

Let's start with the examples from the document I sent:

1. Consider the a sequence  $R_{send}$ , which identifies that a message was sent, and a sequence  $R_{ack}$ , which identifies an acknowledgment. The property

```
assert property (@(posedge clk)
    R_send |->
    (R_ack ##0 (1,  $\alpha$ ) intersect 1[*0:10])
)
```

requires that for every message there should be acknowledge that ends after at most 10 cycles. In this case we would not expect  $\alpha$  to be executed if the acknowledgment ends after more than 10 cycles. For this example, we prefer the tight approach. This type of assertion is common in practice.

2. Consider a sequence  $R_{send}$ , which identifies that a message was sent to two addresses, and the sequences  $R_{ack1}$ , and  $R_{ack2}$ , which identify acknowledgments from address 1 and address 2 respectively. Then, the property

```
assert property (@(posedge clk)
    R_send |->
    (R_ack1 ##0 (1,  $\alpha$ )) and (R_ack2 ##0 (1,  $\beta$ ))
)
```

requires that for every message, both addresses will acknowledge. In this example, we would like the two sub sequences " $R_{ack1} ##0 (1, \alpha)$ " and " $R_{ack2} ##0 (1, \beta)$ ", to be evaluated independently, including an independent execution of  $\alpha$  and  $\beta$ . Thus we prefer the loose approach. This is also a representative form when some independent requirements should be fulfilled whenever some temporal event  $R$  happen, and in order to make the spec more concise all the requirements are joined with **and** as the consequent of  $R$ .

I think that it is interesting to look at derived operators. Derived operators defined using **intersect** are the most interesting since they show the difference between the tight and loose approaches.

- $R_1$  and  $R_2 \equiv$   
 $((R_1 ##1 1[*0:\$]) \text{ intersect } R_2)$   
or  
 $(R_1 \text{ intersect } (R_2 ##1 1[*0:\$]))$ .

Suppose w.l.o.g. that there is an action placed in  $R_1$ . Then by the loose approach, the evaluation of  $R_2$  does not effect the execution of the action. On the other hand, by the tight approach, if the evaluation of  $R_2$  is terminated and there were no matches of  $R_2$  in previous cycles, then the evaluation of  $R_1$  is being terminated. This may effect the execution of the

action. Note that the loose and tight approaches are consistent with the loose and tight definitions for property **and**.

- $R_1$  within  $R_2 \equiv ( 1[*0:\$] \##1 R_1 \##1 1[*0:\$] )$  intersect  $R_2$ .

Note that the evaluation of  $1[*0:\$] \##1 R_1 \##1 1[*0:\$]$  never terminates, because every prefix of every computation matches  $1[*0:\$]$ . Thus, for actions placed in  $R_2$ , there is no difference between the tight and loose approaches.

However, when an action is placed in  $R_1$ , there is a difference. By the tight approach, a termination the evaluation of  $R_2$  should terminate the evaluation of  $1[*0:\$] \##1 R_1 \##1 1[*0:\$]$ . On the other hand, by the loose approach, it does not. Since the evaluations of  $1[*0:\$] \##1 R_1 \##1 1[*0:\$]$  never terminates on its own, it will continually be evaluated until the end of the simulation.

I cannot imagine a scenario where this is the desired behavior. Thus, for this operator, I think that the tight approach is more appropriate.

- $b$  throughout  $R \equiv b [*0:\$]$  intersect  $R$ .

Here again, I would expect a termination of the evaluation of  $b [*0:\$]$ , to terminate the evaluation of  $R$ , thus, the tight approach is more appropriate.

For properties, the only relevant derived operator is **if (b) then  $P_1$  else  $P_2$** . For this operator the tight and loose approaches have the same semantics.

Here is a property using the **or** operator. Assume that the sequence  $R_{\text{send}}$  identifies that a message was sent; that the sequence  $R_{\text{resp1}}$  identifies one type of respond to the message; and that the sequence  $R_{\text{resp2}}$  identifies a second type of respond. Then, the property

```
property foo;
  R_send |->
  ##1 (R_resp1 or R_resp2);
endproperty
```

requires that after every message, there will be a respond of either the first type or the second. We should distinguish between two cases: If the responds are mutually exclusive, then there is no difference between the tight and loose approaches. However, if both may happen at the computation, I don't think that the success of the evaluation of  $R_{\text{resp1}}$  should terminate the evaluation of  $R_{\text{resp2}}$  and vice versa. Thus, the loose approach is more appropriate.