# LUSTRE by Ali Dhanani 000470296

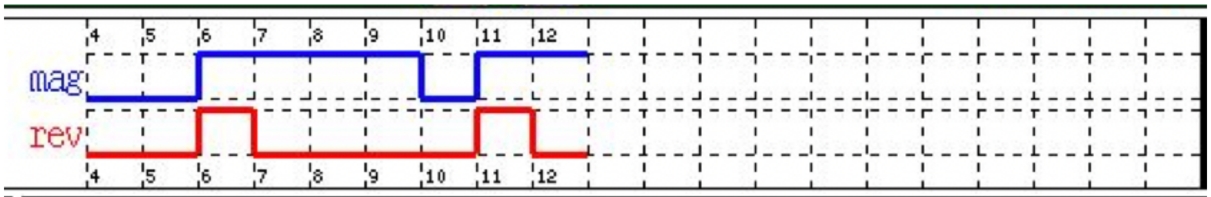
## Question

Q1) What type of malfunctioning can cause lower-than-actual reported speeds: anomalies or failures (if we use the broken sensor for measuring speed)?

Ans) Failure can cause the lower than actual reported speed, as we have supposed that the we are using the broken sensor for measuring speed which means either the front or the back sensor has failed returning an alarm level to be 2. So, when we are using the speedometer, we will only be considering one of the sensors either the front or the back which is not broken for the speed. As one of the conditions of Task5 is that “At any time point, if the speed changes, then we observe a rotation event from a working sensor.” Also, in the condition when there is a broken sensor and the alarm value is being 2, then the torque requests are also 0.

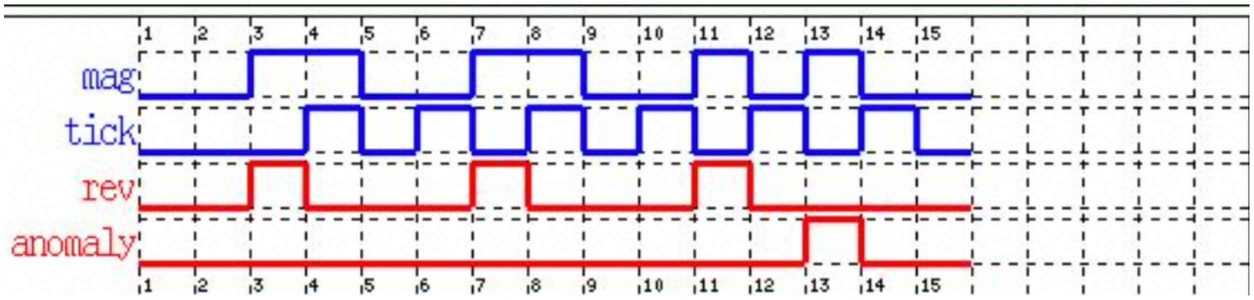
## Tasks

### Task L0



Here the idea is that whenever there is a magnet(mag), then there is a trigger wheel-rotation(rev), but if the mag is true for more count then the rev will be false. There cannot be a condition where rev can be true for more than 1 count in the condition where mag is true for more than one count.

