```
const  ---- Configuration parameters ----

  NODE_NUM : 2;
  DATA_NUM : 2;

type   ---- Type declarations ----

  NODE : scalarset(NODE_NUM);           -- NODE now consists of
                                        -- only concrete nodes
  DATA : scalarset(DATA_NUM);

  ABS_NODE : union {NODE, enum{Other}}; -- ABS_NODE consists of both
                                        -- concrete and abstract nodes

  CACHE_STATE : enum {Invld, Shrd, Excl};
  CACHE : record State : CACHE_STATE; Data : DATA; end;

  MSG_CMD : enum {Empty, ReqS, ReqE, Inv, InvAck, GntS, GntE};
  MSG : record Cmd : MSG_CMD; Data : DATA; end;

  STATE : record
    Cache : array [NODE] of CACHE;      -- Caches
    Chan1 : array [NODE] of MSG;        -- Channels for Req*
    Chan2 : array [NODE] of MSG;        -- Channels for Gnt* and Inv
    Chan3 : array [NODE] of MSG;        -- Channels for InvAck
    InvSet : array [NODE] of boolean;   -- Set of nodes to be invalidated
    ShrSet : array [NODE] of boolean;   -- Set of nodes having valid copies
    ExGntd : boolean;                   -- Excl copy has been granted
    CurCmd : MSG_CMD;                   -- Current request command
    CurPtr : ABS_NODE;                  -- Current request node, which
                                        -- can be abstract
    MemData : DATA;                     -- Memory data
    AuxData : DATA;                     -- Auxiliary variable for latest data
  end;

var   ---- State variables ----

  Sta : STATE;

---- Initial states ----

ruleset d : DATA do
startstate "Init"
  undefine Sta;
  for i : NODE do
    Sta.Cache[i].State := Invld;
    Sta.Chan1[i].Cmd := Empty;
    Sta.Chan2[i].Cmd := Empty;
    Sta.Chan3[i].Cmd := Empty;
    Sta.InvSet[i] := FALSE;
    Sta.ShrSet[i] := FALSE;
  end;
  Sta.ExGntd := FALSE;
  Sta.CurCmd := Empty;
  Sta.MemData := d;
  Sta.AuxData := d;
end; end;

---- Cache node actions ----

ruleset i : NODE; d : DATA do
rule "Store"
  Sta.Cache[i].State = Excl
==>
var NxtSta : STATE;
begin
```

```
  NxtSta := Sta;
--
  NxtSta.Cache[i].Data := d;
  NxtSta.AuxData := d;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "SendReqS"
  Sta.Cache[i].State = Invld &
  Sta.Chan1[i].Cmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.Chan1[i].Cmd := ReqS;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "SendReqE"
  Sta.Cache[i].State != Excl &
  Sta.Chan1[i].Cmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.Chan1[i].Cmd := ReqE;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "RecvInvS"
  Sta.Cache[i].State != Excl &
  Sta.Chan2[i].Cmd = Inv &
  Sta.Chan3[i].Cmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.Cache[i].State := Invld;
  undefine NxtSta.Cache[i].Data;
  NxtSta.Chan2[i].Cmd := Empty;
  NxtSta.Chan3[i].Cmd := InvAck;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "RecvInvE"
  Sta.Cache[i].State = Excl &
  Sta.Chan2[i].Cmd = Inv &
  Sta.Chan3[i].Cmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.Cache[i].State := Invld;
  undefine NxtSta.Cache[i].Data;
  NxtSta.Chan2[i].Cmd := Empty;
  NxtSta.Chan3[i].Cmd := InvAck;
  NxtSta.Chan3[i].Data := Sta.Cache[i].Data;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "RecvGntS"
  Sta.Chan2[i].Cmd = GntS
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.Cache[i].State := Shrd;
  NxtSta.Cache[i].Data := Sta.Chan2[i].Data;
  NxtSta.Chan2[i].Cmd := Empty;
  undefine NxtSta.Chan2[i].Data;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "RecvGntE"
  Sta.Chan2[i].Cmd = GntE
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.Cache[i].State := Excl;
  NxtSta.Cache[i].Data := Sta.Chan2[i].Data;
  NxtSta.Chan2[i].Cmd := Empty;
