

WEEK 8 3806ICT

TASK 1

SOURCE CODE

Each process essentially defines all the possible transitions for the different operations. These processes are owner positions, key positions, door operation and motor. Note that the syntax is slightly different as the subprocesses allow the parameter to be printed in the subprocess call such as `towards.i{}`. With this, if its called with owner 1 then it would print `towards.1{}`

In owner position there are four possible movements. When the owner is far away, they can move towards the car, and when near they can go away to be far from the car. Granted the owner is near, the door is open and the car is not moving then the owner can climb in. Furthermore, in the case that the owner is inside, the door is open and the car is moving then they can climb out.

For the key position, when the key matches the car and the key's owner is in the car they can put it in the car. The second subprocess is when the key matches the car, but the owner is far, and in this case the key is set to faralone: then the key is put outside and far. In the case that the owner is with the key (either both far or both in the car), then the owner can get the key – setting its value to i.

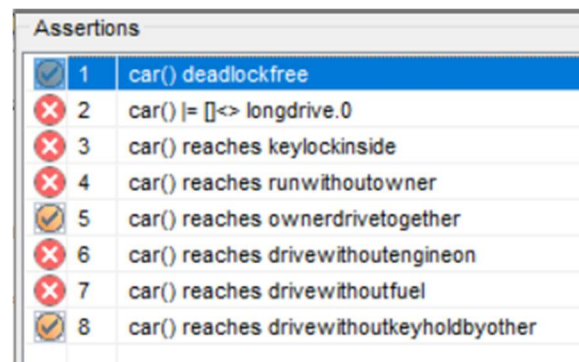
Operating the door has six operations. The first includes when the key matches the car, the owner is near, the door is locked and the car is stationary then the owner can unlock the vehicle. In the case that the owner is near, door is unlocked and the car is stationary they can open the door. When the door is open, it can be closed, and likewise for when it is closed but the owner is still inside then it can be opened. If the owner is inside and its unlocked, they can unlock it and when the door is unlocked, the owner is near and the key matches the car then the owner can lock it.

For the motor operation there are seven subprocesses. To turn the engine on, the owner must inside and the key must match the car, or the key is already in the car, and the engine must be off with fuel in the car. To begin driving, the engine must be on, the owner must be inside and it must be stationary. In the case that it is moving and there is less than 5 fuel, the fuel can be decremented by one and if the fuel runs out it stops moving for a short drive. To make a long drive requires more than five fuel and the car to be moving, causing the fuel to be decremented by five with a check for when the fuel is out the engine should turn off and the car stop moving.

In the case that the car's engine is on, the car is moving and the owner is inside the car can stop, setting the moving to zero. When the fuel is out and the engine is off, the car can be refilled; setting the fuel to ten. To turn the engine off, the engine must be on, the vehicle must be stationary and the owner must be inside.

The program is initialised at line 90 where the `|||` symbol is used to make the processes run simultaneously. For each owner that was set, simulate the motor, door operations, key positions and owner positions. Numerous states are defined, `runwithoutowner`, `ownerdrivetogether`, `keylockinside`, `drivewithoutengineon`, `drivewithoutfuel`, `drivewithoutkeyholdbyother`.

ASSERTION CHECKING



Assertions		
<input checked="" type="checkbox"/>	1	car() deadlockfree
<input type="checkbox"/>	2	car() != []<> longdrive.0
<input type="checkbox"/>	3	car() reaches keylockinside
<input type="checkbox"/>	4	car() reaches runwithoutowner
<input checked="" type="checkbox"/>	5	car() reaches ownerdrivetogether
<input type="checkbox"/>	6	car() reaches drivewithoutengineon
<input type="checkbox"/>	7	car() reaches drivewithoutfuel
<input checked="" type="checkbox"/>	8	car() reaches drivewithoutkeyholdbyother

FIGURE 1 - ASSERTION VERIFICATION

For the deadlock, this simply means that the model is explored until either all transitions are explored or the model has a state that can only terminate the program.

The `!=[]<>` is outside the course content.

As for `reaches keylockinside`, it tests whether its possible for the key to be in the car, the door locked, and neither owner in the car; in other words, the key locked inside. This is asserted as false because to lock the car from outside, the key must be outside.

The `can running without the owner` is never reached, because for the car to run with the owner one of the conditions is that the owner is inside of the car.

Owners can drive in a car together because to climb in the door only needs to be unlocked, then the rest of the conditions are the same.

Driving without the engine on fails because the conditions for the moving subprocess requires the engine to be on, and the same principle for driving without fuel.

Driving without the key can happen as well when the car was already on and the owner with the key climbs out.