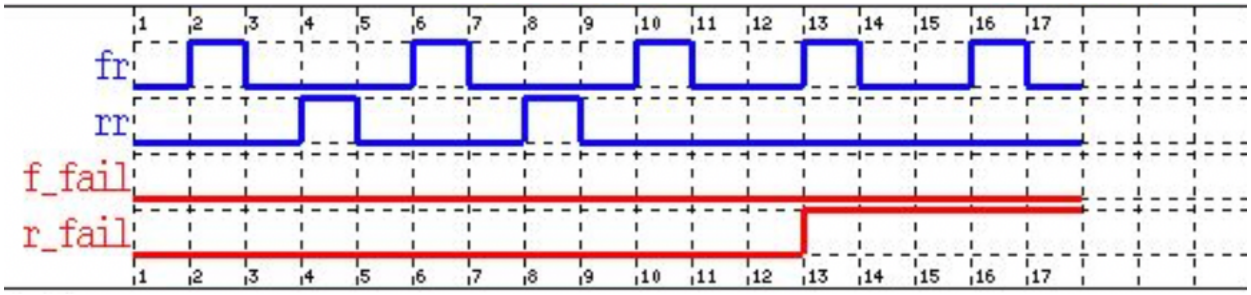
### Task L1



Here we are using the global constant” NOF_TICKS_ANOMALY” which value is 2, the idea of an anomaly is that it will be true when mag is called before 2 tick count. For the first mag, the rev will be true. Rev will only be true when tricks counts are 2 or more then 2. Both anomaly and rev cannot be true at the same time. The maximum number anomaly can be 100. Here we will be using the same technique as of Task L0, where for more than 1 mag, the rev will only appear once. For the first mag there will not be anomaly but rev instead.

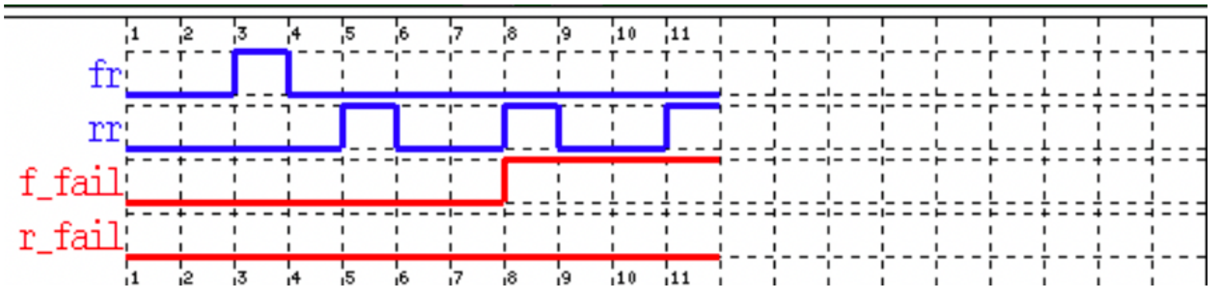
## Task L2

For r_failure



Here we are following the pattern of FR RR FR. Initially both the front sensor and the rear sensor are ok, which means that none of them fail. If we see that the pattern is not following the correct pattern in this case where we see 2 fronts in time 10 and 13, where there is no rr in between which shows that the pattern is violated, and rear sensor fails. Here when the rear sensor failed, it is going to remain fail and the pattern now is FR FR.

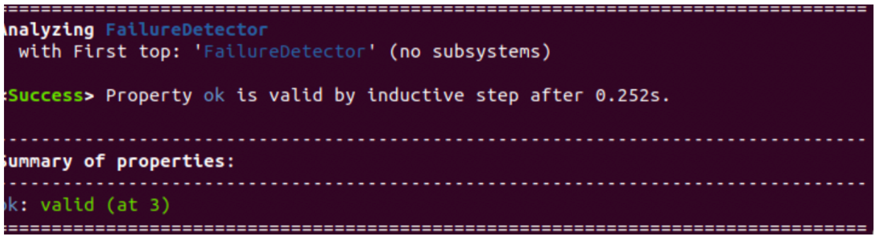
For f_fail



Here we have the same thing as that in the previous figure, but the only difference is that here front sensor fails, and it remains fail throughout. Now the new pattern is RR RR.

In the failure detector, no 2 sensors can fail together. Also, if one sensor fails it cannot return to be normal again. Also, if a sensor then the other sensor cannot fail. We will be checking this property for verification using Kind2. For that we have created a new node for verification. There are 2 conditions that we have verified the results are as follows:

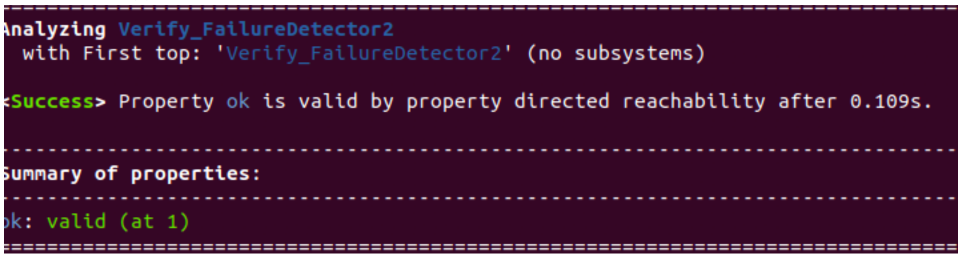
*Failure Verify to check that both sensors cannot fail together*



```
==============================================================================
nalyzing FailureDetector
 with First top: 'FailureDetector' (no subsystems)

Success> Property ok is valid by inductive step after 0.252s.

------------------------------------------------------------------------------
ummary of properties:
------------------------------------------------------------------------------
k: valid (at 3)
==============================================================================
```

