,
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_maxr//free,
            history_round//free,
```

```
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove inv12[i][j][r];
    }
}
/* simple invariants that speed up proofs */
forall (i in PROC) {
    inv13[i] : assert G ( pc[i]=INITIAL ⇒ value[i]=0 );
    forall (r in NUM) {
        subcase inv13[i][r] of inv13[i]
        for round[i]=r;
    }
}
forall (i in PROC) {
    inv14[i] : assert G ( ¬(pc[i]=INITIAL) ⇒ round[i]>0 );
    forall (r in NUM) {
        subcase inv14[i][r] of inv14[i]
            for round[i]=r;
            using NUM⇒{0,r}
            prove inv14[i][r];
    }
}
forall (i in PROC) {
    inv15[i] : assert G ( pc[i]=INITIAL ⇒ count[i]=1 );
    forall (c in PROC) {
        subcase inv15[i][c] of inv15[i]
        for count[i]=c;
        using
            agree//free,
            array_agree//free,
            decide//free,
            fill_maxr//free,
            global_maxr//free,
            history_round//free,
            rounds//free,
            start//free,
            value//free,
            values//free
            prove inv15[i][c];
    }
}
/* show agree does not change when moving to DECIDE of NIL */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    inv16[i][v][j] : assert G ( ( X (pc[i]=DECIDE ∨ pc[i]=NIL) ∧ agree[i][v]) ⇒ X ( array_agree[i][v][j] ) );
    forall (r in NUM) {
        subcase inv16[i][v][j][r] of inv16[i][v][j]
            for fill_maxr[i]=r;
            using
            inv65[i],
            agree//free,
            agree[i][v],
            array_agree//free,
```

```
            array_agree[i][v][j],
            array_minus1_agree//free,
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove inv16[i][v][j][r];
    }
}
/* then prove for whole array using a witness */
/* however, we first need to know the next value of array_agree */
next_array_agree : array PROC of array 1..2 of array PROC of boolean;
/* then define it to be the next value of array_agree */
/* infact the following is sufficent */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    next_agree[i][v][j] : assert G ( next_array_agree[i][v][j] ⇒ X (array_agree[i][v][j]) );
    assume next_agree[i][v][j];
}
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, ¬next_array_agree[i][v][j] };
}
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    inv17[i][v] : assert G ( ( X (pc[i]=DECIDE ∨ pc[i]=NIL) ∧ agree[i][v]) ⇒ X ( agree[i][v] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase inv17[i][v][j][r] of inv17[i][v]
            for j=y1[i][v] ∧ fill_maxr[i]=r;
            using
            (inv17[i][v]),
            inv16[i][v][j][r],
            next_agree[i][v][j],
            agree//free,
            agree[i][v],
            array_agree//free,
            array_agree[i][v][j],
            next_array_agree[i][v][j],
            array_minus1_agree//free,
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            pc//free,
            pc[i],
```

```
            round//free,
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove inv17[i][v][j][r];
    }
}
/* show minus1_agree does not change when moving to DECIDE of NIL */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    inv18[i][v][j] : assert G ( ( X (pc[i]=DECIDE ∨ pc[i]=NIL) ∧ minus1_agree[i][v]) ⇒ X ( array_minus1_agree[i][v][j] ) );
    forall (r in NUM) {
        subcase inv18[i][v][j][r] of inv18[i][v][j]
            for fill_maxr[i]=r;
            using
            inv14[i],
            inv8[i],
            inv65[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],
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove inv18[i][v][j][r];
    }
}
/* then prove for whole array using a witness */
/* however, we first need to know the next value of array_minus1_agree */
next_array_minus1_agree : array PROC of array 1..2 of array PROC of boolean;
/* then define it to be the next value of array_minus1_agree */
/* infact the following is sufficent */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    next_minus1_agree[i][v][j] : assert G ( next_array_minus1_agree[i][v][j] ⇒ X (array_minus1_agree[i][v][j]) );
    assume next_minus1_agree[i][v][j];
}
y3 : array PROC of array 1..2 of PROC;
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    y3[i][v] := { j : j in PROC, ¬next_array_minus1_agree[i][v][j] };
}
```

