Bridge Requirements: EECS4312

JSO,EECS

September 26, 2016

Contents

List of Figures

List of Tables

1 Introduction

For the Bridge problem, see Fig. 1. You describe the requirements as a complete and disjoint function table and validate the requirements with invariants and use cases. This is also the first complete requirements document that you write and submit (atomic requirements, list of monitored/controlled variables, function table specification and validation of the function table).

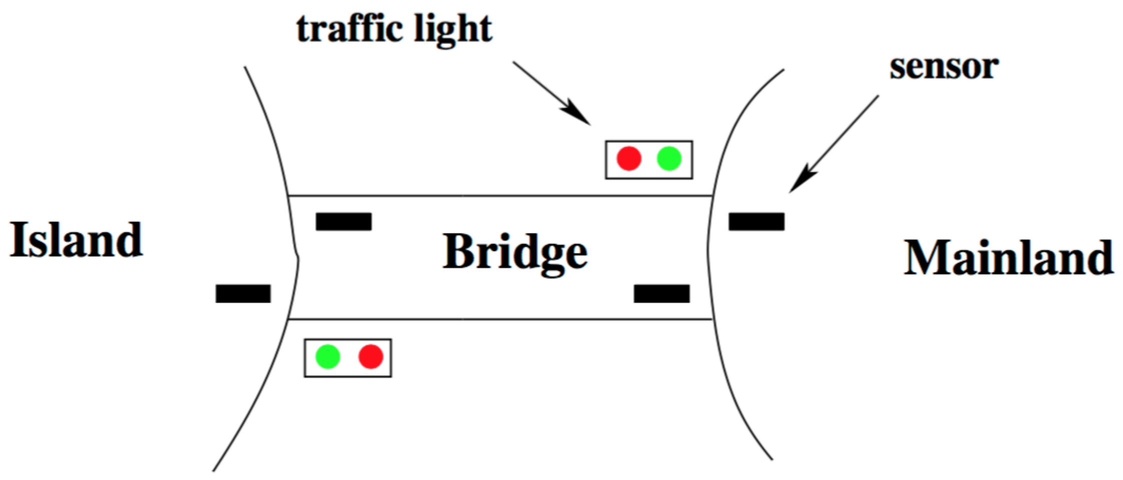
This example is motivated by the second refinement in `http://deploy-eprints.ecs.soton.ac.uk/112/1/sld.ch2.car.pdf` (see slides 234-239 for a summary).

2 Informal Specification

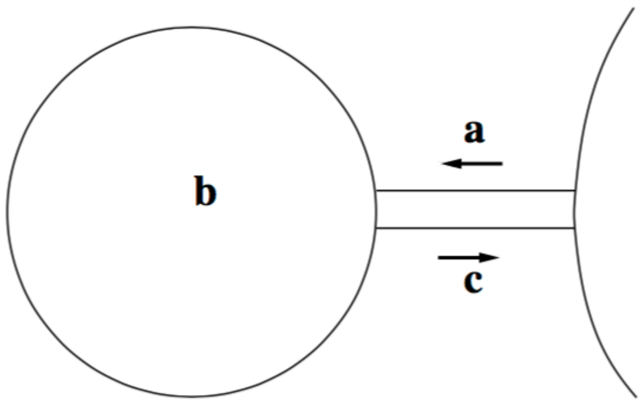
The system controls cars on a bridge connecting the mainland to an island. The system is equipped with traffic lights to control the entrance to the bridge at both ends. It is assumed that cars only pass on to the bridge when the light is green. There are sensors and circuitry that provide measurements ($a, b, c \in \mathbb{N}$) of the traffic pattern in the system. There is a constant $d \in \mathbb{N}1$ that limits the total number of cars on the bridge and island at the same time, i.e. $a + b + c < d$. The bridge is one way, or the other, not both at the same time.

Question 1.

- What is the system boundary?
- How would you draw the context diagram?
- What are the monitored variables?



$a, b, c \in \mathbb{N}$



a denotes the number of cars on bridge going to island

b denotes the number of cars on island

c denotes the number of cars on bridge going to mainland

Figure 1: Controlling Cars on a Bridge

- What are the controlled variables?
- What types do we need?
- Are there any important system invariants?

