1 UNSOUND ABSTRACTION OF RULE NI_LOCAL_GET_GET

For rule NI_LOCAL_GET_GET (Figure 1), its guard contains the conjunct $Dir.HeadPtr \neq src$, and this was abstracted to $Dir.HeadPtr \neq Other$ in [1] (result in Figure 2). However, this is not a conservative abstraction: it neglects the case where $Dir.HeadPtr$ and src are different indices greater than M. In this case, both would be abstracted to $Other$, so that $Dir.HeadPtr \neq src$ is True but $Dir.HeadPtr \neq Other$ is False. We removed this conjunct from the abstraction of the rule (equivalent to abstracting it to True).

```
ruleset src : NODE do
rule "NI_Local_Get_Get"
  src != Home &
  Sta.UniMsg[src].Cmd = UNI_Get &
  Sta.UniMsg[src].Proc = Home &
  Sta.RpMsg[src].Cmd != RP_Replace &
  !Sta.Dir.Pending & Sta.Dir.Dirty &
  !Sta.Dir.Local & Sta.Dir.HeadPtr != src
==>
begin
  Sta.Dir.Pending := true;
  Sta.UniMsg[src].Cmd := UNI_Get;
  Sta.UniMsg[src].Proc := Sta.Dir.HeadPtr;
  if (Sta.Dir.HeadPtr != Home) then
    Sta.FwdCmd := UNI_Get;
  end;
  Sta.PendReqSrc := src;
  Sta.PendReqCmd := UNI_Get;
  Sta.Collecting := false;
endrule;
endruleset;
```

Fig. 1. Rule NI_Local_Get_Get in [1].

```
rule "ABS_NI_Local_Get_Get"
  !Sta.Dir.Pending & Sta.Dir.Dirty &
  !Sta.Dir.Local & Sta.Dir.HeadPtr != Other
==>
begin
  Sta.Dir.Pending := true;
  if (Sta.Dir.HeadPtr != Home) then
    Sta.FwdCmd := UNI_Get;
  end;
  Sta.PendReqSrc := Other;
  Sta.PendReqCmd := UNI_Get;
  Sta.Collecting := false;
endrule;
```

Fig. 2. Rule ABS_NI_Local_Get_Get in [1].

2 RULE NI_InvAck

In [1], rule NI_InvAck and its abstraction Abs_NI_InvAck are shown in Figure 3 and Figure 4. Here NODE in Figure 3 represents parameter list $[1 \cdots N]$, while that in Figure 4 represents parameter list $[1 \cdots M]$. Here statement if b then S endif abbreviates if b then S else skip. This abstraction is not correct because $\bigvee_{p=1}^{N} p \neq i \wedge Sta.Dir.InvSet[p]$ is not *safe*, and $\bigvee_{p=1}^{M} Sta.Dir.InvSet[p]$ is not the abstraction of $\bigvee_{p=1}^{N} p \neq i \wedge Sta.Dir.InvSet[p]$ too. Therefore, the CMP result in [1] is problematic. In order to solve this problem, we split the rule into two rules $\texttt{NI\_InvAck}_1$ and $\texttt{NI\_InvAck}_2$, which are shown in Figure 5 and Figure 6. Notice that $(\bigwedge_{p=1}^{N} p \neq i \longrightarrow \neg Sta.Dir.InvSet[p])$ is the negation of $\bigvee_{p=1}^{N} p \neq i \wedge Sta.Dir.InvSet[p]$. Besides, we find that variable $Sta.LastOtherInvAck$ is not read in any statement or guard but only modified in the FLASH protocol, we remove this variable and the corresponding assignments.

```
ruleset src : NODE do
rule "NI_InvAck"
  src != Home &
  Sta.InvMsg[src].Cmd = INV_InvAck &
  Sta.Dir.Pending & Sta.Dir.InvSet[src]
==>
begin
  Sta.InvMsg[src].Cmd := INV_None;
  Sta.Dir.InvSet[src] := false;
  if (exists p : NODE do p != src &
  Sta.Dir.InvSet[p] end)
  then
    Sta.LastInvAck := src;
    for p : NODE do
      if (p != src & Sta.Dir.InvSet[p]) then
        Sta.LastOtherInvAck := p;
      end;
    end;
  else
    Sta.Dir.Pending := false;
    if (Sta.Dir.Local & !Sta.Dir.Dirty) then
      Sta.Dir.Local := false;
    end;
    Sta.Collecting := false;
    Sta.LastInvAck := src;
  end;
endrule;
endruleset;
```

Fig. 3. Rule NI_InvAck in [1].

```
rule "ABS_NI_InvAck"
  Sta.Dir.Pending & Sta.Collecting &
  Sta.NakcMsg.Cmd = NAKC_None &
  Sta.ShWbMsg.Cmd = SHWB_None &
  forall q : NODE do
    ( Sta.UniMsg[q].Cmd = UNI_Get |
     Sta.UniMsg[q].Cmd = UNI_GetX ->
      Sta.UniMsg[q].Proc = Home ) &
    ( Sta.UniMsg[q].Cmd = UNI_PutX ->
      Sta.UniMsg[q].Proc = Home &
      Sta.PendReqSrc = q )
  end
==>
begin
  if (exists p:NODE do Sta.Dir.InvSet[p] end)
  then
    Sta.LastInvAck := Other;
    for p : NODE do
      if (Sta.Dir.InvSet[p]) then
        Sta.LastOtherInvAck := p;
      end;
    end;
  else
    Sta.Dir.Pending := false;
    if (Sta.Dir.Local & !Sta.Dir.Dirty) then
      Sta.Dir.Local := false;
    end;
    Sta.Collecting := false;
    Sta.LastInvAck := Other;
  end;
endrule;
```