Here I have used the condition as not (f_fail and r_fail) to check that 2 sensors cannot fail together.

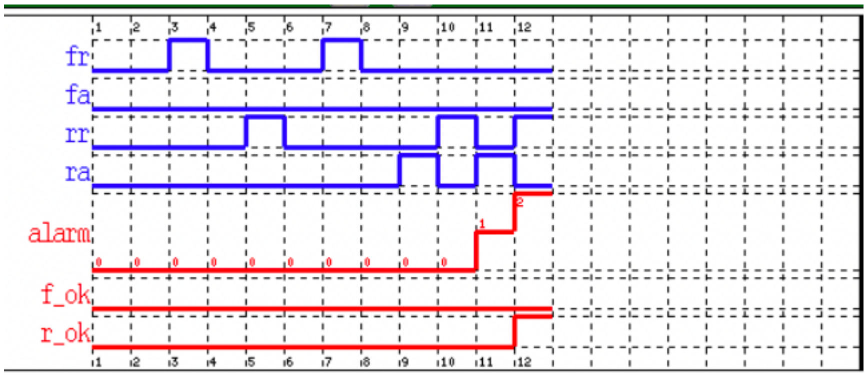
*Failure Verify 2 to check that if a sensor has failed it cannot be restored*



```
Analyzing Verify_FailureDetector2
  with First top: 'Verify_FailureDetector2' (no subsystems)

<Success> Property ok is valid by property directed reachability after 0.109s.

------------------------------------------------------------------------------
Summary of properties:
------------------------------------------------------------------------------
ok: valid (at 1)
==============================================================================
```

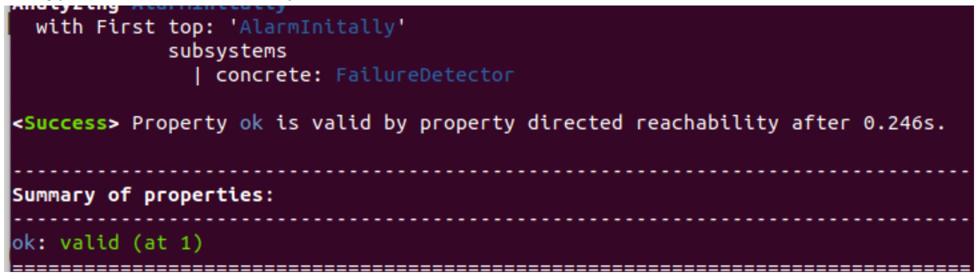
Here I have used the idea to check the counter of what is happening. As the condition says that once it fails it cannot be restored. I will check with condition that if it restores and was previously fail then say condition not true otherwise say it verified.

## Task L3



The idea here is that initially both the front sensor and the rear sensor are working ok. We are using here the idea from the previous failure detector node as well to check for failure. If there is a failure in any sensor then we can say that sensor is not ok which will increase the alarm to 2. The alarm remains to 2 unless it is reset to 0. The reset is not done here, it is done is the speedometer task done in UPPAAL. Also, after 100 anomaly the alarm level increase to 1. For testing we check for the maximum anomaly to be 5. Now we will be verifying some conditions with Kind2 which are as follows with results. Here a separate node was created for verification.

