Université Libre de Bruxelles

INFO-F410 - Embedded Systems Design

Electric-Bike Project (Lustre)

Student (MA1 Computer Science)
SWIRYDOWICZ Szymon
000477108

Lecturer / TAs
RASKIN Jean-François
BALACHANDER Mrudula
CHAKRABORTY Debraj
KHALIMOV Ayrat

April - May 2022

Contents

Chapter 1

Lustre

1.1 Introduction

This part of the project focuses on Lustre, which is "a formally defined, declarative, and synchronous dataflow programming language for programming reactive systems".[1]

As the project description required adding parts of the code to this report, the full code can be found in section 1.7. Explicit comments were added to clearly separate the five given tasks. Additionally, the corresponding `main.lus` code file has been joined with this report.

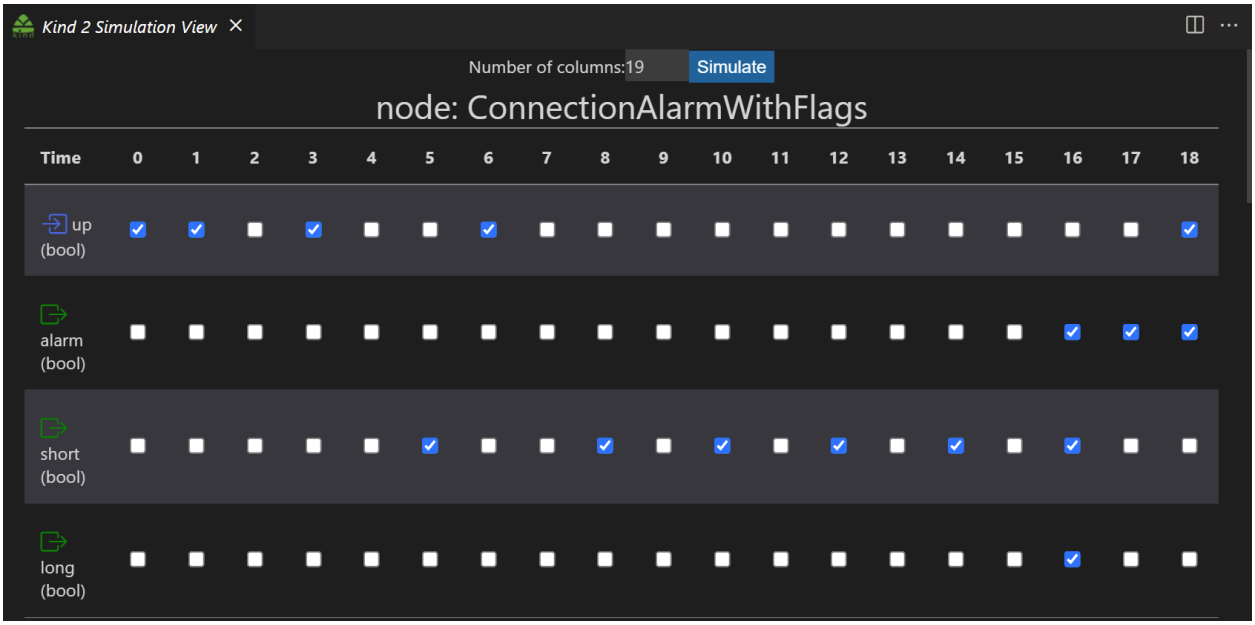
The verification part was executed using the official Kind 2 VS Code extension[2], which allows not only to validate stated properties, but also allows to simulate a node's behavior. The requested verification has been added to section 1.8.

1.2 Task L1

1.2.1 Diagrams

The following diagrams were obtained through the Kind 2 Extension's (cfr. Introduction) Simulation View.

