INDEX

1. Test example
   1.1. The domain
   1.2. Specification rules
   1.3. Program report
1. Test example

For testing purposes we have developed a simple program that simulates the behavior of different trains in a crossroad. This program can directly dump the data into a file or send it to the database as shown in figure 2.

1.1. The domain.

The cross-rail problem is shown in the following figure 13 for two rails. On each rail there is a traffic light to regulate the access so that only one train can be crossing at the same time. Therefore, while a train starts crossing the intersection, the traffic lights for all the other rails have to be red.

![Figure 13. Cross-rail example.](image)

The status for the train on each rail and the transitions are described in figure 14. The initial state is that the train is away from the intersection. Every time a train arrives will start crossing if the traffic light is in green, otherwise will be waiting until it turns green. Once it finishes crossing, it will start reporting that it has finished to cross the intersection.

![Figure 14. Status transition of a train for the cross-rail example.](image)

Each rail is considered as an independent process. The log data consists of a set of log entries each one with a name that corresponds to the name of the rail that is writing the
information and the data includes the status of the train in the rail (away, waiting, crossing or reporting), status of the traffic light for that line (red or green) and the velocity of the train.

For example, one log entry could be:

```
train1 38 speed float 20.5 semaphore string "green" status string "crossing"
```

where:
- `train1` -> name of the “event” in this case is the name of the rail also.
- `38` -> time stamp.
- `20.5` -> speed of the train.
- “green” -> color of the traffic light.
- “crossing” -> state of the train in the rail.

Next section describes the specification rules to be tested in the logged data and section 4.2 shows the output for the tests.

### 1.2. Specification rules

The following list shows the specification rules. Some of them can be defined in a shorter way but we wanted to test the most features of the logic so we tried to use different expressions.

The specifications we want to check are defined in order to see if the system works according to what we described before. For example, to check that two trains never cross at the same time.

```plaintext
# ###############################################
# ################ TRAIN 1 RULES ################
# ###############################################
# intervals while the train 1 is away.
away1 = [train1.status == "away"];

# ******* RULE 1 *******
#----------------------------------------------------------------------
# the speed of the train while is away must be greater
# than 60 and less than 61
#----------------------------------------------------------------------
speed1 = train1.speed;
forall it2_1: away1 {
    it2_1 always ((speed1 > 60) && (speed1 < 61))
};

# ******* RULE 2 (DSL 5, T5) *******
#----------------------------------------------------------------------
# Train 1 and 2 can't cross at the same time
#----------------------------------------------------------------------
cross2 = [train2.status == "crossing"];
forall c1 : [train1.status == "crossing"] { 
    forall c2 : cross2 { 
        !(c1 intersects c2)
    }
};

# OR

cross1 = [train1.status == "crossing"];
forall c1 : cross1 { 
    forall c2 : cross2 { 
        (end(c1) is_before [,] start(c2)) ||
        (end(c2) is_before [,] start(c1))
    }
};
```
# RULE 3
After crossing, the train has to start reporting

\[
\text{report1} = \text{[train1.status == "reporting"]};
\]

\[
\text{forall c1 : cross1 } \{
    \text{exists c2 : report1 } \{
    \text{end(c1) == start(c2)}
    \}
\};
\]

# RULE 4
After crossing, the train has to be reporting

\[
\text{forall c1 : cross1 } \{
    \text{c1 after report1}
\};
\]

# RULE 5
After waiting, the train has to be crossing

\[
\text{wait1} = \text{[train1.status == "waiting"]};
\]

\[
\text{forall c1 : wait1 } \{
    \text{c1 after cross1}
\};
\]

In this case the result if false because of the last interval.

\[
\text{forall c2 : wait2 } \{
    \text{c2 after cross2}
\};
\]

# RULE 6
Before reporting, the train was crossing

\[
\text{forall c1 : report1 } \{
    \text{c1 before cross1}
\};
\]

# RULE 7
1 second after crossing, the report must be finished.

\[
\text{forall c1 : cross1 } \{
    \text{exists c2 : report1 } \{
    \text{end(c1) -> end(c1)-> 1 include end(c2)}
    \}
\};
\]