3 PVS Specification and the Main Function Table

In T-ASM theory we might start writing the specification as shown in Listing 1.

Question 2. Do you understand the main-function-table `spec` in the listing? Note that we decompose the main function table `spec` into a smaller chunks, in this case a sub-table `spec_pos` ($i > 0$) that takes a precondition `precond`. The precondition can be seen in Listing 2 (see `spec_pos_pre`). Ensure that you understand the precondition (needed for completeness of the sub-function-table), which is like the **assume** you have seen in informal function tables up to now. When doing proofs of completeness/disjointness as TCCs, we can use the proof rule `(typepred "p")` to obtain the precondition. Next, we need another sub-function-table `spec_red_red` which we leave for you to define) The "do-nothings" are needed for disjointness (ensure that you understand this).

4 PVS Validation of the Function Table

Obviously your function table must be proved complete and disjoint and will which is why we use the COND construct. As before, ensure that you do not specify a circular function table. You cannot observe and change a control variable at the very same time. There are various requirements that you must capture in your function table.

Your function table can be validated with a system invariant (`invariant`) shown below:

```
invariant: CONJECTURE
   (FORALl (i:DTIME) : spec(i))
   IMPLIES
   (FORALL (i:DTIME) : NOT (c_ml(i) = green AND c_il(i) = green))
```

You must also prove Use Case shown in Listing 3.

```
bridge: THEORY
BEGIN
delta: posreal
IMPORTING Time[delta]

%Used for writing preconditions
precond : DATATYPE
   BEGIN
      precond: precond?
   END precond
PRE (p : bool) : TYPE = { x : precond | p }

COLOR: TYPE = {green, red}

TURN: TYPE = {mainland, island}

d: upfrom(1) %nat1
% Monitored variables
m_a: [DTIME -> nat] % No of cars on bridge towards island
m_b: [DTIME -> nat] % No of cars on island
m_c: [DTIME -> nat] % No of cars on bridge twoards mainland

s_turn: [DTIME -> TURN] % mainland and island pass variables

%Controlled variables
c_ml: [DTIME -> COLOR] %Mainland traffic light
c_il: [DTIME -> COLOR] %Island traffic light
c_err: [DTIME -> bool]

...% TBD

% Main Function table
spec(i:DTIME): bool =
  COND
      i=0
       -> c_ml(i)=red AND c_il(i)=red
          AND s_turn(i)=mainland AND c_err(i)=False
    , i>0
      AND m_a(i) + m_b(i) + m_c(i) <= d
      AND (m_a(i) = 0 OR m_c(i) = 0)
      AND NOT (c_il(i-1) = green AND c_ml(i-1) = green)
      -> spec_pos(i)(precond) AND c_err(i)=False
    , i>0
      AND (   m_a(i) + m_b(i) + m_c(i) > d
           OR (m_a(i) > 0 AND m_c(i) > 0)
           OR (c_il(i-1) = green AND c_ml(i-1) = green))
      -> c_err(i) = True AND c_ml(i)=red AND c_il(i)=red
  ENDCOND
```

Table 1: PVS Specification

```
spec_pos_pre(i : POS_DTIME): bool =
                      m_a(i) + m_b(i) + m_c(i) <= d
                  AND (m_a(i)=0 OR m_c(i)=0)
                  AND NOT (c_il(i-1) = green AND c_ml(i-1) = green)

spec_pos(i:POS_DTIME)(p: PRE(spec_pos_pre(i))): bool =
  COND
    % when both lights are red
        c_ml(i-1)=red AND c_il(i-1)=red
     -> spec_red_red(i)(p)
    % Turn mainland traffic light red
    ,     c_ml(i-1)=green
      AND c_il(i-1)=red
      AND s_turn(i-1)=island
      AND 0 < m_b(i) AND m_a(i)=0
     ->   c_ml(i)=red
      AND c_il(i)=red
      AND s_turn(i)=s_turn(i-1) %NC pass variables
    % Do nothing
    ,       c_ml(i-1)=green
        AND c_il(i-1)=red
        AND (s_turn(i-1)=mainland OR 0 = m_b(i) OR m_a(i) > 0)
     ->     c_ml(i)=c_ml(i-1)
        AND c_il(i)=c_il(i-1)
        AND s_turn(i)=s_turn(i-1)
  % Turn island traffic light red
    ,     c_ml(i-1)=red
      AND c_il(i-1)=green
      AND s_turn(i-1)=mainland
      AND m_b(i)=0
     ->   c_ml(i)=red
      AND c_il(i)=red
      AND s_turn(i)=s_turn(i-1) %NC pass variables
    % Do nothing
    ,       c_ml(i-1)=red
        AND c_il(i-1)=green
        AND (s_turn(i-1)=island OR  m_b(i)> 0)
     ->     c_ml(i)=c_ml(i-1)
        AND c_il(i)=c_il(i-1)
        AND s_turn(i)=s_turn(i-1)
  ENDCOND
```