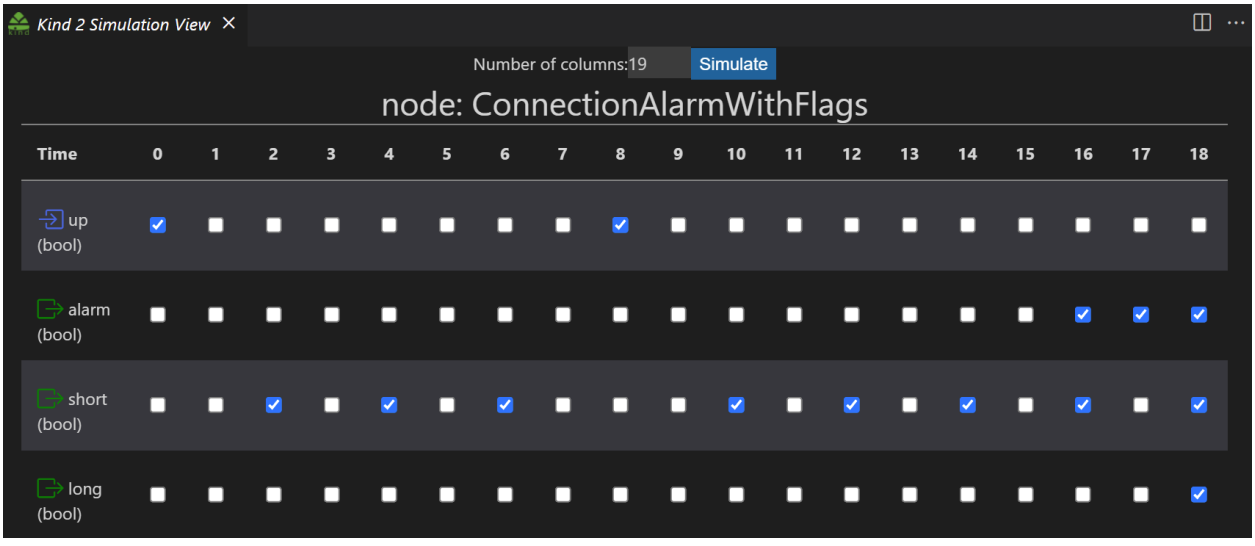
[1] Source: https://en.wikipedia.org/wiki/Lustre_(programming_language)
[2] Kind2 VSC extension: https://marketplace.visualstudio.com/items?itemName=kind2-mc.vscode-kind2



Task L1: 1 Long drop alarm.

By making `UP` true at times 0, 1, 3, 6 and 18, the system raises the SHORT flag at times 5, 8, 10, 12, 14 and 16, the LONG flag only at time 16, and in consequence of the long drop, the ALARM flag also rises at times 16, 17 and 18. This corresponds precisely, in inputs and outputs, to the diagram showcased in the Lustre project description.

Additionally, the following diagram has been added where the ALARM flag raises because of 7 short drops.



Task L1: 7 Short drops alarm.

Note: Those diagrams could have been obtained using Luciole's sim2chro tool. However, for unknown reasons, that tool only raised errors on my machine. In consequence, I prefered to simply include the above simulation view as it implements the same behaviour (despite looking a bit less professional).

1.2.2 LTL

We have to translate "once the alarm is set, it stays set forever" into LTL.

I would write it in LTL using the Globally (Always) operator and the implication operator (response) as follows:

```
G ( alarm -> X alarm )
```

Which in other words means that no matter when, if alarm is true at one position, it also has to be true at the next position. Since the alarm in that next position will also be true and the global implication will still hold, basically the alarm will continue forever.

1.2.3 Verification

The property "*once the alarm is set, it stays set forever*", is described in the following way:

```
--%PROPERTY (false->pre(alarm) = true) => (alarm = true);
```

In other words, it ensures that if the alarm has been set at time $(t-1)$, it must also be set at time t. And inductively, since alarm at t must be `true`, the alarm at $(t+1)$ will also have to be `true`.

1.3 Task L2

1.3.1 Verification

Two properties had to be verified.

First, "whenever the cyclists brakes hard, the clinch should happen". This is verified using the following two lines:

```
--%PROPERTY (brk > BRK_SOFT and ConnectionAlarm(up)) => (clinch = true);
--%PROPERTY (brk > BRK_SOFT and not ConnectionAlarm(up)) => (clinch = true);
```

It simply states that when the brake is hard, a clinch must happen. The "`(not) ConnectionAlarm(up)`" part is clearly not necessary, but it has been added to better put in contrast the following two properties:

```
--%PROPERTY (brk > 0.0 and brk <= BRK_SOFT and not ConnectionAlarm(up)) => (clinch = false);
--%PROPERTY (brk > 0.0 and brk <= BRK_SOFT and ConnectionAlarm(up)) => (clinch = true);
```

that make sure that soft breaking is true only when the connection alarm is also true.

