Car-Interlock

Requirements Document-Lab3

EECS 4312 Section E
Fall 2016
Lassonde school of engineering

By
Bhumika Patel
#213245808

10/6/2016

Table of Contents

Table of figures

Informal specifications

This problem is about creating a software system for entry/exit of a parking lot. The parking lot is a single lane passage. We have to ensure that by controlling the indicators; only one car can pass through the entry/exit at a time in order to prevent car accidents between entering and leaving cars.

Atomic R-descriptions

REQ1	The traffic stop/go lights shall not be green at very same time

REQ2	Only one car shall pass through entry/exit at a time

REQ3	The cars waiting to exit shall go before cars waiting to enter

REQ4	When the exit light is off, the entering car must wait for previous exiting car to exit first

Context Diagram

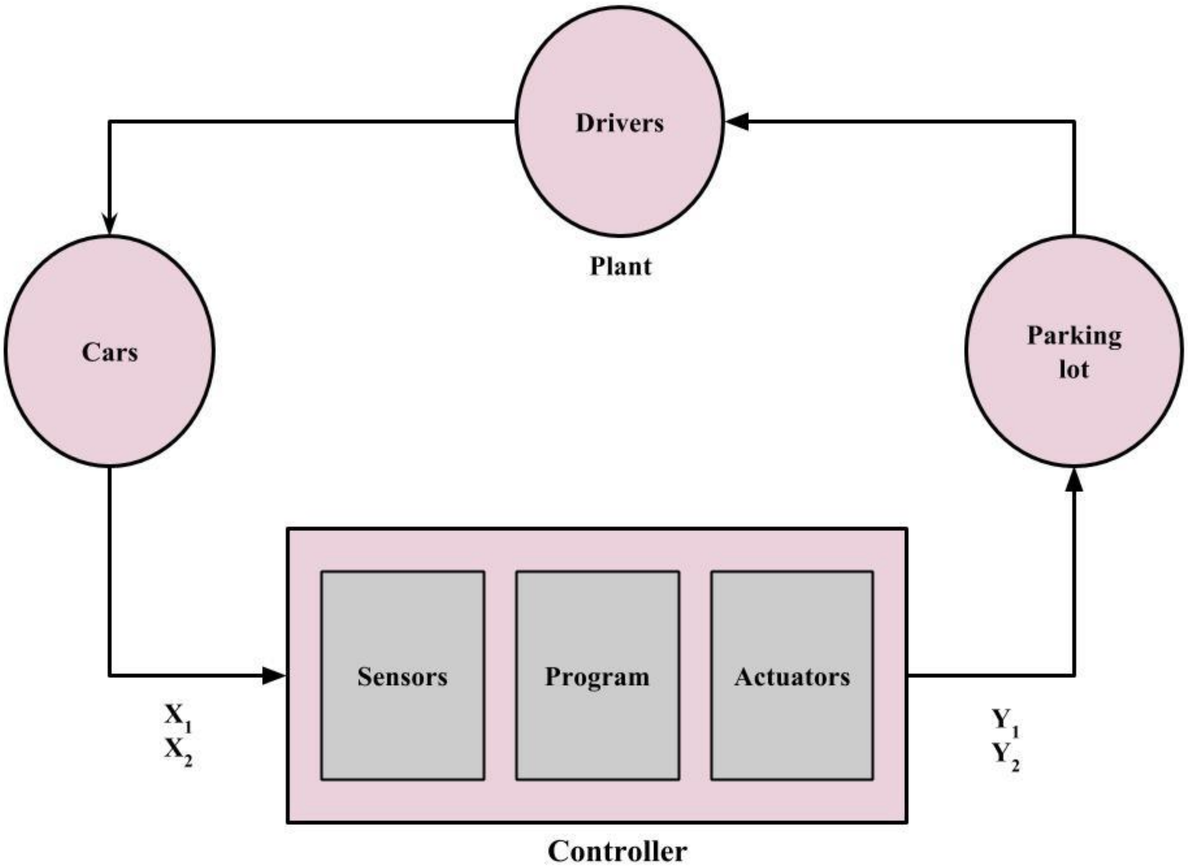

Figure 1: Context diagram

Variables

Monitored variables:

