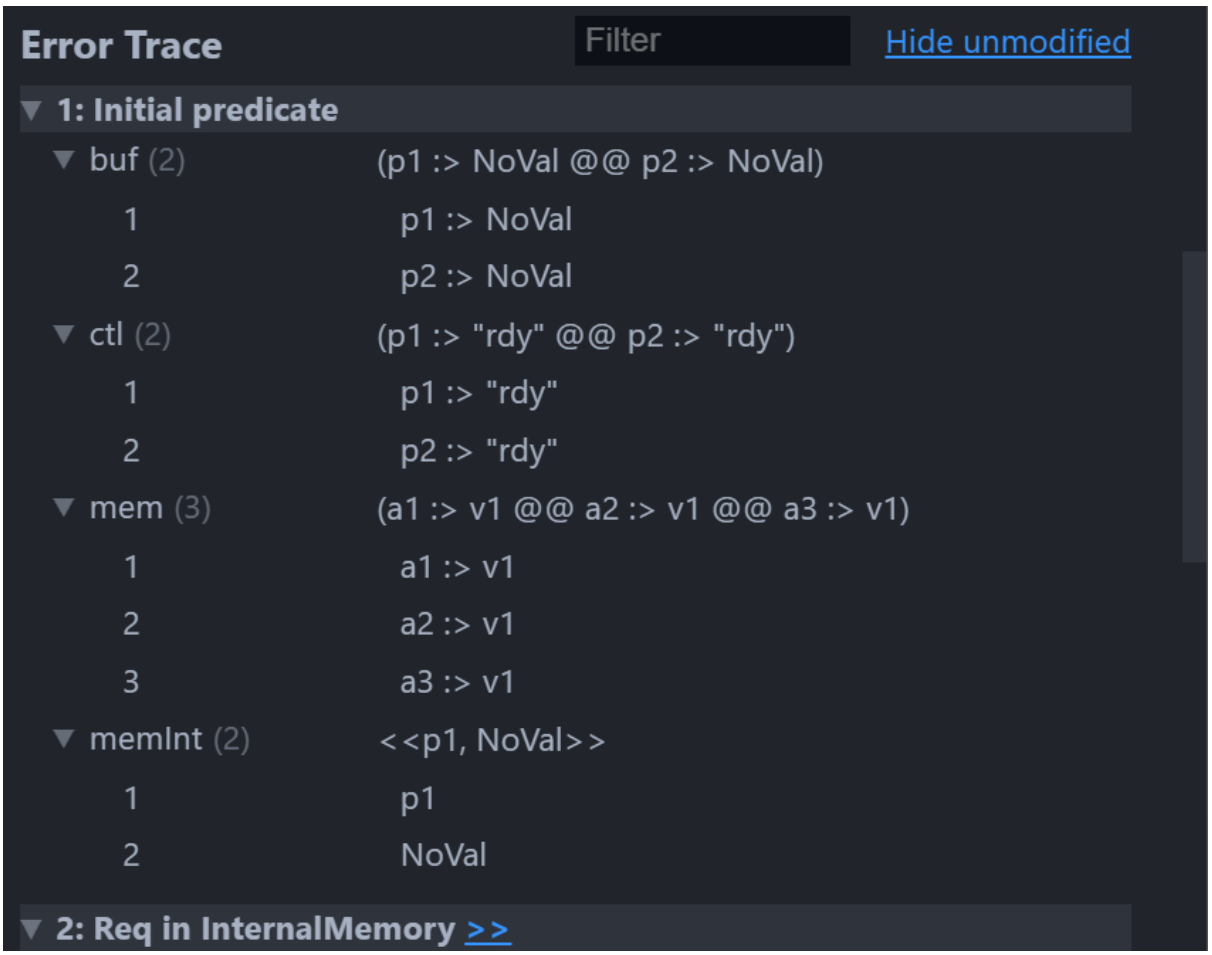
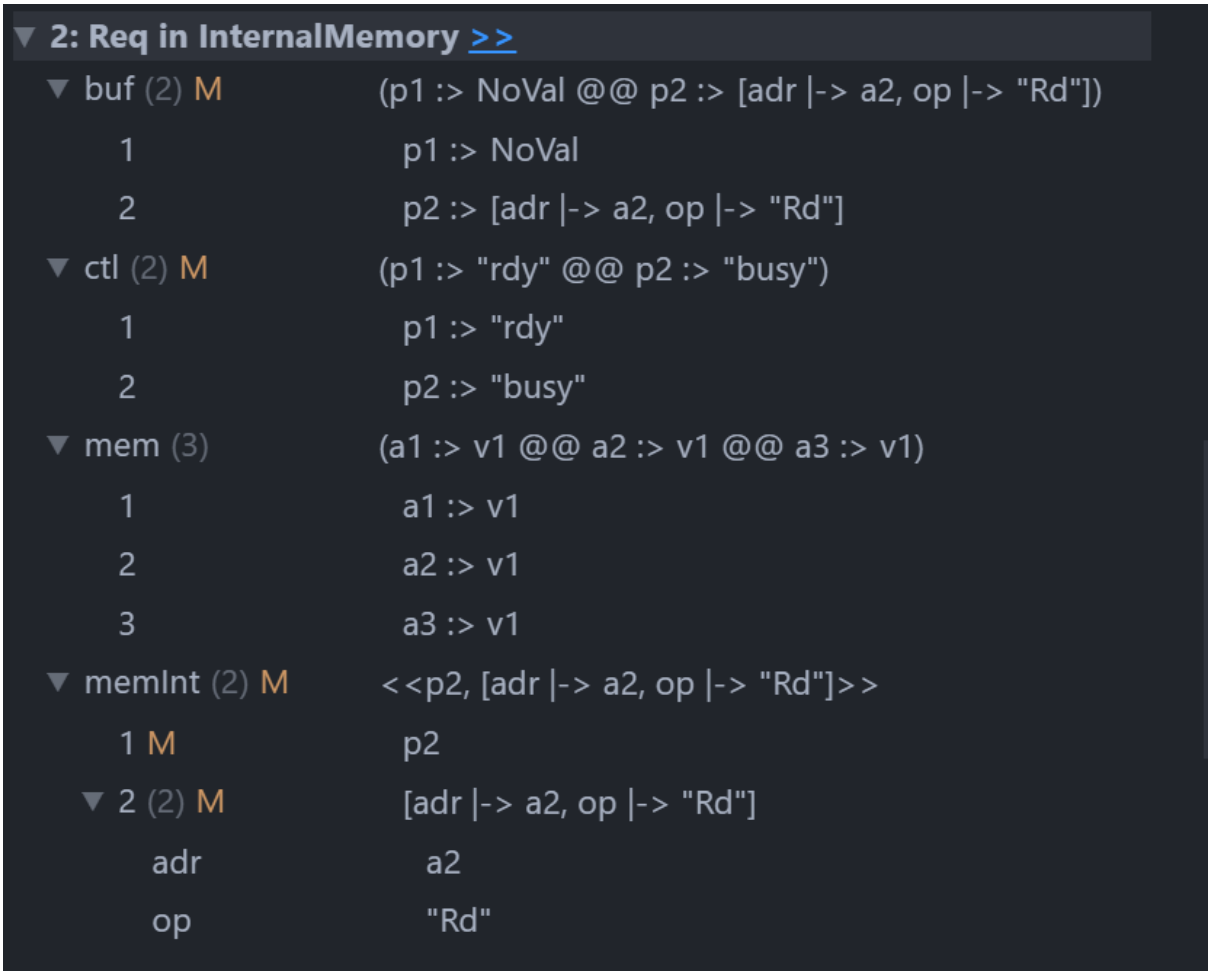