Next, "whenever the clinch happens, BRK is not zero".

```
--%PROPERTY (clinch = true) => (brk <> 0.0);
```

Which simply tells that if clinch happens (is true) then it implies that breaking is not zero.

1.4 Task L3

1.4.1 Verification

Although, not demanded, the following two properties ensure that the node works as expected.

```
--%PROPERTY (not up) => (brk_rcvd = noise);
--%PROPERTY (up) => (brk_rcvd = brk);
```

1.5 Task L4

This part was implemented by first creating two Booleans: `assist_allowed` and `regen_allowed`, which are respectively for checking assistance and regenerative braking. After a series of conditionals setting those true or false, we determine the assistance level. If both of them happen to be true, then the regenerative braking will be prioritized, except in the case where the received `brk_rcvd` is set to 0.0, in which case assistance goes first.

Verification Different properties were added to test the correctness of the node. Those can be found in section 1.7.

1.6 Task L5

1.6.1 Verification

If the connection alarm is true and the braking happens, brake-cable clinch must happen (i.e., CLINCH must be true);

```
--%PROPERTY ((alarm = true) and (brk > 0.0)) => (clinch = true);
```

When the connection alarm is true, the regenerative braking cannot happen; (reminder: regenerative braking happens when assist level if below zero)

```
--%PROPERTY (alarm = true) => (asst_lvl >= 0.0);
```

Regenerative braking occurs only if the braking is requested by the human;

```
--%PROPERTY (asst_lvl < 0.0) => (brk > 0.0);
```

When the speed is above 7 m/s, the assistance is negative or zero;

```
--%PROPERTY (speed > MAX_SPEED) => (asst_lvl <= 0.0);
```

The assistance level is always equal to one of the following values: 0, -BRK, HUMAN.

```
--%PROPERTY (asst_lvl = 0.0) or (asst_lvl = -brk) or (asst_lvl = human);
```

If the braking and assisting are both requested at the same time and the connection is up and there is no alarm, then the motor should engage the regenerative braking.

```
--%PROPERTY ((brk > 0.0) and (alarm = false) and (up = true)) => (asst_lvl < 0.0);
```

1.7 Code

```
--------------------------------------------------------------------
-- Constants:
const
  SHORT_DROP_NUM:int = 2;
  LONG_DROP_NUM:int = 10;
  MAX_NOF_SHORT_DROPS:int = 7;
  MAX_SPEED:real = 7.0;
  BRK_SOFT:real = 1.0;

--------------------------------------------------------------------

----------------------
-- ::: TASK L1 ::: ---
----------------------
-- private to the ShortDrop node
-- counts number of 'time' units since UP is false
node ShortCounter(up:bool) returns (count:int);
let
  count = ( if up then 0
            else if not up then (0 -> pre(count) + 1)
            else 0 -> pre(count)
  );
tel

node ShortDrop(up:bool) returns (drop:bool);
var can_short:bool;
let
  can_short = ShortCounter(up) mod SHORT_DROP_NUM = 0;
  -- we check if i and i-1 of UP are false,
  -- if so, set the shortDrop to true iff we didn't have a short drop before (modulo)
  drop = ( if not up and not (true->pre(up)) then can_short
            else false
  );
tel

-- counts the number of ShortDrops encountered during program execution
node ShortDropCounter(up:bool) returns (count:int);
let
  count = ( if ShortDrop(up) then (0->pre(count) + 1)
            else 0->pre(count)
  );
tel

node LongDropCounter(up:bool) returns (count:int);
let
  count = ( if up then 0
            else if not up then (0->pre(count) + 1)
            else 0->pre(count)
  );
tel

node ConnectionAlarm(up:bool) returns (alarm:bool);
let
  alarm = ( if false->pre(alarm) = true then true -- ensures alarm stays ON once set
            else if ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS then true -- looks for 7 short
                drops of 2 units
```