Fig. 4. Rule ABS_NI_InvAck in [1].

```
ruleset src : NODE do
rule "NI_InvAck1"
  Sta.InvMsg[src].Cmd = INV_InvAck &
  Sta.Dir.Pending & Sta.Dir.InvSet[src] &
  (forall p : NODE do p != src -> !Sta.Dir.
       InvSet[p] end)
==>
begin
  Sta.InvMsg[src].Cmd := INV_None;
  Sta.Dir.InvSet[src] := false;
  Sta.Dir.Pending := false;
   if (Sta.Dir.Local & !Sta.Dir.Dirty) then
    Sta.Dir.Local := false;
  end;
  Sta.Collecting := false;
endrule;
endruleset;
```

Fig. 5. Rule NI_InvAck_1.

```
ruleset src : NODE do
rule "NI_InvAck2"
  Sta.InvMsg[src].Cmd = INV_InvAck &
  Sta.Dir.Pending & Sta.Dir.InvSet[src]
==>
begin
  Sta.InvMsg[src].Cmd := INV_None;
  Sta.Dir.InvSet[src] := false;
endrule;
endruleset;
```

Fig. 6. Rule NI_InvAck_2.

```
rule "NI_ShWb"
  Sta.ShWbMsg.Cmd = SHWB_ShWb
==>
  Sta.ShWbMsg.Cmd := SHWB_None;
  undefine Sta.ShWbMsg.Proc;
  Sta.Dir.Pending := false;
  Sta.Dir.Dirty := false;
  Sta.Dir.ShrVld := true;
  for p : NODE do
    Sta.Dir.ShrSet[p] := (p =
         Sta.ShWbMsg.Proc) |
         Sta.Dir.ShrSet[p];
    Sta.Dir.InvSet[p] := (p =
         Sta.ShWbMsg.Proc) |
         Sta.Dir.ShrSet[p];
  end;
endrule;
```

Fig. 7. Rule NI_ShWb in [1].

```
rule "ABS_NI_ShWb"
  Sta.ShWbMsg.Cmd = SHWB_ShWb &
  Sta.ShWbMsg.Proc = Other
==>
begin
  Sta.ShWbMsg.Cmd := SHWB_None;
  undefine Sta.ShWbMsg.Proc;
  undefine Sta.ShWbMsg.Data;
  Sta.Dir.Pending := false;
  Sta.Dir.Dirty := false;
  Sta.Dir.ShrVld := true;
  for p : NODE do
    Sta.Dir.ShrSet[p] := Sta.Dir.ShrSet[p];
    Sta.Dir.InvSet[p] := Sta.Dir.ShrSet[p];
  end;
endrule;
```

Fig. 8. Rule ABS_NI_ShWb in [1].

3 RULE NI_SHWB

Rule NI_ShWb and its abstraction ABS_NI_ShWb according to [1] are shown in Figure 7 and Figure 8. This abstraction is highly irregular. We rewrite the rule NI_ShWb into a form that can be abstracted following our syntax-directed procedure, and so can be processed by autoCMP. It involves adding a parameter *src* and requiring it to equal *Sta.ShWbMsg.Proc* in the conditions, so that within the rule *Sta.ShWbMsg.Proc* can be replaced by *src*. Further, the loop of assignments is split into assignments over $p \neq src$ and assignment on *src*. The result is shown in Figure 9, and the result of abstraction according to syntax-directed procedure is shown in Figure 10, which is equivalent to the result in Figure 8.

```
ruleset src : NODE do
rule "NI_ShWb"
  Sta.ShWbMsg.Cmd = SHWB_ShWb &
  Sta.ShWbMsg.Proc = src
==>
begin
  Sta.ShWbMsg.Cmd := SHWB_None;
  Sta.Dir.Pending := false;
  Sta.Dir.Dirty := false;
  Sta.Dir.ShrVld := true;
  Sta.Dir.ShrSet[src] := true;
  Sta.Dir.InvSet[src] := true;
  for p : NODE do
    if (p != src) then
      Sta.Dir.InvSet[p]:=Sta.Dir.ShrSet[p];
    end;
  end;
  undefine Sta.ShWbMsg.Proc;
endrule;
endruleset;
```

Fig. 9. Our remodelling of rule NI_ShWb.

```
rule "ABS_NI_ShWb"
  Sta.ShWbMsg.Cmd = SHWB_ShWb &
  Sta.ShWbMsg.Proc = Other
==>
begin
  Sta.ShWbMsg.Cmd := SHWB_None;
  Sta.Dir.Pending := false;
  Sta.Dir.Dirty := false;
  Sta.Dir.ShrVld := true;
  for p : NODE do
    Sta.Dir.InvSet[p] := Sta.Dir.ShrSet[p];
  end;
  undefine Sta.ShWbMsg.Proc;
endrule;
```

Fig. 10. Our abstraction of rule NI_ShWb

REFERENCES

[1] Ching-Tsun Chou, Phanindra K. Mannava, and Seungjoon Park. 2004. A Simple Method for Parameterized Verification of Cache Coherence Protocols. In *Proc. 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD'04) (Lecture Notes in Computer Science, Vol. 3312)*. Springer, 382–398.