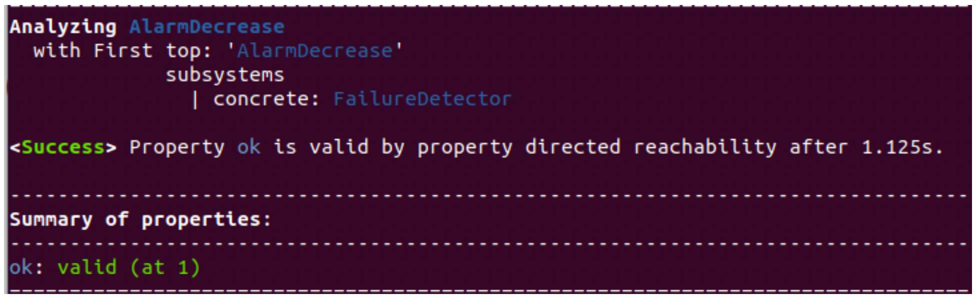
*Verify Alarm to be Initally 0*



```
Analyzing AlarmInitally
  with First top: 'AlarmInitally'
             subsystems
               | concrete: FailureDetector

<Success> Property ok is valid by property directed reachability after 0.246s.

------------------------------------------------------------------------------
Summary of properties:
------------------------------------------------------------------------------
ok: valid (at 1)
==============================================================================
```

Here I have used the condition" alarm=0 -> true" to check if initially alarm is 0.

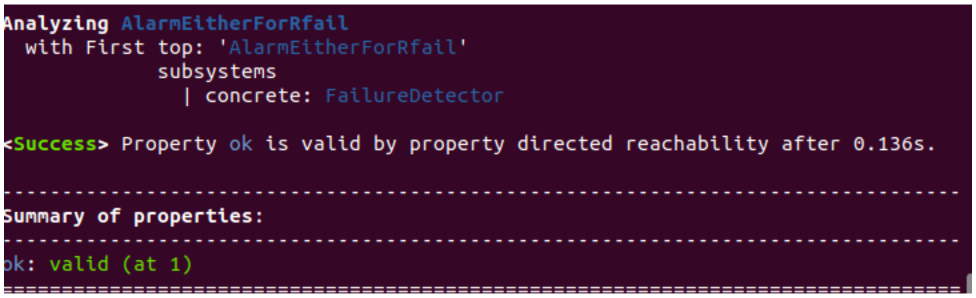
### *Verify Alarm non-strictly increases*



```
Analyzing AlarmDecrease
  with First top: 'AlarmDecrease'
             subsystems
               | concrete: FailureDetector

<Success> Property ok is valid by property directed reachability after 1.125s.

--------------------------------------------------------------------------------
Summary of properties:
--------------------------------------------------------------------------------
ok: valid (at 1)
================================================================================
```

Here with using the if condition, I have checked that if previously alarm is 0 then it can be 1 or 2. If previously alarm is 1 then it can be 2 but not 0. In the condition where alarm is 2 it cannot go back to being 0 or 1. If none of the condition violate then it is verified otherwise it is not verified.

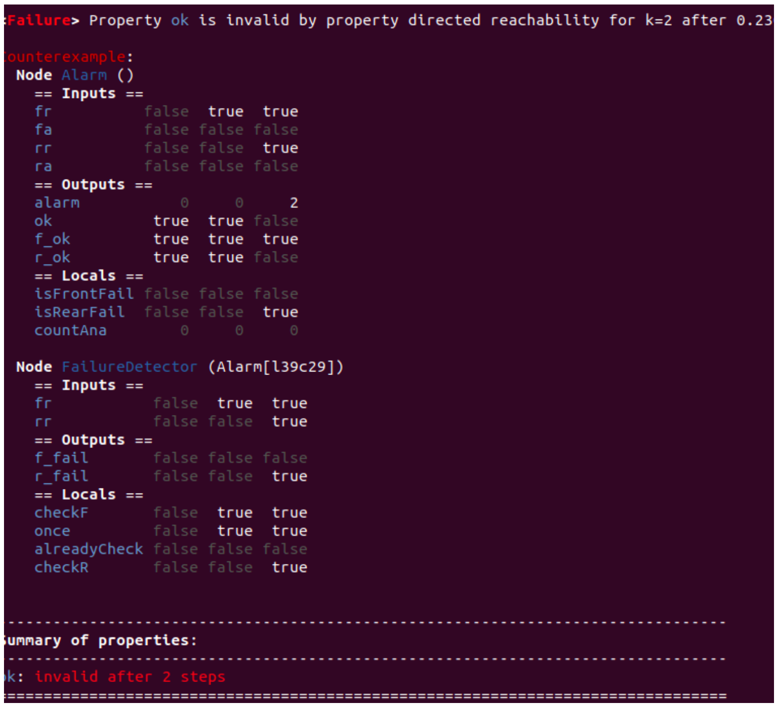
### *Verify Alarm 2 if either f_fail or r_fail is true*



```
nalyzing AlarmEitherForRfail
  with First top: 'AlarmEitherForRfail'
             subsystems
               | concrete: FailureDetector

<Success> Property ok is valid by property directed reachability after 0.136s.

--------------------------------------------------------------------------------
ummary of properties:
--------------------------------------------------------------------------------
ok: valid (at 1)
================================================================================
```