```
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    inv19[i][v] : assert G ( ( X (pc[i]=DECIDE ∨ pc[i]=NIL) ∧ minus1_agree[i][v]) ⇒ X ( minus1_agree[i][v] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase inv19[i][v][j][r] of inv19[i][v]
            for j=y3[i][v] ∧ fill_maxr[i]=r;
            using
            (inv19[i][v]),
            inv18[i][v][j][r],
            next_minus1_agree[i][v][j],
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            next_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],
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove inv19[i][v][j][r];
    }
}
/* show fill_agree and agree are the same when a process is in check */
/* need to consider individual elements of each array separately */
/* so introduce witnesses */
y5 : array PROC of array 1..2 of PROC;
y6 : array PROC of array 1..2 of PROC;
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    y5[i][v] := { j : j in PROC, ¬array_fill_agree[i][v][j] };
    y6[i][v] := { j : j in PROC, ¬array_agree[i][v][j] };
}
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    inv20[i][v][j] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]= DECIDE ∨ pc[i]= NIL)
                                        ⇒ ( agree[i][v] ⇒ array_fill_agree[i][v][j] ) );
    forall (r in NUM) {
        subcase inv20[i][v][j][r] of inv20[i][v][j]
            for fill_maxr[i]=r;
            using
            inv65[i],
            PROC⇒{i,j,N},
            agree//free,
            agree[i][v],
            array_agree//free,
            array_agree[i][v][j],
            array_fill_agree//free,
            array_fill_agree[i][v][j],
```

```
            array_minus1_agree//free,
            count//free,
            count[i],
            decide//free,
            global_maxr//free,
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values[i][j]
            prove inv20[i][v][j][r];
    }
}
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    inv21[i][v] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]= DECIDE ∨ pc[i]= NIL) ⇒ ( agree[i][v] ⇒ fill_agree[i][v] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase inv21[i][v][j][r] of inv21[i][v]
            for y5[i][v]=j ∧ fill_maxr[i]=r;
            using
            inv20[i][v][j][r],
            inv65[i],
            agree//free,
            agree[i][v],
            array_agree//free,
            array_agree[i][v][j],
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            array_minus1_agree//free,
            count//free,
            decide//free,
            global_maxr//free,
            fill_agree//free,
            fill_agree[i][v],
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values[i][j]
            prove inv21[i][v][j][r];
    }
}
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    inv22[i][v][j] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]= DECIDE ∨ pc[i]= NIL)
                                    ⇒ ( fill_agree[i][v] ⇒ array_agree[i][v][j] ) );
    forall (r in NUM) {
        subcase inv22[i][v][j][r] of inv22[i][v][j]
            for fill_maxr[i]=r;
            using
            inv65[i],
            PROC⇒{i,j,N},
```

```
            agree//free,
            array_agree//free,
            array_agree[i][v][j],
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            array_minus1_agree//free,
            count//free,
            count[i],
            decide//free,
            global_maxr//free,
            fill_agree//free,
            fill_agree[i][v],
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values[i][j]
            prove inv22[i][v][j][r];
    }
}
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    inv23[i][v] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]= DECIDE ∨ pc[i]= NIL) ⇒ ( fill_agree[i][v] ⇒ agree[i][v] ) );
    forall (j in PROC) forall (r in NUM) {
        subcase inv23[i][v][j][r] of inv23[i][v]
            for y6[i][v]=j ∧ fill_maxr[i]=r;
            using
            inv22[i][v][j][r],
            inv65[i],
            agree//free,
            agree[i][v],
            array_agree//free,
            array_agree[i][v][j],
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            array_minus1_agree//free,
            count//free,
            decide//free,
            global_maxr//free,
            fill_agree//free,
            fill_agree[i][v],
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values[i][j]
            prove inv23[i][v][j][r];
    }
}
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    inv24[i][v] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]= DECIDE ∨ pc[i]= NIL) ⇒ (agree[i][v] = fill_agree[i][v]) );
```

```
    using inv21[i][v], inv23[i][v] prove inv24[i][v];
}
/* round always increases unboundedly or a process decides */
forall (r in NUM) forall (i in PROC) {
    inv25[r][i] : assert G ( round[i]=r ⇒ F G ( round[i]≥r ∨ pc[i]=NIL ) );
    forall (r1 in NUM) {
        subcase inv25[r][i][r1] of inv25[r][i]
        for round[i]=r1;
            using
            fair[i],
            (inv25[r][i][r1]),
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            decide//free,
            fill_maxr//free,
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            start//free,
            value//free,
            values//free
            prove inv25[r][i][r1];
    }
}
/* invariants for fillr_agree */
/* results concerning agree and fillr_agree */
forall (i in PROC) forall (r in NUM) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    inv26[i][r][v][j] : assert G ( round[i]>r ⇒ (fillr_agree[i][r][v] ⇒ array_agree[i][v][j]) );
    using
        inv8[i],
        agree//free,
        array_agree//free,
        array_agree[i][v][j],
        array_minus1_agree//free,
        array_fillr_agree//free,
        array_fillr_agree[i][r][v][j],
        count//free,
        count[i],
        decide//free,
        fillr_agree//free,
        fillr_agree[i][r][v],
        fill_maxr//free,
        fill_maxr[i],
        global_maxr//free,
        history_round//free,
        minus1_agree//free,
        pc//free,
        round//free,
        rounds//free,
        rounds[i][j],
        start//free,
        value//free,
        values//free,
        values[i][j]
```