  undefine NxtSta.Chan2[i].Data;
--
  Sta := NxtSta;
end; end;

---- Home node state actions ----

ruleset i : NODE do
rule "RecvReqS"
  Sta.CurCmd = Empty &
  Sta.Chan1[i].Cmd = ReqS
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.CurCmd := ReqS;
  NxtSta.CurPtr := i;
  for j : NODE do NxtSta.InvSet[j] := Sta.ShrSet[j] end;
  NxtSta.Chan1[i].Cmd := Empty;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "RecvReqE"
  Sta.CurCmd = Empty & Sta.Chan1[i].Cmd = ReqE
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.CurCmd := ReqE;
  NxtSta.CurPtr := i;
  for j : NODE do NxtSta.InvSet[j] := Sta.ShrSet[j] end;
  NxtSta.Chan1[i].Cmd := Empty;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "SendInvReqS"
  Sta.CurCmd = ReqS &
  Sta.InvSet[i] = TRUE &
  Sta.ExGntd = TRUE &
  Sta.Chan2[i].Cmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.InvSet[i] := FALSE;
  NxtSta.Chan2[i].Cmd := Inv;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "SendInvReqE"
  Sta.CurCmd = ReqE &
  Sta.InvSet[i] = TRUE &
  Sta.Chan2[i].Cmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.InvSet[i] := FALSE;
  NxtSta.Chan2[i].Cmd := Inv;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "RecvInvAckS"
  Sta.CurCmd != Empty &
  Sta.ExGntd = FALSE &
  Sta.Chan3[i].Cmd = InvAck
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.ShrSet[i] := FALSE;
  NxtSta.Chan3[i].Cmd := Empty;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "RecvInvAckE"
  Sta.CurCmd != Empty &
  Sta.ExGntd = TRUE &
  Sta.Chan3[i].Cmd = InvAck
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.ShrSet[i] := FALSE;
  NxtSta.ExGntd := FALSE;
  NxtSta.MemData := Sta.Chan3[i].Data;
  NxtSta.Chan3[i].Cmd := Empty;
  undefine NxtSta.Chan3[i].Data;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "SendGntS"
  Sta.CurCmd = ReqS &
  Sta.CurPtr = i &
  Sta.ExGntd = FALSE &
  Sta.Chan2[i].Cmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.CurCmd := Empty;
  undefine NxtSta.CurPtr;
  NxtSta.ShrSet[i] := TRUE;
  NxtSta.Chan2[i].Cmd := GntS;
  NxtSta.Chan2[i].Data := Sta.MemData;
--
  Sta := NxtSta;
end; end;

ruleset i : NODE do
rule "SendGntE"
  Sta.CurCmd = ReqE &
  Sta.CurPtr = i &
  forall j : NODE do Sta.ShrSet[j] = FALSE end &
  Sta.ExGntd = FALSE &
  Sta.Chan2[i].Cmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.CurCmd := Empty;
  undefine NxtSta.CurPtr;
  NxtSta.ShrSet[i] := TRUE;
  NxtSta.ExGntd := TRUE;
  NxtSta.Chan2[i].Cmd := GntE;
  NxtSta.Chan2[i].Data := Sta.MemData;
--
  Sta := NxtSta;
end; end;

---- Invariant properties ----

invariant "CtrlProp"
  forall i : NODE do forall j : NODE do
    i != j ->
    (Sta.Cache[i].State = Excl -> Sta.Cache[j].State = Invld) &
    (Sta.Cache[i].State = Shrd -> Sta.Cache[j].State = Invld |
                                  Sta.Cache[j].State = Shrd)
  end end;

invariant "DataProp"
  (Sta.ExGntd = FALSE -> Sta.MemData = Sta.AuxData) &
  forall i : NODE do
    Sta.Cache[i].State != Invld -> Sta.Cache[i].Data = Sta.AuxData
  end;

---- Abstract actions ----
```

```
rule "ABS_Stutter" end;

ruleset d : DATA do
rule "ABS_Store"
  TRUE &
-- Noninterference lemma instantiations:
  Sta.ExGntd = TRUE &                   -- Step 3
  forall j : NODE do
    Sta.Cache[j].State = Invld &        -- Step 5
    Sta.Chan2[j].Cmd != GntS &          -- Step 5
    Sta.Chan2[j].Cmd != GntE &          -- Step 5
    Sta.Chan3[j].Cmd != InvAck          -- Step 6
  end
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.AuxData := d;
--
  Sta := NxtSta;
end; end;

-- Actions SendReqS, SendReqE, RecvInvS, RecvInvE, RecvGntS, and RecvGntE
-- are all abstracted by ABS_Stutter.