Monitored variables	Type	Description
X_1	[DTIME ->SENSOR]	Sensor for entering car
X_2	[DTIME ->SENSOR]	Sensor for exiting car

Figure 2: Monitored variables

Controlled variables:

Controlled variables	Type	Description
Y_1	[DTIME ->ACTUATOR]	Traffic light(actuator) for entering car
Y_2	[DTIME ->ACTUATOR]	Traffic light(actuator) for exiting car

Figure 3: Controlled variables

Function Table

Input				**Output**	
				$Y_1(i)$	**$Y_2(i)$**
i = 0				stop	stop
i > 0	$X_2(i)$ = on			stop	go
	$X_2(i)$ = off	$X_1(i)$ = on	$Y_2(i-1)$ = go	stop	stop
			$Y_2(i-1)$ = stop	go	stop
		$X_1(i)$ = off	$Y_2(i-1)$ = go	stop	stop
			$Y_2(i-1)$ = stop	stop	stop

Figure 4: Function table

Use cases

Main use case

This use case given in the document tests all the features of the function table.

use_case: CONJECTURE

(control_ft(0) AND control_ft(1) AND control_ft(2) AND control_ft(3) AND x1(1) = on AND x2(1) = on AND y1(0) = stop AND x1(2) = on AND x2(2) = off AND x1(3) = on AND x2(3) = off)
=>
(y2(1) = go AND y1(1) = stop AND y1(2) = stop AND y2(2) = stop AND y2(3) = stop AND y1(3) = go)

Other use cases

I created these use cases to individually check different functions of the function table. And all the use cases were provable.

use_case1: CONJECTURE
x2(2) = off AND x1(2) = on AND y2(1) = go AND control_ft(2)
=>
y1(2) = stop AND y1(2) = stop

use_case2: CONJECTURE
control_ft(4) AND x2(4) = on => y1(4) = stop AND y2(4) = go

use_case3: CONJECTURE
x2(2) = off AND x1(2) = on AND y2(1) = stop AND control_ft(2)
=>
y1(2) = go AND y2(2) = stop

use_case4: CONJECTURE
x2(2) = off AND x1(2) = off AND y2(1) = go AND control_ft(2)
=>
y1(2) = stop AND y1(2) = stop

use_case5: CONJECTURE
x2(2) = off AND x1(2) = off AND y2(1) = stop AND control_ft(2)
=>
y1(2) = stop AND y1(2) = stop

Validation of use_case

Below is the proof tree for the use_case(the one given in the document) which also helps validate the function table.

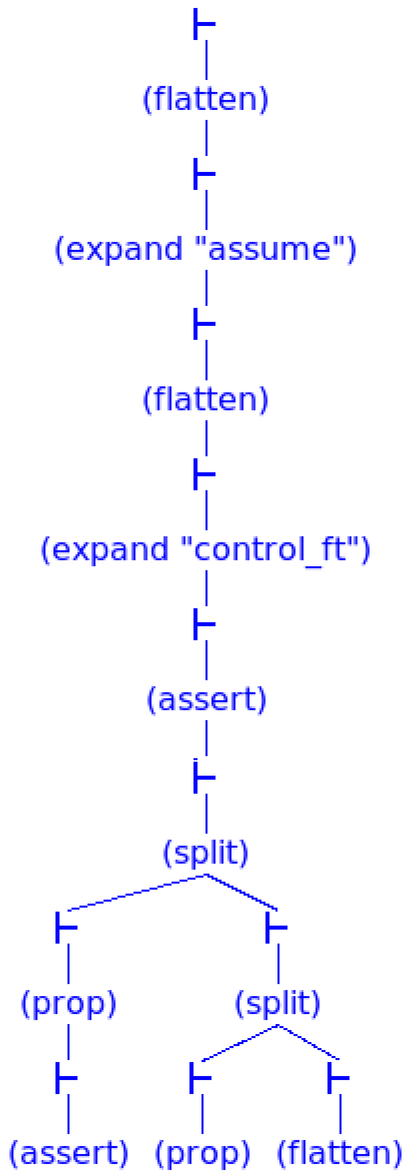

Figure 5: Proof tree of use_case

Invariants

Below is the system invariant for car interlock. This invariant makes sure that for all DTIME i, only one light is green that is only one car can pass through the parking lot at particular instance of time and two cars cannot enter or exit parking lot at the same time. And I was able to prove this invariant successfully.

inv(i): bool = NOT (y1(i) = go AND y2(i) = go)

inv_holds:CONJECTURE
(FORALL i: control_ft(i)) => (FORALL i: inv(i))

Validation of invariant

Below is the proof of invariant which validates the function table.