```
        prove inv26[i][r][v][j];
}
/* now prove for the whole of agree */
/* witness (use earlier witness (y6) */
forall (i in PROC) forall (r in NUM) for(v = 1; v ≤ 2; v = v + 1) {
    inv27[i][r][v] : assert G ( round[i]>r ⇒ (fillr_agree[i][r][v] ⇒ agree[i][v]) );
    forall (j in PROC) {
        subcase inv27[i][r][v][j] of inv27[i][r][v]
            for j=y6[i][v];
            using
            inv26[i][r][v][j],
            agree//free,
            agree[i][v],
            array_agree//free,
            array_agree[i][v][j],
            array_minus1_agree//free,
            array_fillr_agree//free,
            array_fillr_agree[i][r][v][j],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            pc//free,
            round//free,
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove inv27[i][r][v][j];
    }
}
/* results concerning minus1_agree and fillr_agree */
forall (i in PROC) forall (r in NUM) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    inv28[i][r][v][j] : assert G ( round[i]>r+1 ⇒ (fillr_agree[i][r][v] ⇒ array_minus1_agree[i][v][j]) );
    forall (r1 in NUM) {
        subcase inv28[i][r][v][j][r1] of inv28[i][r][v][j]
            for fill_maxr[i]=r1;
            using
            inv8[i],
            NUM⇒{r..r+1,r1-1..r1},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            array_fillr_agree//free,
            array_fillr_agree[i][r][v][j],
            count//free,
            count[i],
            decide//free,
            fillr_agree//free,
            fillr_agree[i][r][v],
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
```

```
            history_round//free,
            minus1_agree//free,
            pc//free,
            round//free,
            rounds//free,
            start//free,
            value//free,
            values//free
            prove inv28[i][r][v][j][r1];
    }
}
/* now prove for the whole of minus1_agree */
/* witness */
y7 : array PROC of array 1..2 of PROC;
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    y7[i][v] := { j : j in PROC, ¬array_minus1_agree[i][v][j] };
}
forall (i in PROC) forall (r in NUM) for(v = 1; v ≤ 2; v = v + 1) {
    inv29[i][r][v] : assert G ( round[i]>r+1 ⇒ (fillr_agree[i][r][v] ⇒ minus1_agree[i][v]) );
    forall (r1 in NUM) forall (j in PROC) {
        subcase inv29[i][r][v][r1][j] of inv29[i][r][v]
            for fill_maxr[i]=r1 ∧ j=y7[i][v];
            using
            inv28[i][r][v][j][r1],
            NUM⇒{r1-1..r1,r..r+1},
            agree//free,
            array_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            array_fillr_agree//free,
            array_fillr_agree[i][r][v][j],
            count//free,
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            history_round//free,
            minus1_agree//free,
            minus1_agree[i][v],
            pc//free,
            round//free,
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values//free,
            values[i][j]
            prove inv29[i][r][v][r1][j];
    }
}
/* lemmas for proving invariant 6.3 */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    inv30[i][v][j] : assert G ( ( minus1_agree[i][v] ∧ fill_agree[i][v] ∧ (fill_maxr[i]=round[i]) ∧ value[i]=v ∧ pc[j]=INITIAL ∧
                                    round[i]+1=round[j]) ⇒ count[i]≤j );
    forall (r in NUM) {
        subcase inv30[i][v][j][r] of inv30[i][v][j]
            for round[i]=r;
            using
            inv64[i],
            inv13[i],
```

```
            /* free variables in cone */
            agree//free,
            array_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            fill_agree//free,
            fill_agree[i][v],
            global_maxr//free,
            pc//free,
            pc[j],
            start//free,
            value//free,
            value[j]
            prove inv30[i][v][j][r];
    }
}
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) forall (k in PROC) {
    inv31[i][v][j][k] : assert G ( (fill_agree[i][v] ∧ (fill_maxr[i]=round[i]) ∧ value[i]=v ∧ count[i]≤k ∧
                                   (pc[i]=INITIAL ∨ pc[i]=READ1 ∨ pc[i]=READ2))
                                           ⇒ (rounds[j][k]≥round[i] ⇒ values[j][k]=v) );
    forall (r1 in NUM) {
        subcase inv31[i][v][j][k][r1] of inv31[i][v][j][k]
            for round[i]=r1;
            using
            inv1[k],
            inv8[i],
            inv10[i],
            inv65[i],
            /* required abstraction */
            PROC⇒{1,i,j,k,N},
            /* free variables in cone */
            agree//free,
            count//free,
            count[i],
            decide//free,
            array_fill_agree//free,
            array_fill_agree[i][v][k],
            fill_agree//free,
            fill_agree[i][v],
            fill_maxr//free,
            fill_maxr[i],
            pc//free,
            pc[i],
            pc[j],
            round//free,
            round[i],
            round[k],
            rounds//free,
            rounds[j][k],
            start//free,
            value//free,
            value[i],
            value[k],
            values//free,
            values[j][k]
```