rule "ABS_RecvReqS"
  Sta.CurCmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.CurCmd := ReqS;
  NxtSta.CurPtr := Other;
  for j : NODE do NxtSta.InvSet[j] := Sta.ShrSet[j] end;
--
  Sta := NxtSta;
end;

rule "ABS_RecvReqE"
  Sta.CurCmd = Empty
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.CurCmd := ReqE;
  NxtSta.CurPtr := Other;
  for j : NODE do NxtSta.InvSet[j] := Sta.ShrSet[j] end;
--
  Sta := NxtSta;
end;

-- Actions SendInvReqS, SendInvReqE, and RecvInvAckS
-- are all abstracted by ABS_Stutter.

rule "ABS_RecvInvAckE"
  Sta.CurCmd != Empty &
  Sta.ExGntd = TRUE &
-- Noninterference lemma instantiations:
  forall j : NODE do
    Sta.Cache[j].State != Excl &        -- Step 1
    Sta.Chan2[j].Cmd != GntE &          -- Step 1
    Sta.Chan3[j].Cmd != InvAck          -- Step 2
  end
==>
```

```
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.ExGntd := FALSE;
-- Noninterference lemma instantiations:
  NxtSta.MemData := Sta.AuxData;        -- Step 4
                                        -- Replaces "undefine NxtSta.MemData;"
--
  Sta := NxtSta;
end;

rule "ABS_SendGntS"
  Sta.CurCmd = ReqS &
  Sta.CurPtr = Other &
  Sta.ExGntd = FALSE
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.CurCmd := Empty;
  undefine NxtSta.CurPtr;
--
  Sta := NxtSta;
end;

rule "ABS_SendGntE"
  Sta.CurCmd = ReqE &
  Sta.CurPtr = Other &
  forall j : NODE do Sta.ShrSet[j] = FALSE end &
  Sta.ExGntd = FALSE
==>
var NxtSta : STATE;
begin
  NxtSta := Sta;
--
  NxtSta.CurCmd := Empty;
  undefine NxtSta.CurPtr;
  NxtSta.ExGntd := TRUE;
--
  Sta := NxtSta;
end;

---- Noninterference lemmas ----

invariant "Lemma_1"
  forall i : NODE do
    Sta.Chan3[i].Cmd = InvAck & Sta.CurCmd != Empty & Sta.ExGntd = true ->
    forall j : NODE do
      j != i -> Sta.Cache[j].State != Excl &  -- Step 1
                Sta.Chan2[j].Cmd != GntE &    -- Step 1
                Sta.Chan3[j].Cmd != InvAck    -- Step 2
    end &
    Sta.Chan3[i].Data = Sta.AuxData           -- Step 4
  end;

invariant "Lemma_2"
  forall i : NODE do
    Sta.Cache[i].State = Excl ->
    Sta.ExGntd = true &                       -- Step 3
    forall j : NODE do
      j != i -> Sta.Cache[j].State = Invld &  -- Step 5
                Sta.Chan2[j].Cmd != GntS &    -- Step 5
                Sta.Chan2[j].Cmd != GntE &    -- Step 5
                Sta.Chan3[j].Cmd != InvAck    -- Step 6
    end
end;
```