Here to verified I have placed the xor between f_fail and r_fail to check either one be 1, in the case of that and the alarm level being 0 or 1, the condition does not verify, otherwise it verifies. To do verification here I have used the counter work.

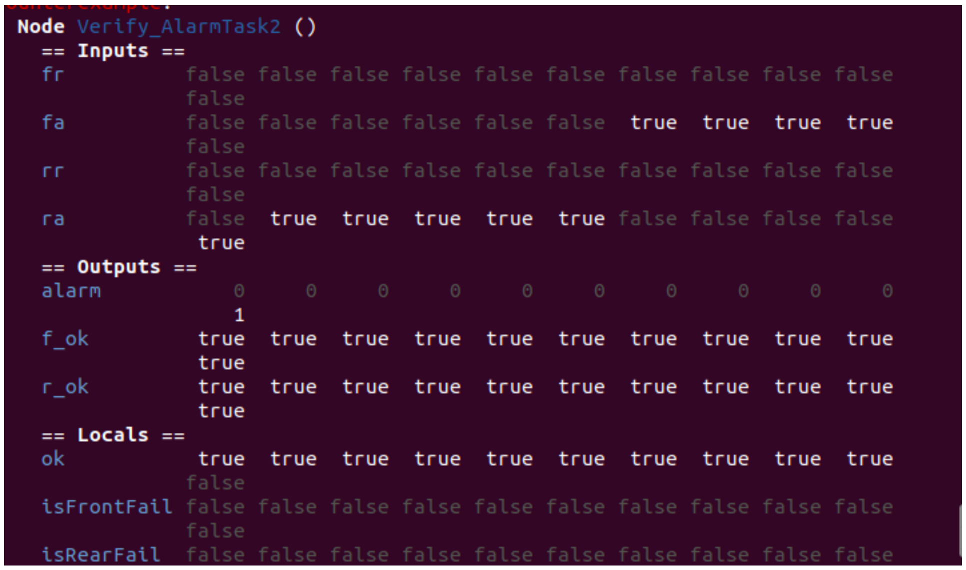
### *Verify for alarm = 2*



```
Failure> Property ok is invalid by property directed reachability for k=2 after 0.23

ounterexample:
 Node Alarm ()
   == Inputs ==
   fr           false  true  true
   fa           false false false
   rr           false false  true
   ra           false false false
   == Outputs ==
   alarm            0     0     2
   ok            true  true false
   f_ok          true  true  true
   r_ok          true  true false
   == Locals ==
   isFrontFail false false false
   isRearFail  false false  true
   countAna         0     0     0

 Node FailureDetector (Alarm[l39c29])
   == Inputs ==
   fr            false  true  true
   rr            false false  true
   == Outputs ==
   f_fail        false false false
   r_fail        false false  true
   == Locals ==
   checkF        false  true  true
   once          false  true  true
   alreadyCheck false false false
   checkR        false false  true


--------------------------------------------------------------------------------
ummary of properties:
--------------------------------------------------------------------------------
k: invalid after 2 steps
================================================================================
```

Here we can see that it eventually goes to 2. As defined in the project manual there is no way to check for all the branches so we will be using a counter example which will surely be invalid. Which is “not (alarm = 2)”, that goes from 0 0 2, showing there is a way to make alarm 2.

