```
/* consensus protocol [AH90]
/* gxn 23/12/00*/
/* 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 PROOF OF AGREEMENT */

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

/* 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 INVARIANT 6.1 */
/* just consider the case when value[i]=1 */

/* EXTRA INVARIANTS */
/* the following are proved in invariants.smv */
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] ) );
    assume inv17[i][v];
}
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] ) );
    assume inv19[i][v];
}
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]) );
    assume inv24[i][v];
}
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]) );
    assume inv33[i][v];
}
forall (i in PROC) {
    inv65[i] : assert G ( (pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]= DECIDE ∨ pc[i]= NIL) ⇒ count[i]=N);
    assume inv65[i];
}
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] );
    assume inv610a[i];
}
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] );
    assume inv610b[i];
}
/* invariant 6.3b */
forall (i in PROC) forall (j in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    inv63b[i][j][v] : assert G ( (minus1_agree[i][v] ∧ fill_agree[i][v] ∧ fill_maxr[i]=round[i] ∧ value[i]=v)
                                              ⇒ (round[j]≥round[i] ⇒ value[j]=v) );
    assume inv63b[i][j][v];
}
/*preliminary lemmas*/
/* show cannot decide on a value different to its current value */
/* when value is 1 */
forall (i in PROC) {
    agree1[i] : assert G ( (fill_maxr[i]=round[i] ∧ value[i]=1 ∧
            (count[i]>i ∨ pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]=DECIDE ∨ pc[i]=NIL)) ⇒ ¬minus1_agree[i][2] );
    forall (r in NUM) {
        subcase agree1[i][r] of agree1[i]
            for round[i]=r;
            using
            /* required abstraction */
            PROC⇒{i,N},
            /* free variables in cone */
            agree//free,
            array_agree//free,
            array_fill_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][2][i],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            minus1_agree//free,
            minus1_agree[i][2],
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][i],
            value//free,
            value[i],
            values//free,
            values[i][i]
            prove agree1[i][r];
    }
}
/* when value is 2 */
forall (i in PROC) {
    agree2[i] : assert G ( (fill_maxr[i]=round[i] ∧ value[i]=2 ∧
                    (count[i]>i ∨ pc[i]=CHECK1 ∨ pc[i]=CHECK2 ∨ pc[i]=DECIDE ∨ pc[i]=NIL)) ⇒ ¬minus1_agree[i][1] );
    forall (r in NUM) {
        subcase agree2[i][r] of agree2[i]
            for round[i]=r;
            using
            /* required abstraction */
            PROC⇒{i,N},
            /* free variables in cone */
            agree//free,
            array_agree//free,
            array_fill_agree//free,
            array_minus1_agree//free,
            array_minus1_agree[i][1][i],
            count//free,
            count[i],
            decide//free,
            fill_maxr//free,
            fill_maxr[i],
            global_maxr//free,
            minus1_agree//free,
            minus1_agree[i][1],
            pc//free,
            pc[i],
            round//free,
            round[i],
            rounds//free,
            rounds[i][i],
            value//free,
            value[i],
            values//free,
            values[i][i]
            prove agree2[i][r];
    }
}
/* show decide is true once a process has decided */
```

```
forall (i in PROC) {
    agree3[i] : assert G ( (pc[i]=DECIDE ∨ pc[i]=NIL) ⇒ decide[i] );
    forall (r in NUM) {
        subcase agree3[i][r] of agree3[i]
            for round[i]=r;
            using
            inv17[i],
            inv19[i]
            prove agree3[i][r];
    }
}
/* putting the above together */
forall (i in PROC) forall (j in PROC) for(v = 1; v ≤ 2; v = v + 1) {
    agree4[i][j][v] : assert G ( ( value[i]=v ∧ (pc[i]=DECIDE ∨ pc[i]=NIL)) ⇒ (round[j]≥round[i] ⇒ value[j]=v) );
    forall (r in NUM) {
        subcase agree4[i][j][v][r] of agree4[i][j][v]
            for round[i]=r;
            using
            agree1[i][r],
            agree2[i][r],
            agree3[i][r],
            inv33[i][v],
            inv63b[i][j][v],
            /* free variables in cone */
            agree//free,
            agree[i][v],
            array_agree//free,
            array_fill_agree//free,
            array_minus1_agree//free,
            count//free,
            decide//free,
            decide[i],
            fill_maxr//free,
            fill_maxr[i],
            minus1_agree//free,
            minus1_agree[i][v],
            rounds//free,
            start//free,
            values//free
            prove agree4[i][j][v][r];
    }
}
/*agreement */
forall (i in PROC) forall (j in PROC) {
    inv61[i][j] : assert G ( ¬ ( (value[i]=1 ∧ (pc[i]=DECIDE ∨ pc[i]=NIL)) ∧ (value[j]=2 ∧ (pc[j]=DECIDE ∨ pc[j]=NIL)) ) );
    forall (r1 in NUM) forall (r2 in NUM) {
        subcase inv61[i][j][r1][r2] of inv61[i][j]
            for round[i]=r1 ∧ round[j]=r2;
            using
            agree4[i][j][1],
            agree4[j][i][2],
            /* free variables in cone */
            agree//free,
            array_agree//free,
            array_fill_agree//free,
            array_minus1_agree//free,
            count//free,
            decide//free,
            fill_maxr//free,
            minus1_agree//free,
            rounds//free,
            start//free,
            values//free
            prove inv61[i][j][r1][r2];
    }
}

/*———————————————-END———————————————————*/
}
```