TASK 2

MOVES TO COMPLETE

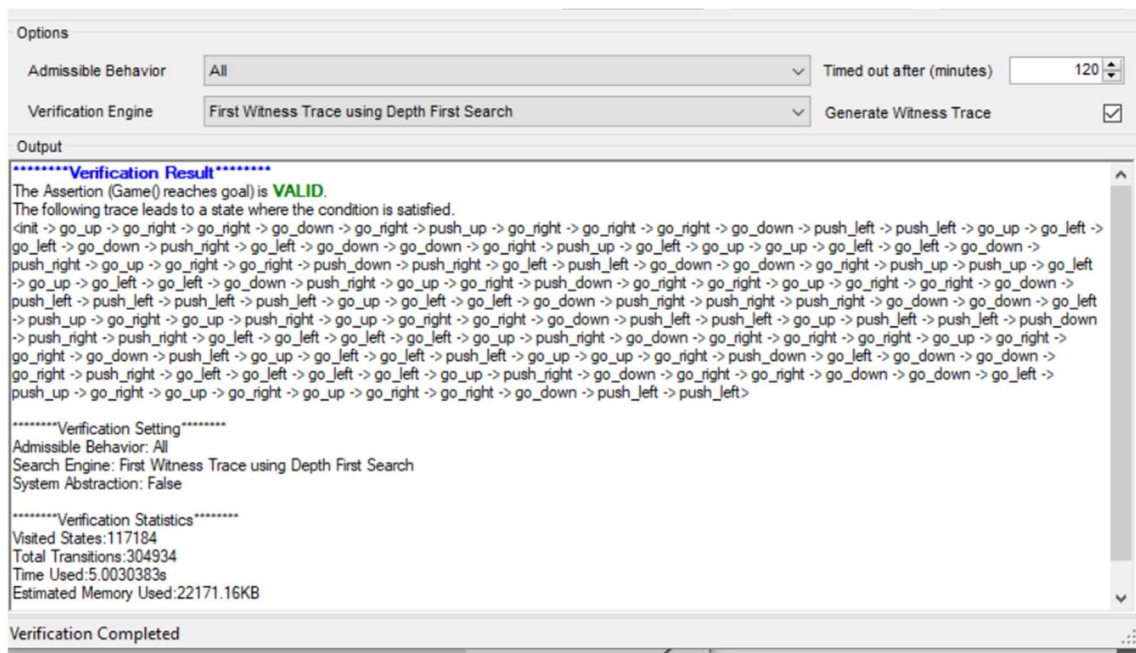
Could not find a solution, that is why we are using PAT.

UNDERSTANDING

In the source code the board is defined using a two dimensional array with obstacles (o), available (a) and white (w). As black is moving it's simpler to define its position on its own (instead of keeping it in the board), and this is done by defining an integer with a range of 0->N-1 for the rows and 0->M-1 for columns. Following this the possible game moves are defined as move up, down, left, right and a push for each of these. Essentially each adds one value in the given direction when the square adjacent is available, and the pushes move the white as well when it is in the way. These are 'hosted' by the game process, which manages the black's coordinates in a way that is impossible for it to get off the board (if moving up, it's row cannot be higher than r-1 (one below the top)).

With regards to the assertions, the first two check whether its possible for a white to arrive against the edge (its stuck as the black cannot push it out), and

DFS GENERATED PLAN



This shows it is possible, which has 136 transitions.

Event Trace		
Src State	Event	Tgt State
112	go_down	113
113	go_right	114
114	push_right	115
115	go_left	116
116	go_left	117
117	go_left	118
118	go_left	119
119	go_up	120
120	push_right	121
121	go_down	122
122	go_right	123
123	go_right	124
124	go_down	125
125	go_down	126
126	go_left	127
127	push_up	128
128	go_right	129
129	go_up	130
130	go_right	131
131	go_up	132
132	go_right	133
133	go_right	134
134	go_down	135
135	push_left	136
136	push_left	137

BFS SEARCH

Selected Assertion

Game() reaches goal

Verify
View Büchi Automata
Simulate Witness Trace

Options

Admissible Behavior
All
Timed out after (minutes)
120

Verification Engine
Shortest Witness Trace using Breadth First Search
Generate Witness Trace
☒

Output

*****Verification Result*****
The Assertion (Game() reaches goal) is **VALID**.
The following trace leads to a state where the condition is satisfied.
<init -> go_up -> go_right -> go_right -> go_down -> go_right -> push_up -> go_down -> go_left -> go_down -> go_down -> go_right -> push_up -> push_up -> go_left
-> go_up -> go_left -> go_left -> go_down -> push_right -> go_up -> go_right -> push_down -> go_right -> go_right -> go_up -> push_left -> push_left -> go_up ->
go_up -> go_right -> push_down -> go_left -> go_down -> go_down -> go_left -> go_left -> go_up -> push_right -> go_down -> go_right -> go_right -> go_down ->
go_down -> go_left -> push_up -> go_right -> go_up -> go_right -> go_up -> go_right -> go_right -> go_down -> push_left -> push_left>

*****Verification Setting*****
Admissible Behavior: All
Search Engine: Shortest Witness Trace using Breadth First Search
System Abstraction: False

*****Verification Statistics*****
Visited States: 112140
Total Transitions: 291001
Time Used: 5.6177768s
Estimated Memory Used: 28674.864KB

This shows the shortest possible path, which has 54 moves

Event Trace

Src State	Event	Tgt State	
30	go_right	31	
31	push_down	32	
32	go_left	33	
33	go_down	34	
34	go_down	35	
35	go_left	36	
36	go_left	37	
37	go_up	38	
38	push_right	39	
39	go_down	40	
40	go_right	41	
41	go_right	42	
42	go_down	43	
43	go_down	44	
44	go_left	45	
45	push_up	46	
46	go_right	47	
47	go_up	48	
48	go_right	49	
49	go_up	50	
50	go_right	51	
51	go_right	52	
52	go_down	53	
53	push_left	54	
54	push_left	55	