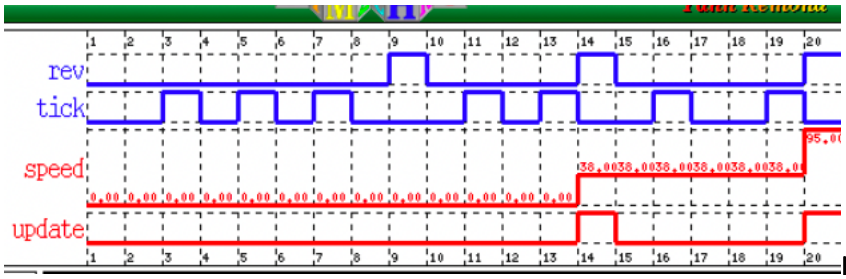
### *Verify for Alarm=1*



```
Node Verify_AlarmTask2 ()
  == Inputs ==
  fr          false false false false false false false false false false
              false
  fa          false false false false false false  true  true  true  true
              false
  rr          false false false false false false false false false false
              false
  ra          false  true  true  true  true  true false false false false
               true
  == Outputs ==
  alarm           0     0     0     0     0     0     0     0     0     0
                  1
  f_ok         true  true  true  true  true  true  true  true  true  true
               true
  r_ok         true  true  true  true  true  true  true  true  true  true
               true
  == Locals ==
  ok           true  true  true  true  true  true  true  true  true  true
              false
  isFrontFail false false false false false false false false false false
              false
  isRearFail  false false false false false false false false false false
```

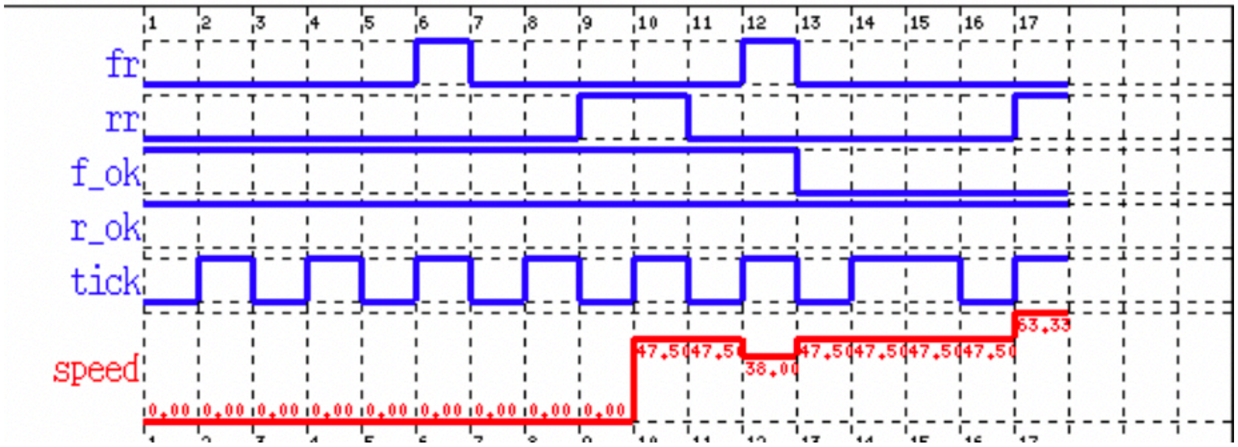
Here we can see that it eventually goes to 1. Same technique that we had used for alarm 2, here we will be using for alarm one which is “not (alarm = 1)”, which is that there is eventually a way to turn the alarm level to 1.

### Task L4



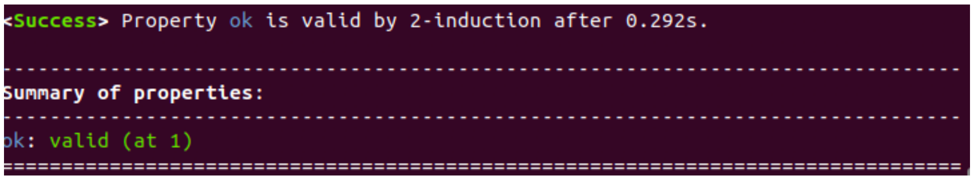
Here we have created the node for naivespeed, where we have used the formula 1.9/(tick count) x 0.01. Here for the first rev there is no speed effect, but the second rev, we are using the formula as above and have used the tick counts. Once the speed value changes the update also becomes true and then for the next comes false again. Every time the speed changes update becomes true as well. Speed is there when the rev parameter becomes true. When rev is true, the counter of ticks becomes back to 0 and the counter starts again. For the formula we will be first using the counter for formula and then initializing the counter back to 0.