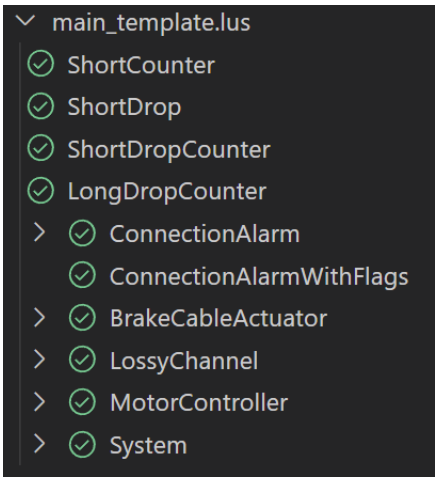
```
          else if LongDropCounter(up) >= LONG_DROP_NUM then true -- looks for 1 long drop of 10
              units
          else false
  );
  --%PROPERTY (false->pre(alarm) = true) => (alarm = true);
  --%PROPERTY (ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) => (alarm = true);
  --%PROPERTY (LongDropCounter(up) >= LONG_DROP_NUM) => (alarm = true);
tel

-- COPY OF THE ConnectionAlarm FOR THE DIAGRAM TASK (+= short/long flags)
node ConnectionAlarmWithFlags(up:bool) returns (alarm:bool; short:bool; long:bool);
let
  alarm = ( if false->pre(alarm) = true then true
          else if ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS then true
          else if LongDropCounter(up) >= LONG_DROP_NUM then true
          else false
  );
  short = ShortDrop(up);
  long = (not up) and LongDropCounter(up) mod LONG_DROP_NUM = 0;
tel

----------------------
-- ::: TASK L2 ::: ---
----------------------
node BrakeCableActuator(brk:real; up:bool) returns (clinch:bool);
let
  clinch = ( if ConnectionAlarm(up) and (brk <= BRK_SOFT and brk > real(0.0)) then true -- SOFT
      BREAK + ALARM
             else if brk > BRK_SOFT then true -- HARD BREAK
             else false
  );
  --%PROPERTY (clinch = true) => (brk <> 0.0);
  --%PROPERTY (brk > 0.0 and brk <= BRK_SOFT and not ConnectionAlarm(up)) => (clinch = false);
  --%PROPERTY (brk > 0.0 and brk <= BRK_SOFT and ConnectionAlarm(up)) => (clinch = true);
  --%PROPERTY (brk > BRK_SOFT and ConnectionAlarm(up)) => (clinch = true);
  --%PROPERTY (brk > BRK_SOFT and not ConnectionAlarm(up)) => (clinch = true);
tel

----------------------
-- ::: TASK L3 ::: ---
----------------------
node LossyChannel(noise,brk:real; up:bool) returns (brk_rcvd:real);
let
  brk_rcvd = (if up then brk else noise);
  --%PROPERTY (not up) => (brk_rcvd = noise);
  --%PROPERTY (up) => (brk_rcvd = brk);
tel

----------------------
-- ::: TASK L4 ::: ---
----------------------
node MotorController(brk_rcvd:real; up:bool; human,speed,battery:real) returns (asst_lvl:real);
var
  assist_allowed:bool;
  regen_allowed:bool;
let
  -- * assumption: if both assist and regen are true then regen is prioritised (exception:
      brk_rcvd = 0.0)
  assist_allowed = (if ConnectionAlarm(up) then false
```

```
                  else if speed > MAX_SPEED then false
                  else if battery = 0.0 then false
                  else true
  );
  regen_allowed = ( if ConnectionAlarm(up) then false
                  else if not up then false
                  else true
  );
  asst_lvl = ( if regen_allowed and assist_allowed and brk_rcvd = 0.0 then human
               else if regen_allowed then -brk_rcvd
               else if assist_allowed then human
               else 0.0
  );
  --%PROPERTY (not up) => (regen_allowed = false);
  --%PROPERTY (speed > MAX_SPEED) => (assist_allowed = false);
  --%PROPERTY (battery = 0.0) => (assist_allowed = false);
  --%PROPERTY ((regen_allowed = true) and (assist_allowed = false)) => (asst_lvl = -brk_rcvd);
  --%PROPERTY ((regen_allowed = false) and (assist_allowed = true)) => (asst_lvl = human);
  --%PROPERTY ((brk_rcvd = 0.0) and (assist_allowed = true) and ((regen_allowed = true))) =>
      (asst_lvl = human);
  --%PROPERTY ((brk_rcvd <> 0.0) and (assist_allowed = true) and ((regen_allowed = true))) =>
      (asst_lvl = -brk_rcvd);
tel

----------------------
-- ::: TASK L5 ::: ---
----------------------
node System(up:bool; noise,brk,human,speed,battery:real) returns (clinch:bool; asst_lvl:real);
var
  alarm:bool;
  brk_rcvd:real;
  temp_assist:real;
let
  brk_rcvd = LossyChannel(noise,brk,up);
  alarm = ConnectionAlarm(up);

  clinch = (if alarm and (brk_rcvd > 0.0) then true
           else BrakeCableActuator(brk,up)
  );

  temp_assist = MotorController(brk_rcvd,up,human,speed,battery);
  asst_lvl = ( if alarm and (temp_assist < 0.0) then 0.0
               else if (temp_assist < 0.0) and (brk <= 0.0) then 0.0
               else if (speed > MAX_SPEED) and (temp_assist > 0.0) then 0.0
               else if not (temp_assist = 0.0) and not (temp_assist = -brk) and not (temp_assist =
                   human) then 0.0
               else if (brk > 0.0) and (temp_assist > 0.0) and up and (not alarm) then -brk
               else temp_assist
  );

  --%MAIN;
  --%PROPERTY ((alarm = true) and (brk > 0.0)) => (clinch = true);
  --%PROPERTY (alarm = true) => (asst_lvl >= 0.0);
  --%PROPERTY (asst_lvl < 0.0) => (brk > 0.0);
  --%PROPERTY (speed > MAX_SPEED) => (asst_lvl <= 0.0);
  --%PROPERTY (asst_lvl = 0.0) or (asst_lvl = -brk) or (asst_lvl = human);
  --%PROPERTY ((brk > 0.0) and (alarm = false) and (up = true)) => (asst_lvl < 0.0);
tel
```