Table 2: spec_pos

```
 % use case
j: DTIME
usecase1: CONJECTURE
       j > 0
   AND spec(j-1) AND spec(j) AND NOT c_err(j-1)
   AND c_ml(j-1)=red AND c_il(j-1)=red
   AND s_turn(j-1) = mainland
   AND m_a(j) + m_b(j) < d AND m_c(j)=0
   IMPLIES
    c_ml(j)=green AND c_il(j) = red AND NOT c_err(j)
END bridge
```

Table 3: Use Case

Question 3. What does the Use Case check?

You should see the following when done:

```
Proof summary for theory bridge
    spec_pos_pre_TCC1.....................proved - complete
    spec_red_red_pre_TCC1.................proved - complete
    spec_red_red_TCC1.....................proved - complete
    spec_red_red_TCC2.....................proved - complete
    spec_red_red_TCC3.....................proved - complete
    spec_pos_TCC1.........................proved - complete
    spec_pos_TCC2.........................proved - complete
    spec_pos_TCC3.........................proved - complete
    spec_pos_TCC4.........................proved - complete
    spec_TCC1.............................proved - complete
    spec_TCC2.............................proved - complete
    spec_TCC3.............................proved - complete
    spec_TCC4.............................proved - complete
    spec_TCC5.............................proved - complete
    invariant.............................proved - complete
    usecase1_TCC1.........................proved - complete
    usecase1..............................proved - complete
    Theory totals: 17 formulas, 17 attempted, 17 succeeded (3.73 s)

Grand Totals: 17 proofs, 17 attempted, 17 succeeded (3.73 s)
```

In proving the Use Case, there is only one path that you must unwind with `(expand)` as shown in Fig. 2, due to disjointness. The other branches discharge easily with `(grind)`.

5 Writing a Requirements Document for the Car Interlock

You do not need to write a requirements document for the Bridge system. But as before, you might consider what a Requirements document might look like.

- Where is the System Boundary?
- What are the monitored variables? What are their types?
- What are the controlled variables? What are their types?
- Specify a complete and disjoint function table that describes the input/output behaviour of the SUD? (You might want to draw this table in your requirements document to help with the PVS specification.)
- Now (a) write out the atomic R-descriptions for the plant (number them) (b) state what the monitored variables are and in a different table what the controlled variables are and (c) draw the function table for the car interlock system and provide evidence that you have validated the function table.
- Submit your document as `car-interlock.pdf`.
- We will be using Latex to prepare documentation for the assignment and project. You may try to prepare the document using Latex. See `https://wiki.eecs.yorku.ca/project/sel-students/p:tutorials:latex:` (login). There is a link to a Latex table generator.
- You are not required to use Latex for this Lab. Use any documentation preparation system you like provided it is neat.

Ensure that it is neat! Ensure that it is minimal! It does not require more than half a page in a large font. Too many cooks spoil the broth (i.e. too many rows and not enough organization and thought spoil the function table).

6 What will be in your document

Understand and agree upon what users want before attempting to create solutions.

Finding out what is needed instead of rushing into presumed solutions is the key to every aspect of system development. Most technical problems can be solved, given determination, patience, a skilled team—and a well-defined problem to solve.

A precise requirements document will contain all the information needed by the developers to build the system wanted by the users—and no more (i.e. it should not be polluted with design and implementation detail).

It is important to write the informal requirements atomically (with a number to track the requirements), e.g.

