Chapter 4

Prune away Unnecessary Analysis in Reachability Testing for Analysis Livelock

In previous chapter, we propose a mechanism adopted in the monitor protocol to locate deadlock in reachability testing. In this chapter, we will solve another problem at the stage of race-variant generation, livelock.

4.1 RV Derivation of Reachability Testing

Algorithm*: Derive race-variants form a SYN-sequence by generating a RV-diagram.*
Input*: A SYN-sequence EXELOG*
Output*: A set RV which contains all the possible race-variants of EXELOG*

(1) Name the root node with (0, 0, 0,…,0) and label it "unmarked".
Set the value of version_V1 to version_Vm to 0.
For all nodes from execlog1 to execlogn, set these node to empty.

(2) Choose a node (i1, i2, i3,…, in) which is not marked.
For each ij, $1 \leq ij \leq n$
if ij < length of EXELOGj then
Create a child node (i1, i2, i3,…, ij+1,…, in), label it "unmarked".
Copy the value (version_ V1,…, version_ Vm)
from node (i1, i2, i3,…, in) to node (i1, i2, i3,…, ij+1,…, in).
Copy the (node_execlog1, …, node_execlogn)
from node (i1, i2, i3,…, in) to node (i1, i2, i3,…, ij+1,…, in).
if EXELOGj(ij+1)=R(Vk, ?) then
Append R(Vk, version_Vk) to node_execlogj.
if R(Vk, version_Vk) != EXELOGj(ij+1) then // race-variant
Output this node.
if EXELOGj(ij+1)=W(Vk, ?) then

version_Vk = version_Vk + 1 ;
Append W(Vk, version_Vk) to node_execlogj.
if W(Vk, version_ Vk) != EXELOGj(ij+1) then // race-variant
Output this node.
Label this node "marked".
(3) Repeat step(2) until all nodes are marked.

Figure 4.1: Race-variant derivation algorithm of reachability testing

4.2 Improvement of RV Derivation

In this section, we will propose a mechanism to prune away unnecessary analysis livelocks in reachability testing. First of all, we need to record the program counter relative to the loop statement. Thus, the monitor protocol has to be modified to record the execution-log in new format.

To prune away unnecessary analysis livelocks, we have to add another shared variable during the execution of prefix-based replay. As the previous chapter mentioned, only the version of shared variables would be recorded. This is insufficient for identifying the expanded reachable states contain with analysis livelocks or not. Here, we need to record all the values of the shared variables in *loop.CheckVariableSet*. But it is unnecessary to record the values every times the read operations performed; we only record the values while the loop statements take the action of polling events.

```
1   LoopRead_entry(loop)
2   begin
3     if (mode=MONITOR) then
4       P(loop.lock) ;
5       if (loop.num_of_execution = 0)
6         loop.num_of_execution ++ ;
7       else
8         loop.isDeadlock = 1 ;
9         for all variable from loop.CheckVariableSet
10          P(variable.lock) ;
11          if (variable.RemainWriters > 0)
12            loop.isDeadlock = 0 ;
```

```
13            /* For prune away unnecessary analysis livelocks. */
14            Append (loop.num_of_execution, variable, variable_value)
15               to EXELOOPLOG
16            V(variable.lock) ;
17          if (loop.isDeadlock)
18            claim the program is deadlocked, force terminate
19    end
20
21    LoopRead_exit(loop)
22    begin
23       if (mode=MONITOR) then
24          V(loop.lock) ;
25    end
```

Figure 4.2: Modified protocol to record variable_value

Algorithm*: Derive race-variants with pruning away unnecessary livelocks mechanism.*

Input*: A SYN-sequence EXELOG*

Output*: A set RV which contains all the possible race-variants of EXELOG*

(1) Name the root node with (0, 0, 0,…,0) and label it "unmarked".
Set the value of version_V1 to version_Vm to 0.
For all nodes from execlog1 to execlogn, set these node to empty.

(2) Choose a node (i1, i2, i3,…, in) which is not marked.
For each ij, $1 \leq ij \leq n$

/ Prune */*
For all loop_ProgCountera
in (loop_ProgCounter1, …, loop_ProgCountern)
if any loop_ProgCountera=0
checkout (value_ V1,…, value_ Vm) from EXELOOPLOG
if there is a previous node with same
(loop_ProgCounter1, …, loop_ProgCountern)
and (value_ V1,…, value_ Vm)
Label this node "marked", and jump to (3)

if ij < length of EXELOGj then
Create a child node (i1, i2, i3,…, ij+1,…, in), label it "unmarked".
Copy the value (version_ V1,…, version_ Vm)
from node (i1, i2, i3,…, in) to node (i1, i2, i3,…, ij+1,…, in).
Copy the (node_execlog1, …, node_execlogn)
from node (i1, i2, i3,…, in) to node (i1, i2, i3,…, ij+1,…, in).
Copy the (loop_ProgCounter1, …, loop_ProgCountern)
from node (i1, i2, i3,…, in) to node (i1, i2, i3,…, ij+1,…, in).

```
        if EXELOGj(ij+1)=R(Vk, ?) then
            if EXELOG j(ij+1) is a polling event in loop statement
                loop_ProgCounterj=0;
            else
                loop_ProgCounterj++;
            Append R(Vk, version_Vk) to node_execlogj.
            if R(Vk, version_Vk) != EXELOGj(ij+1) then      // race-variant
                Output this node.
        if EXELOGj(ij+1)=W(Vk, ?) then
            loop_ProgCounterj++;
            version_Vk = version_Vk + 1 ;
            Append W(Vk, version_Vk) to node_execlogj.
            if W(Vk, version_ Vk) != EXELOGj(ij+1) then      // Race-variant
                Output this node.
    Label this node "marked".
(3) Repeat step(2) until all nodes are marked.
```

Figure 4.3: Race-variant derivation algorithm of reachability testing

4.4 An Example

Because of the exhaustive testing that reachability testing accomplished, all the loops appeared in the testing target will become disturber, even the possibilities of them are rare, like livelock. Therefore, we called this kind of livelock, especially effective to the reachability testing, "analysis livelock".