1.8 Verification - outputs

1.8.1 Summary

All the properties present in (the code of) section 1.7 were tested successfully by Kind 2.



Kind 2 verification of all the properties

1.8.2 Task L1

```
Analyzing ConnectionAlarm
  with First top: 'ConnectionAlarm'
            subsystems
              | concrete: ShortDropCounter, ShortDrop, ShortCounter, LongDropCounter

<Success> Property ((LongDropCounter(up) >= LONG_DROP_NUM) => (alarm = true)) is valid by property
    directed reachability after 0.082s.

<Success> Property ((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) => (alarm = true)) is valid by
    property directed reachability after 0.082s.

<Success> Property ((false -> ((pre alarm) = true)) => (alarm = true)) is valid by property
    directed reachability after 0.082s.
```

```
Summary of properties:

((LongDropCounter(up) >= LONG_DROP_NUM) => (alarm = true)): valid (at 1)
((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) => (alarm = true)): valid (at 1)
((false -> ((pre alarm) = true)) => (alarm = true)): valid (at 1)
```

1.8.3 Task L2

```
Analyzing BrakeCableActuator
  with First top: 'BrakeCableActuator'
            subsystems
              | concrete: ShortDropCounter, ShortDrop, ShortCounter, LongDropCounter,
                  ConnectionAlarm
```

```
<Success> Property (((brk > BRK_SOFT) and (not ConnectionAlarm(up))) => (clinch = true)) is valid
    by property directed reachability after 0.083s.

<Success> Property (((brk > BRK_SOFT) and ConnectionAlarm(up)) => (clinch = true)) is valid by
    property directed reachability after 0.083s.

<Success> Property ((((brk > 0.0) and (brk <= BRK_SOFT)) and ConnectionAlarm(up)) =>
  (clinch = true)) is valid by property directed reachability after 0.083s.

<Success> Property ((((brk > 0.0) and (brk <= BRK_SOFT)) and (not ConnectionAlarm(up))) =>
  (clinch = false)) is valid by property directed reachability after 0.083s.

<Success> Property ((clinch = true) => (brk <> 0.0)) is valid by property directed reachability
    after 0.083s.

<Success> Property ConnectionAlarm[l78c18].((false -> ((pre alarm) = true)) => (alarm = true)) is
    valid by property directed reachability after 0.083s.

<Success> Property ConnectionAlarm[l78c18].((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) =>
    (alarm = true)) is valid by property directed reachability after 0.083s.

<Success> Property ConnectionAlarm[l78c18].((LongDropCounter(up) >= LONG_DROP_NUM) => (alarm =
    true)) is valid by property directed reachability after 0.083s.
```

```
Summary of properties:

(((brk > BRK_SOFT) and (not ConnectionAlarm(up))) => (clinch = true)): valid (at 1)
(((brk > BRK_SOFT) and ConnectionAlarm(up)) => (clinch = true)): valid (at 1)
((((brk > 0.0) and (brk <= BRK_SOFT)) and ConnectionAlarm(up)) =>
  (clinch = true)): valid (at 1)
((((brk > 0.0) and (brk <= BRK_SOFT)) and (not ConnectionAlarm(up))) =>
  (clinch = false)): valid (at 1)
((clinch = true) => (brk <> 0.0)): valid (at 1)
ConnectionAlarm[l78c18].((false -> ((pre alarm) = true)) => (alarm = true)): valid (at 1)
ConnectionAlarm[l78c18].((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) => (alarm = true)): valid
    (at 1)
ConnectionAlarm[l78c18].((LongDropCounter(up) >= LONG_DROP_NUM) => (alarm = true)): valid (at 1)
```