## Task L5



Here the idea is that if any one of the sensors which is either f_ok or r_ok are being false at that point the speed will be 0. FR and RR are using the same pattern as it was used in the failure detector. We are aware that it depends on the NaiveSpeed node made in the previous task. The NaiveSpeed node is used on both the FR and RR. We know that in FR and RR there is the return of update as well other than speed. The visual speed will be of the one showing update true. If the update is true for FR than the FR speed is shown. Speed as we saw in the previous task was related to the count value. We will be using the Kind2 for verification where we will be verifying the condition for seeing the initial speed being 0 and to check that the speed changes with the rotation update.

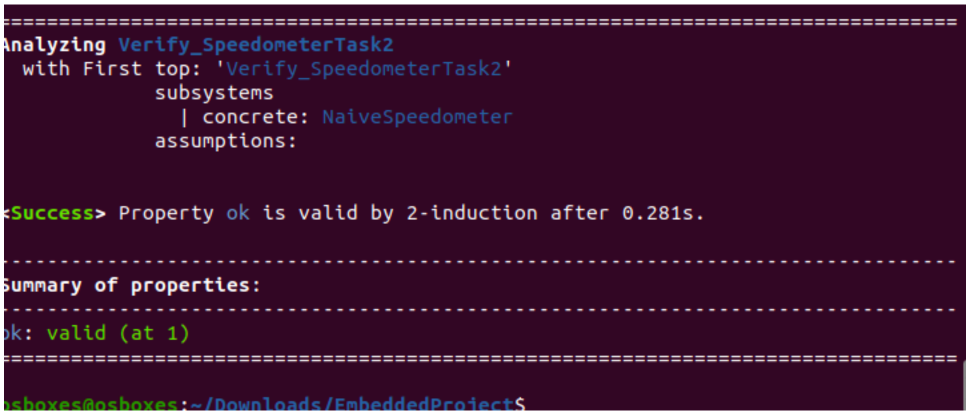
### *Verify Speed initially 0*



```
<Success> Property ok is valid by 2-induction after 0.292s.

--------------------------------------------------------------------------------
Summary of properties:
--------------------------------------------------------------------------------
ok: valid (at 1)
================================================================================
```

For this we using " (speed=0.0 -> true)" to verify that the initial speed is 0.

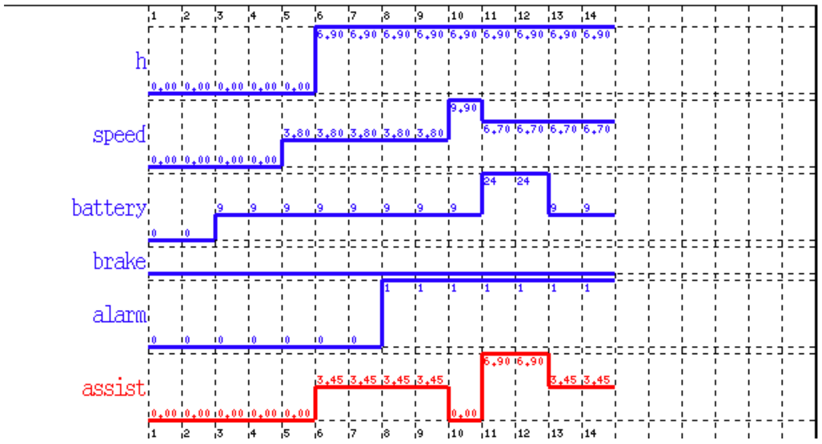
### *Verify that at any point of the speed changes then there is a rotation event.*



```
================================================================================
Analyzing Verify_SpeedometerTask2
  with First top: 'Verify_SpeedometerTask2'
              subsystems
                | concrete: NaiveSpeedometer
              assumptions:


<Success> Property ok is valid by 2-induction after 0.281s.

--------------------------------------------------------------------------------
Summary of properties:
--------------------------------------------------------------------------------
ok: valid (at 1)
================================================================================

osboxes@osboxes:~/Downloads/EmbeddedProject$
```