REQ1	The bridge is one way or the other but not both at the same time

The above requirement will be checked using the invariant conjecture in PVS.
Your document will contain:

- Informal statement of the problem
- Context diagram
- Table of monitored variables with its types and another table of controlled variables with their types.
- Atomic requirements.
- The function table that is complete and disjoint.
- Various use cases (one is enough for this introductory document)
- The PVS specification of the function table and validation of completeness/disjointness, invariants and use cases. We showed one use case, but in general there will be many.

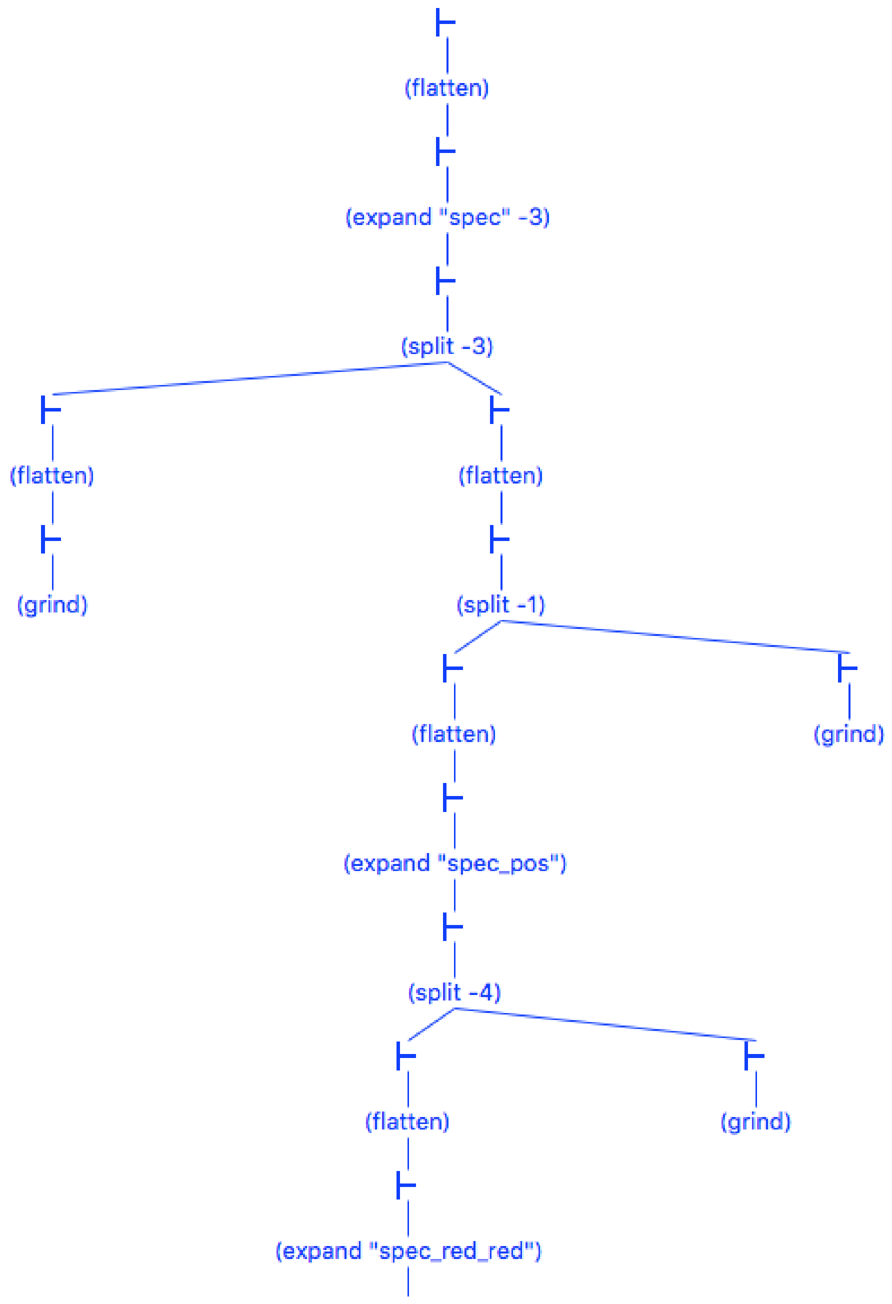


Figure 2: Proof Tree for Use Case