1.8.4 Task L5

```
Analyzing System
  with First top: 'System'
            subsystems
              | concrete: ShortDropCounter, ShortDrop, ShortCounter, MotorController,
                  LossyChannel, LongDropCounter,
                        ConnectionAlarm, BrakeCableActuator

<Success> Property (((asst_lvl = 0.0) or (asst_lvl = (- brk))) or (asst_lvl = human)) is valid by
    inductive step after 0.144s.

<Success> Property ((speed > MAX_SPEED) => (asst_lvl <= 0.0)) is valid by inductive step after
    0.144s.

<Success> Property ((asst_lvl < 0.0) => (brk > 0.0)) is valid by inductive step after 0.144s.
```

<Success> Property ((alarm = true) => (asst_lvl >= 0.0)) is valid by inductive step after 0.144s.

<Success> Property ConnectionAlarm[l135c11].((false -> ((pre alarm) = true)) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property ConnectionAlarm[l135c11].((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property ConnectionAlarm[l135c11].((LongDropCounter(up) >= LONG_DROP_NUM) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property LossyChannel[l134c14].((not up) => (brk_rcvd = noise)) is valid by inductive step after 0.144s.

<Success> Property LossyChannel[l134c14].(up => (brk_rcvd = brk)) is valid by inductive step after 0.144s.

<Success> Property BrakeCableActuator[l138c18].ConnectionAlarm[l78c18].((LongDropCounter(up) >= LONG_DROP_NUM) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property BrakeCableActuator[l138c18].ConnectionAlarm[l78c18].((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property BrakeCableActuator[l138c18].ConnectionAlarm[l78c18].((false -> ((pre alarm) = true)) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property BrakeCableActuator[l138c18].((clinch = true) => (brk <> 0.0)) is valid by inductive step after 0.144s.

<Success> Property BrakeCableActuator[l138c18].((((brk > 0.0) and (brk <= BRK_SOFT)) and (not ConnectionAlarm(up))) =>
  (clinch = false)) is valid by inductive step after 0.144s.

<Success> Property BrakeCableActuator[l138c18].((((brk > 0.0) and (brk <= BRK_SOFT)) and ConnectionAlarm(up)) =>
  (clinch = true)) is valid by inductive step after 0.144s.

<Success> Property BrakeCableActuator[l138c18].(((brk > BRK_SOFT) and ConnectionAlarm(up)) => (clinch = true)) is valid by inductive step after 0.144s.

<Success> Property BrakeCableActuator[l138c18].(((brk > BRK_SOFT) and (not ConnectionAlarm(up))) => (clinch = true)) is valid by inductive step after 0.144s.

<Success> Property MotorController[l141c17].ConnectionAlarm[l104c24].((LongDropCounter(up) >= LONG_DROP_NUM) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property MotorController[l141c17].ConnectionAlarm[l104c24].((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property MotorController[l141c17].ConnectionAlarm[l104c24].((false -> ((pre alarm) = true)) => (alarm = true)) is valid by inductive step after 0.144s.

<Success> Property MotorController[l141c17].((not up) => (regen_allowed = false)) is valid by inductive step after 0.144s.