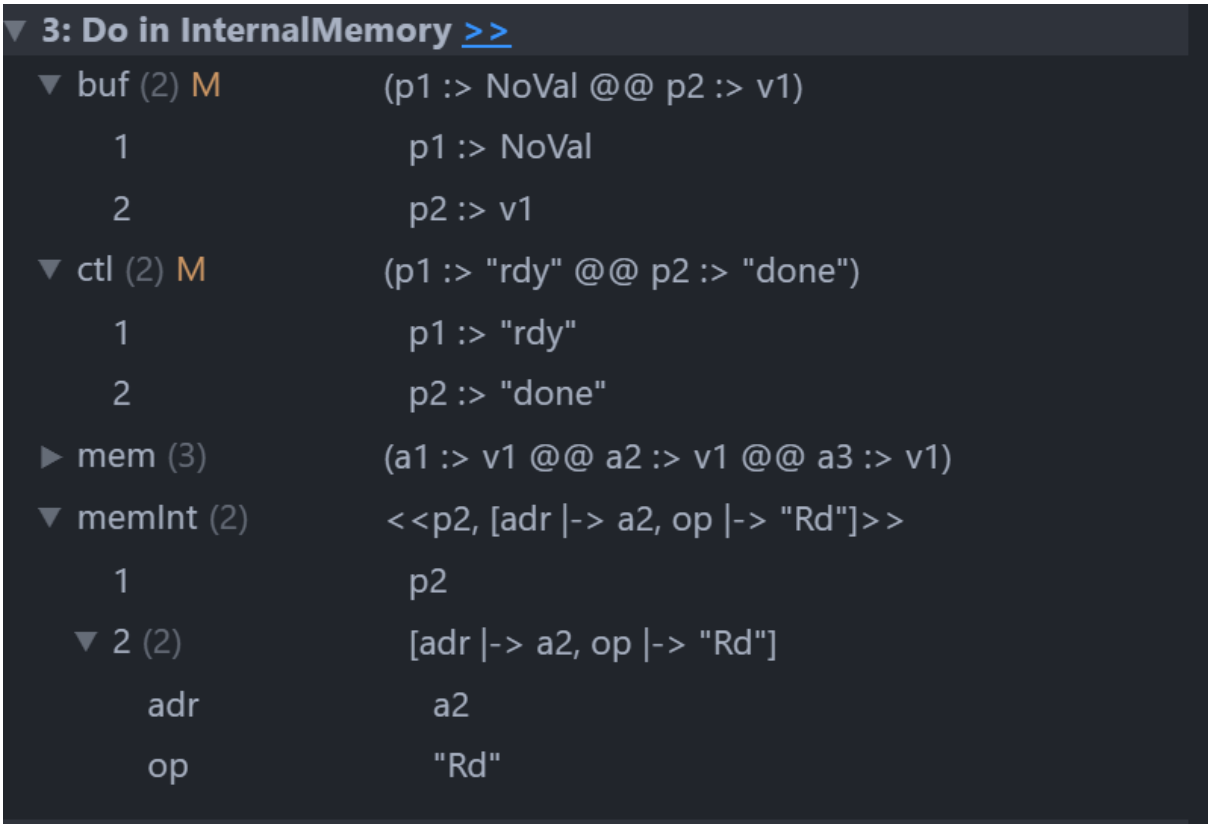
1/ Couterexample of $\forall p \in Proc : (\text{ctl}[p] = \text{"rdy"}) ; (\text{ctl}[p] = \text{"busy"})$ is