# RULE 7
Within 4 and 6 milliseconds, the report must be finished.

\[
\text{forall c1 : cross1 } \{
    \text{exists c2 : report1 } \{
    \text{end(c1)-> 0.004 -> end(c1)-> 0.006 include end(c2)}
    \}
\};
\]
forall c1 : cross1 {
    exists c2 : report1 {
        (time(end(c1)) <= time(start(c2))) &&
        (time(end(c1)) <= time(start(c2))
    }
};
# ----------------------------------------------
# Basic examples
# ----------------------------------------------
# ******* RULE (DSL 8) *******
# always the speed have to be positive.
# ----------------------------------------------

-> always (train1.speed >= 0);
# OR

speed1 = train1.speed;
-> always (speed1 >= 0);
# but sometimes is waiting so, this should be false.
-> always (speed1 > 0);

# ******* RULE (DSL 14) *******
# Allways, the duration of crossing has to be greater than 2 msec.
# and less than 15 msec.
# ----------------------------------------------
forall a_1:cross1 {
    (duration(a_1) > 0.002) &&
    (duration(a_1) < 0.015)
};

# ******* RULE (DSL 17, 20, 22, 24 and 25) *******
# the status of the train is never unknown.
# ----------------------------------------------
unknown = [train1.status == "unknown"];
cardinal( unknown ) == 0;

# ******* RULE (DSL 27, 28) *******
# if the train is not waiting, it has to be moving (speed >0).
# ----------------------------------------------
not_waiting1 = - > -- wait1;
moving = [train1.speed >0];
empty = not_waiting1 -- moving;
cardinal(empty) == 0;
empty = moving -- not_waiting1;
cardinal(empty) == 0;
# Relations between actions
# ----------------------------------------------

# ******* RULE (DSL 12) *******
# After the train is away, it has to be crossing or waiting
# ----------------------------------------------
after_away1 = wait1 union cross1;
forall a_1:away1 {
    exists a2_1:after_away1 {
        a_1 after a2_1
    }
};
forall a_1:away1 {
    a_1 after after_away1
};

# RULE T1, T4
# between 4 and 6 milliseconds
# after crossing, the report must be finished.
forall c1 : cross1 {
    exists c2 : report1 {
        time(end(c1)) > (time(end(c2)) - 0.006) &&
        time(end(c1)) < (time(end(c2)) - 0.004)
    }
};

# RULE
# The train can't go from away to report without crossing
away_report1 = end(away1) -> start(report1);
forall a2_1:away_report1 {
    exists a1_1:cross1 {
        a2_1 include a1_1
    }
};

# NOTE: An interval
# include the start event but not the end. That explains why
# the first print returns true and the second false.
a2_1 include start(a2_1);
a2_1 include end(a2_1);

# RULE (DSL 9)
# if train 1 is not waiting, then it has to be crossing, away
# or reporting.
cross_away_rep1 = cross1 union away1 union report1;
not_wait1 = - > -- wait1;
cardinal(cross_away_rep1 -- not_wait1)== 0;

# OR BETTER
cross_away_rep1 = cross1 | away1 | report1;
not_wait1 = - > -- wait1;
cardinal(cross_away_rep1 -- not_wait1)== 0;

# RULE
# train 1 can be either waiting or in one of these states:
cross_away_rep1 = cross1 | away1 | report1;
not_wait1 = - > -- wait1;
(cardinal(cross_away_rep1 -- not_wait1) == 0) &&
(cardinal(not_wait1 -- cross_away_rep1) == 0);

# OR
# PROBLEMS WITH THE NULL INTERVALS!!
forall a_1:wait1 {
    forall a2_1:cross_away_rep1 {
        !(a2_1 intersects a_1)
    }
};
# RULE 6

1 millisecond before start reporting, the semaphore must be in green.

\[
\text{one_msec_bef} = 0.001 \overset{\sim}{\leftarrow} \text{start(report1)};
\]

\[
\text{trail_in_green} = [\text{train1.semaphore} == \text{"green"}];
\]

\[
\text{forall e1_1:one_msec_bef} \{\exists a1_1:trail_in_green \{ \text{a1_1 include e1_1} \} \};
\]

OR

\[
\text{train1semaphore} = \text{train1.semaphore};
\]

\[
\text{forall a1_1:report1} \{ 0.001 \overset{\sim}{\leftarrow} \text{start(a1_1)} \rightarrow \text{start(a1_1) begin} \}
\]

\[
\text{train1semaphore} == \text{"green"}
\]

# RULE 6

1 millisecond before start reporting, the semaphore must be in green and must keep in red at least until start reporting.