```
            prove inv31[i][v][j][k][r1];
    }
}

/* show minus1_agree implies fillr_agree a process is in check */
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) forall (j in PROC) {
    inv32[i][v][j] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]= DECIDE ∨ pc[i]= NIL)
                                        ⇒ ( minus1_agree[i][v] ⇒ array_fill_agree[i][v][j] ) );
    forall (r in NUM) {
        subcase inv32[i][v][j][r] of inv32[i][v][j]
            for fill_maxr[i]=r;
            using
            inv65[i],
            inv8[i],
            inv14[i],
            inv5[i][j],
            PROC⇒{i,j,N},
            NUM⇒{r-1..r},
            agree//free,
            array_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            count[i],
            decide//free,
            global_maxr//free,
            minus1_agree//free,
            minus1_agree[i][v],
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values[i][j]
            prove inv32[i][v][j][r];
    }
}
forall (i in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    inv33[i][v] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]=DECIDE ∨ pc[i]=NIL)
                                        ⇒ (minus1_agree[i][v] ⇒ fill_agree[i][v]) );
    forall (j in PROC) forall (r in NUM) {
        subcase inv33[i][v][j][r] of inv33[i][v]
            for y5[i][v]=j ∧ fill_maxr[i]=r;
            using
            inv32[i][v][j][r],
            inv65[i],
            agree//free,
            array_agree//free,
            array_fill_agree//free,
            array_fill_agree[i][v][j],
            array_minus1_agree//free,
            array_minus1_agree[i][v][j],
            count//free,
            decide//free,
            global_maxr//free,
```

```
            fill_agree//free,
            fill_agree[i][v],
            minus1_agree//free,
            minus1_agree[i][v],
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][j],
            start//free,
            value//free,
            values[i][j]
            prove inv33[i][v][j][r];
    }
}

/*————————————INVARIANTS FROM PSL00————————————*/

/* invariant 6.4 from [PSL00] */
forall (i in PROC) {
    inv64[i] : assert G ( pc[i]=INITIAL ⇒ round[i]=0 );
    using
        NUM⇒{0},
        agree//free,
        count//free,
        decide//free
        prove inv64[i];
}
/* slight variant we need */
forall (i in PROC) forall (j in PROC) {
    inv64a[i][j] : assert G ( pc[i]=INITIAL ⇒ rounds[i][j]=0 );
    using
        NUM⇒{0},
        /* abstract variables which do not effect result */
        agree//free,
        count//free,
        decide//free,
        round//free
        prove inv64a[i][j];
}
/* invariant 6.5 from [PSL00] */
/* added extra parts to the cone for new smv */
forall (i in PROC) {
    inv65[i] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]= DECIDE ∨ pc[i]=NIL) ⇒ count[i]=N);
    using
        PROC⇒{i,N},
        agree//free,
        count//free,
        count[i],
        decide//free,
        maxr//free,
        obs_agree//free,
        pc//free,
        pc[i]
        prove inv65[i];
}
/* invariant 6.10 from [PSL00] */
/* split into cases for rounds and values */
/* rounds */
```

```
forall (i in PROC) forall (j in PROC) {
    inv610a[i] : assert G ( (count[i]>i ∨ pc[i]=CHECK1 ∨ pc[i]=CHECK2) ⇒ round[i]=rounds[i][i] );
    forall (r in NUM) forall (c in PROC) {
        subcase inv610a[i][r][c] of inv610a[i]
            for round[i]=r ∧ count[i]=c;
            using (inv610a),
            inv64,
            inv64a,
            /* abstract variables which do not effect result */
            agree//free,
            count//free,
            count[i],
            decide//free,
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][i]
            prove inv610a[i][r][c];
    }

}
/* values */
forall (i in PROC) forall (j in PROC) {
    inv610b[i] : assert G ( (count[i]>i ∨ pc[i]=CHECK2 ∨ pc[i]=CHECK2) ⇒ value[i]=values[i][i] );
    forall (r in NUM) forall (c in PROC) {
        subcase inv610b[i][r][c] of inv610b[i]
            for round[i]=r ∧ count[i]=c;
            using (inv610b),
            inv64,
            inv64a,
            /* abstract variables which do not effect result */
            agree//free,
            count//free,
            count[i],
            decide//free,
            pc//free,
            pc[i],
            round//free,
            rounds//free,
            start//free
            prove inv610b[i][r][c];
    }

}

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

}
```