<Success> Property MotorController[l141c17].((speed > MAX_SPEED) => (assist_allowed = false)) is valid by inductive step after 0.144s.

```
<Success> Property MotorController[l141c17].((battery = 0.0) => (assist_allowed = false)) is valid
    by inductive step after 0.144s.

<Success> Property MotorController[l141c17].(((regen_allowed = true) and (assist_allowed = false))
    =>
  (asst_lvl = (- brk_rcvd))) is valid by inductive step after 0.144s.

<Success> Property MotorController[l141c17].(((regen_allowed = false) and (assist_allowed = true))
    => (asst_lvl = human)) is valid by inductive step after 0.144s.

<Success> Property MotorController[l141c17].((((brk_rcvd = 0.0) and (assist_allowed = true)) and
    (regen_allowed = true)) =>
  (asst_lvl = human)) is valid by inductive step after 0.144s.

<Success> Property ((((brk > 0.0) and (alarm = false)) and (up = true)) => (asst_lvl < 0.0)) is
    valid by property directed reachability after 4.250s.

<Success> Property (((alarm = true) and (brk > 0.0)) => (clinch = true)) is valid by property
    directed reachability after 4.250s.
```

```
Summary of properties:

((((brk > 0.0) and (alarm = false)) and (up = true)) => (asst_lvl < 0.0)): valid (at 1)
(((asst_lvl = 0.0) or (asst_lvl = (- brk))) or (asst_lvl = human)): valid (at 1)
((speed > MAX_SPEED) => (asst_lvl <= 0.0)): valid (at 1)
((asst_lvl < 0.0) => (brk > 0.0)): valid (at 1)
((alarm = true) => (asst_lvl >= 0.0)): valid (at 1)
(((alarm = true) and (brk > 0.0)) => (clinch = true)): valid (at 1)
ConnectionAlarm[l135c11].((false -> ((pre alarm) = true)) => (alarm = true)): valid (at 1)
ConnectionAlarm[l135c11].((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS) => (alarm = true)): valid
    (at 1)
ConnectionAlarm[l135c11].((LongDropCounter(up) >= LONG_DROP_NUM) => (alarm = true)): valid (at 1)
LossyChannel[l134c14].((not up) => (brk_rcvd = noise)): valid (at 1)
LossyChannel[l134c14].(up => (brk_rcvd = brk)): valid (at 1)
BrakeCableActuator[l138c18].ConnectionAlarm[l78c18].((LongDropCounter(up) >= LONG_DROP_NUM) =>
    (alarm = true)): valid (at 1)
BrakeCableActuator[l138c18].ConnectionAlarm[l78c18].((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS)
    => (alarm = true)): valid (at 1)
BrakeCableActuator[l138c18].ConnectionAlarm[l78c18].((false -> ((pre alarm) = true)) => (alarm =
    true)): valid (at 1)
BrakeCableActuator[l138c18].((clinch = true) => (brk <> 0.0)): valid (at 1)
BrakeCableActuator[l138c18].((((brk > 0.0) and (brk <= BRK_SOFT)) and (not ConnectionAlarm(up))) =>
  (clinch = false)): valid (at 1)
BrakeCableActuator[l138c18].((((brk > 0.0) and (brk <= BRK_SOFT)) and ConnectionAlarm(up)) =>
  (clinch = true)): valid (at 1)
BrakeCableActuator[l138c18].(((brk > BRK_SOFT) and ConnectionAlarm(up)) => (clinch = true)): valid
    (at 1)
BrakeCableActuator[l138c18].(((brk > BRK_SOFT) and (not ConnectionAlarm(up))) => (clinch = true)):
    valid (at 1)
MotorController[l141c17].ConnectionAlarm[l104c24].((LongDropCounter(up) >= LONG_DROP_NUM) =>
    (alarm = true)): valid (at 1)
MotorController[l141c17].ConnectionAlarm[l104c24].((ShortDropCounter(up) >= MAX_NOF_SHORT_DROPS)
    => (alarm = true)): valid (at 1)
MotorController[l141c17].ConnectionAlarm[l104c24].((false -> ((pre alarm) = true)) => (alarm =
    true)): valid (at 1)
MotorController[l141c17].((not up) => (regen_allowed = false)): valid (at 1)
MotorController[l141c17].((speed > MAX_SPEED) => (assist_allowed = false)): valid (at 1)
MotorController[l141c17].((battery = 0.0) => (assist_allowed = false)): valid (at 1)
```

```
MotorController[l141c17].(((regen_allowed = true) and (assist_allowed = false)) =>
  (asst_lvl = (- brk_rcvd))): valid (at 1)
MotorController[l141c17].(((regen_allowed = false) and (assist_allowed = true)) => (asst_lvl =
    human)): valid (at 1)
MotorController[l141c17].((((brk_rcvd = 0.0) and (assist_allowed = true)) and (regen_allowed =
    true)) =>
  (asst_lvl = human)): valid (at 1)
```