\[
\text{one_msec_bef_int} = 0.001 \overset{\sim}{\leftarrow} \text{start(report1)} \rightarrow \text{start(report1)};
\]

\[
\text{trail_in_green} = [\text{train1.semaphore} == \text{"green"}];
\]

\[
\text{forall a2_1:one_msec_bef_int} \{\exists a1_1:trail_in_green \{ \text{a1_1 include a2_1} \} \};
\]

OR A SOLUTION VERY TIME CONSUMING:

\[
\text{forall a1_1:report1} \{ 0.001 \overset{\sim}{\leftarrow} \text{start(a1_1)} \rightarrow \text{start(a1_1) always} \}
\]

\[
\text{train1semaphore} == \text{"green"}
\]

# RULE 6

only one train can be waiting on the first trail this is, between two crossing only one arrival is possible.

\[
\text{btw_cross1} = \text{end(cross1)} \rightarrow \text{end(cross1)};
\]

\[
\text{forall a1_1:btw_cross1} \{(\text{cardinal(tmp:wait1 st start(tmp) inside a1_1) == 1)} \}
\]

# RULE 6 (GENERAL CASE)

only N trains can be waiting on the first trail this is, between two crossing only one arrival is possible.

\[
\text{N} = 10;
\]

\[
\text{before_cross1} = \leftarrow \text{start(cross1)};
\]

\[
\text{forall a1_1:before_cross1} \{(\text{cardinal(tmp:wait1 st start(tmp) inside a1_1) <= \text{N + cardinal(tmp:cross1 st start(tmp) inside a1_1)})} \}
\]

# RULE

in a interval of T msec the train 1 can't change the speed in more than max.

\[
\text{T} = 0.001;
\]

\[
\text{max} = 70;
\]

\[
\text{speed1} = \text{train1.speed};
\]
forall a1_1: train1. \(\rightarrow\) train1. \(\rightarrow\) T { 
    a1_1 always (((speed1 - speed1[time(start(a1_1))]) < max) && 
    ((speed1 - speed1[time(start(a1_1))]) > (0 - max)))
};

#=======================================================
#
#   END OF FILE
#
#=======================================================

1.3. Program report

Next list shows the output of the program for the specification rules defined in the last section.

-- Specification:
forall it2_1: [away1] { 
    it2_1 always ((speed1 > 60) && (speed1 < 61));
}

Is TRUE

----

-- Specification:
forall c1 : [train1.status == "crossing"] { 
    forall c2 : [cross2] { 
        !(c1 intersects 
    .

Is TRUE

----

-- Specification:
forall c1 : [cross1] { 
    forall c2 : [cross2] { 
        ((end(c1)) is_before [,] (start(c2)))
    .

Is TRUE

----

-- Specification:
forall c1 : [cross1] { 
    exists c2 : [report1] { 
        end(c1) == start(c2);
    }
}

Is TRUE

----

-- Specification:
forall c1 : [cross1] { 
    c1 after [report1];
}

Is TRUE

----
-- Specification:
forall c1 : [wait1] {
  c1 after [cross1];
}

Is TRUE

-- Specification:
forall c2 : [wait2] {
  c2 after [cross2];
}

is FALSE because:

In the c2 interval:
Intervalvar=
  Start: sec = 2.14748e+09 usec = 0
  End: sec = 2.14748e+09 usec = 0
forall becomes false

-- Specification:
forall c1 : [report1] {
  c1 before [cross1];
}

Is TRUE

-- Specification:
forall c1 : [cross1] {
  exists c2 : [report1] {
    (end(c1) -> end(c1)~> 1) include end(c2)
  ...
}

is FALSE because:

In the c1 interval:
Intervalvar=
  Start: sec = 248 usec = 668000
  End: sec = 248 usec = 678000
forall becomes false
There is no c2 event holding the exists condition

-- Specification:
forall c1 : [cross1] {
  exists c2 : [report1] {
    (end(c1)~> 0.004 -> end(c1)~> 0.006) in
  ...
}

is FALSE because:

In the c1 interval:
Intervalvar=
  Start: sec = 248 usec = 668000
  End: sec = 248 usec = 678000
forall becomes false
There is no c2 event holding the exists condition
-- Specification:
forall cl : [cross1] {  
    exists c2 : [report1] {  
        (time(end(cl)) <= time(start(c2))) &&  
        . . .  
    }  
}  

Is TRUE  
-----------------------------------------------

-- Specification:
( ->)always (train1.speed >= 0);  
Is TRUE  
-----------------------------------------------