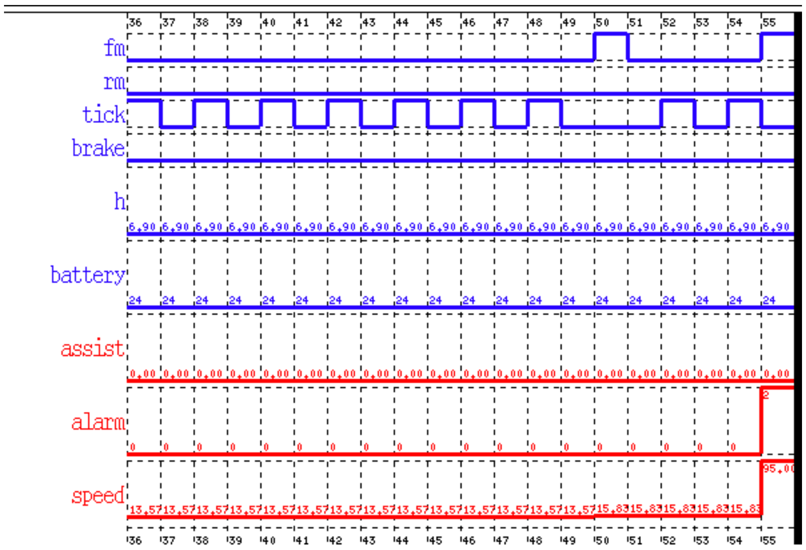
Here we will counter of property to check if it gives false or not. So, we will check in the update of RR if it does not show RR speed the give false, same we use for FR, if both does not work then we will give true that verified correctly.

## Task L6



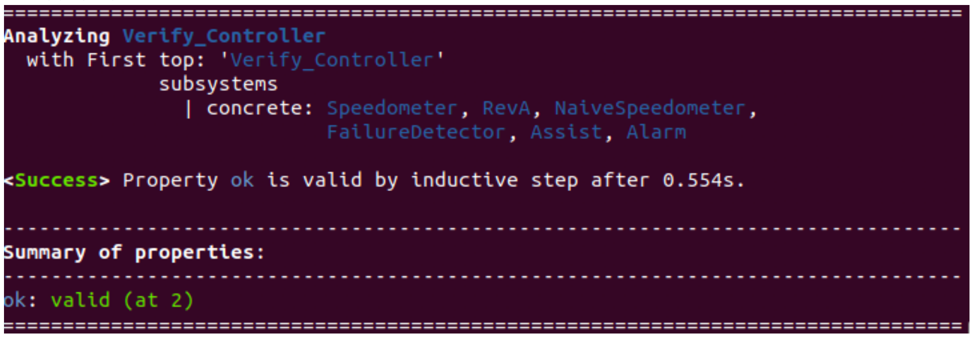
Here we have used the condition in the project manual. The next formula assist will be depending on the torque force. The condition where torque be 0 is when bike is off, alarm is 2, speed being more than 7 and brakes are active. Once we have the torque value that does not satisfy the any condition. Once we have torque value that is not 0, then we check for battery level, if it is more than 20 then the human force is output else it is half.

Task L7



Here it’s the combination of all the nodes in the past. All nodes are just called in this node. Here the idea is just for verification while using Kind2.

*Verify if speed > 7 then assist is 0*



```
================================================================================
Analyzing Verify_Controller
  with First top: 'Verify_Controller'
             subsystems
               | concrete: Speedometer, RevA, NaiveSpeedometer,
                           FailureDetector, Assist, Alarm

<Success> Property ok is valid by inductive step after 0.554s.

--------------------------------------------------------------------------------
Summary of properties:
--------------------------------------------------------------------------------
ok: valid (at 2)
================================================================================
```

To verify the condition to check in a condition where speed is more than 7 and the assist force is not 0 then say that it does not verify. Otherwise we can verify that all is verified correctly.