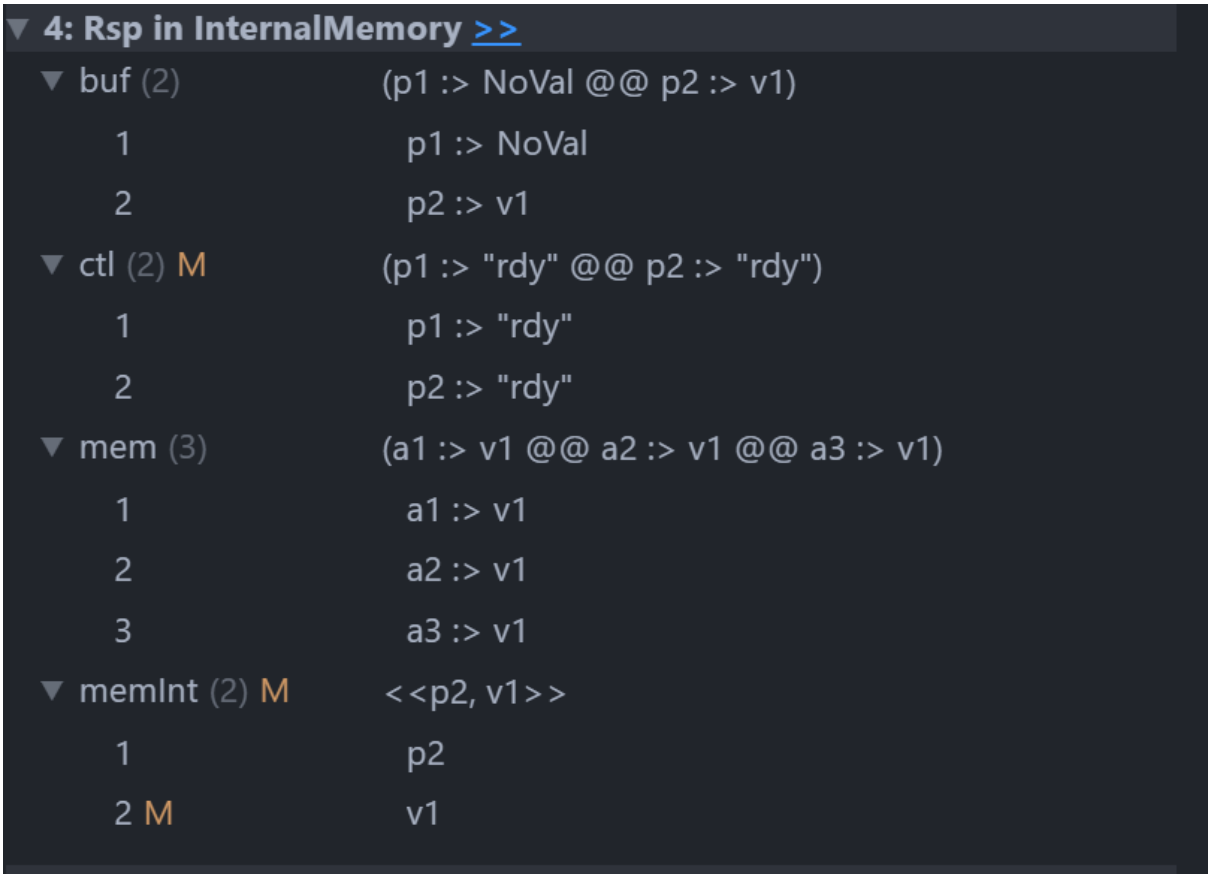
- Buf: both p1 and p2 is “rdy” state with NoVal
- Ctl : both p1,p2 in “rdy” state
- Mem: a1,a2,a3 has the “v1” value
- memInt: memory Interface initialized with p1 at NoVal