-- Specification:
( ->)always (speed1 >= 0);  
Is TRUE  
-----------------------------------------------

-- Specification:
( ->)always (speed1 > 0);  
is FALSE because:  
-----------------------------------------------

-- Specification:
forall a_1:[cross1] {  
    (duration(a_1) > 0.002) &&  
    (duration(a_1) < 0.015);  
}  

Is TRUE  
-----------------------------------------------

-- Specification:  
|[unknown]| == 0;  
Is TRUE  
-----------------------------------------------

-- Specification:  
|[empty]| == 0;  
Is TRUE  
-----------------------------------------------

-- Specification:  
|[empty]| == 0;  
Is TRUE  
-----------------------------------------------
forall a_1:[away1] { 
    exists a2_1:[after_away1] { 
        a_1 after a2_1; 
    } 
} 

Is TRUE 
----------

forall a_1:[away1] { 
    a_1 after [after_away1]; 
} 

Is TRUE 
----------

forall c_1 : [cross1] { 
    exists c_2 : [report1] { 
        time(end(c_1)) > (time(end(c_2)) - 
        ... 
    } 
} 

Is FALSE because: 
--------
In the c_1 interval: 
IntervalVar= 
Start: sec = 248 usec = 668000 
End: sec = 248 usec = 678000 
forall becomes false 
There is no c_2 event holding the exists condition 
----------

forall a2_1:[away_report1] { 
    exists a1_1:[cross1] { 
        a2_1 include a1_1; 
    } 
} 

Is TRUE 
----------

forall a2_1 include start(a2_1); 

Is TRUE 
----------

forall a2_1 include end(a2_1); 

is FALSE because: 
--------
----------

forall 
{
    [[cross_away_rep1] -- [not_wait1]] == 0; 
} 

Is TRUE
-- Specification:
\(|[[\text{cross\_away\_rep1} -- \text{not\_wait1}}]| == 0); 
Is TRUE

-- Specification:
\(|[[\text{cross\_away\_rep1} -- \text{not\_wait1}}]| == 0) \&\&  
  \(|[[\text{not\_wait1} -- \text{cross\_away\_rep1}}]| == 0)
. . . 
Is TRUE

-- Specification:
forall a_1:[\text{wait1}] { 
  forall a2_1:[\text{cross\_away\_rep1}] { 
    !(a2_1 intersects a_1);
  }
}
. . . 
Is FALSE because:
In the a_1 interval:
Intervalvar= 
  Start: sec = 0 usec = 768000
  End: sec = 0 usec = 768000
forall becomes false

In the a2_1 interval:
Intervalvar= 
  Start: sec = 0 usec = 738000
  End: sec = 0 usec = 797000
forall becomes false

OPERATION: NOT is FALSE because the operands are: 
  First Operand: 
    Boolvar= true

-- Specification:
forall e1_1:[\text{one\_msec\_bef}] { 
  exists a1_1:[\text{trail\_in\_green}] { 
    (a1_1 include e1_1);
  }
}
. . . 
Is TRUE

-- Specification:
forall a1_1:[\text{report1}] { 
  (0.001 <~ \text{start}(a1_1) -> \text{start}(a1_1)) begin 
    \text{train1semaphore =}
  end
}
. . . 
Is TRUE
forall a2_1:[one_msec_bef_int] {
  exists a1_1:[trail_in_green] {
    (a1_1 include a2_1);
    . . .
    #######
    Is TRUE
  }
}

forall a1_1:[report1] {
  (0.001 <- start(a1_1) -> start(a1_1)) always
  (train1semaphore
    . . .
    #######
    Is TRUE
  )
}

forall a1_1:[btw_cross1] {
  (|
    [tmp:[wait1] at start(tmp) inside a1_1] == 1 |
  );
  #######
  Is TRUE
}

forall a1_1:[before_cross1] {
  (|
    [tmp:[wait1] at start(tmp) inside a1_1] <=
    (N
      . . .
      #######
      Is TRUE
    )
  )
}

forall a1_1:[<train1.> -><<train1.>>T>] {
  tmp = (speed1 - speed1[time(start(a1_1))]);
  . . .
  #######
  Is TRUE
}

forall a1_1:[<train1.> -><<train1.>>T>] {
  tmp = (speed1 - train1(time(start(a1_1))).speed);
  . . .
  #######
  Is TRUE
}