To prune away unnecessary analysis livelock, we must understand how the analysis livelock performed and their characteristics first. For the reason that all the value of shared variables are unknown until runtime. Thus, we adopt the symbolic execution for the pruning technique.

Assume that there are n processes (P1…Pn) in the testing target program. At initialization stage, the first step is to confirm the number of instructions to consist with the loop statements in each process. This number called the length of loop

statements, and is a key to estimate how many instructions to be executed will get into the same program counter as previous. And if the succeeded ones have the same shared variable value, then we prune the succeeded ones from RV-diagram because of they have the same program status.

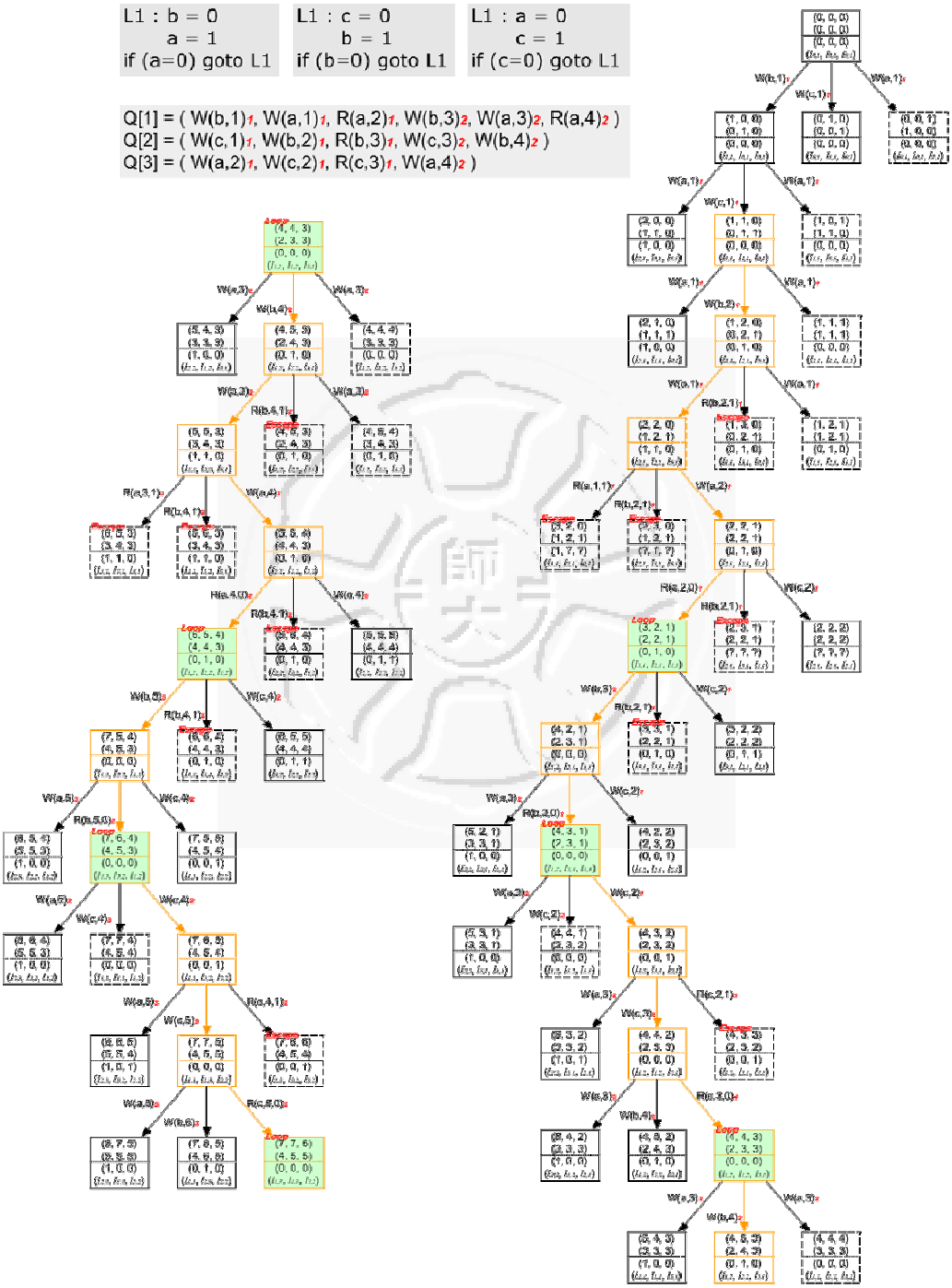

Figure 4.4: Digest from RV-diagram

In Figure 4.4, we found that all processes are consisting of three instructions for a loop statement. Thus, we exam all the node with index vector, which one number of the index vector equal to three and determine whether it have the same shared variable value. In this case, the node with index vector (1, 1, 3) and the succeeded node with index vector (4, 4, 3) have the same program status. So we can prune the node with index vector (4, 4, 3) from original RV-diagram to avoiding unnecessary analysis livelock effects.