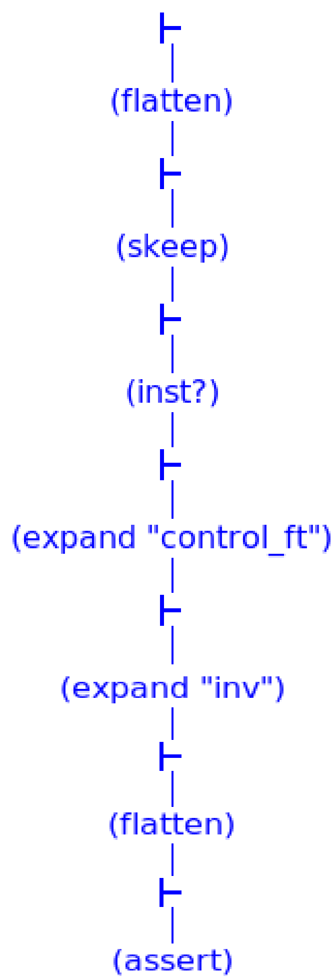

Figure 6: Proof tree of invariant

PVS specification of the function table

control_ft(i):bool=
COND
i=0 -> y1(0) = stop AND y2(0) = stop,
i>0 ->
COND
x2(i) = on -> y1(i) = stop AND y2(i) = go,
x2(i) = off ->
COND
x1(i) = on ->
COND
y2(i-1) = go -> y1(i) = stop AND y2(i) = stop,
y2(i-1) = stop -> y1(i) = go AND y2(i) = stop
ENDCOND,
x1(i) = off ->
COND
y2(i-1) = go -> y1(i) = stop AND y2(i) = stop,
y2(i-1) = stop -> y1(i) = stop AND y2(i) = stop
ENDCOND
ENDCOND
ENDCOND
ENDCOND

Validation of completeness/dis-jointness of function table

Below is the proof of this problem car interlock. It shows that all TCCS related to disjoitness, completeness and type correctness were proved successfully. We can also see from function table specification that it is complete and disjoint that is the function table has all possible inputs and one cannot be in two rows of the table at the same time. Also we have shown in previous parts that all the use cases and system invariants were proved successfully which also validate the function table.

```
]Proof summary for theory car_interlock
   control_ft_TCC1.......................proved - complete   [shostak](0.10 s)
   control_ft_TCC2.......................proved - complete   [shostak](0.10 s)
   control_ft_TCC3.......................proved - complete   [shostak](0.12 s)
   control_ft_TCC4.......................proved - complete   [shostak](0.07 s)
   control_ft_TCC5.......................proved - complete   [shostak](0.09 s)
   control_ft_TCC6.......................proved - complete   [shostak](0.08 s)
   control_ft_TCC7.......................proved - complete   [shostak](0.07 s)
   control_ft_TCC8.......................proved - complete   [shostak](0.06 s)
   control_ft_TCC9.......................proved - complete   [shostak](0.08 s)
   control_ft_TCC10......................proved - complete   [shostak](0.04 s)
   control_ft_TCC11......................proved - complete   [shostak](0.04 s)
   inv_holds.............................proved - complete   [shostak](0.10 s)
   use_case..............................proved - complete   [shostak](0.24 s)
   use_case1.............................proved - complete   [shostak](0.14 s)
   use_case2.............................proved - complete   [shostak](0.06 s)
   use_case3.............................proved - complete   [shostak](0.15 s)
   use_case4.............................proved - complete   [shostak](0.07 s)
   use_case5.............................proved - complete   [shostak](0.09 s)
   Theory totals: 18 formulas, 18 attempted, 18 succeeded (1.71 s)
```

Figure 7: Proof summary in PVS