- buf: p2 process a read request “Rd” for “a2” address
- ctl : p2 changes to “busy” state
- mem : unchanged
- menInt: updated “p2” and the read request



- buf: p2 now has the “v1” value
- ctl: “p2” updated to “done: state
- mem: unchanged
- memInt: unchanged

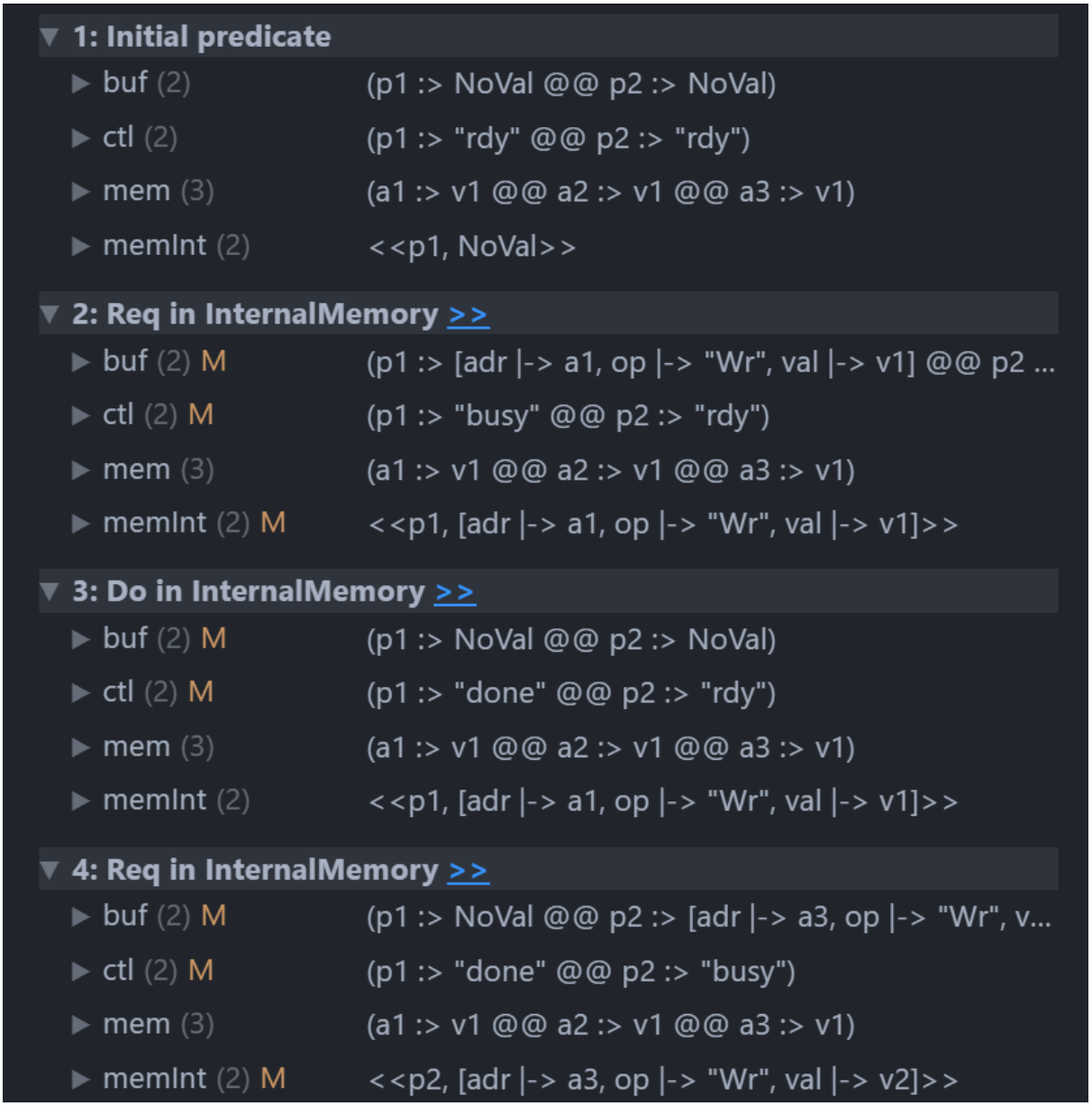


- buf: p2 still has the “v1” value
- ctl: “p2” changed to “rdy”: state
- mem: unchanged
- memInt: changed with p2 and value v1

in Request in Internal Memory step, p2 do a read request and change to “busy” state, continue at the next step, it completed the process and changed to “done”. But in the Response in Internal Memory p2 change back to “rdy” without the middle “busy” state. This violates the LivenessProperty we redefined, which required everytime a processor do a request, it should eventually changes to “busy” before change to “rdy”.

2/

The different is LISpec include ISpec and adding a liveness condition to make sure Do or Rsp will eventually happen for each processor.



The missing of liveness condition when we use ISpec instead of LISpec could lead to the situation that response not eventually created. This show in the output above, response of “p1” is created “done but response of “p2” still processing “busy”, which is a violation of liveness property. The liveness condition could stop this situation by make sure that every request will eventually receives a response.