Graded lab assignment: Container ship

1 Overview

The NuSMV model for this exercise describes a ship's cargo hold with a cooling compartment that can keep cargo frozen efficiently. To achieve this, the cargo hold has an airlock with two doors where only one door can be open at a given time (see Figure 1). The doors are controlled by buttons on either side of the door. The goal of this exercise is to model the doors and the controller and to

1. complete the model by formalizing the transition relations,
2. formalize the given properties as temporal-logic properties, and
3. ensure that the requirements are satisfied.

The base model has to be complete and correct for passing. Furthermore, you are not allowed to make additional assumptions about the initial state of physical components if not already given by the template. If your model passes all mandatory features, you can implement optional features for a higher grade.

2 Base features (required for P)

Figure 1 shows the lab, which contains an airlock with double doors and a button on each side of each door.

2.1 Buttons

There is one pair of buttons for each door. Each button can be pressed non-deterministically. A pressed button stays active until the controller resets it (see Algorithm 1).

2.2 Door

The airlock is equipped with two doors that can either be open or closed. The door responds to commands (inputs) "open", "close", and "nop", which cause it to open, close, or stay in its current state.

2.3 Tasks: Airlock (controller)

The airlock controller opens and closes the doors. It takes as input (sensor data) the status of each door and the status of all buttons. It also has a state variable *last_ open,* which denotes which door was opened last. This variable is needed to avoid giving precedence to the same door all the time (and avoid a possible livelock situation). The outputs of the controller are *inner_ door_ cmd* and *outer_ door_ cmd.*

Tasks: Specify the controller logic for the inner and outer door.

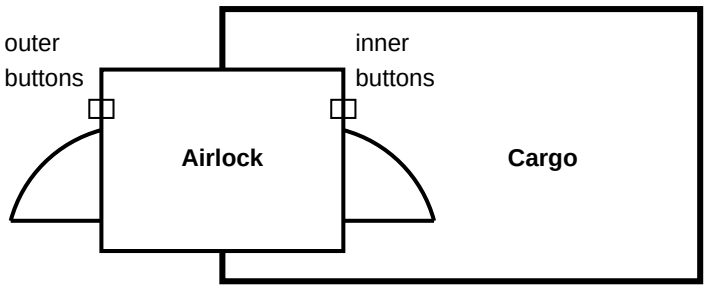

Figure 1: Schematics of the cooling cargo compartment with the airlock.

Algorithm 1 Button and door modules.

```
MODULE Button(reset)
  VAR pressed : boolean;
  ASSIGN
    init(pressed) := FALSE;
    next(pressed) := case
        pressed & reset  : FALSE;
        pressed & !reset : TRUE;
        !pressed         : {FALSE,TRUE};
    esac;
```

```
MODULE Door(door_cmd)
  VAR status : { open, closed };
  ASSIGN
    init(status) := closed;
    next(status) := case
        door_cmd = open  : open;
        door_cmd = close : closed;
        door_cmd = nop   : status;
    esac;
```

2.4 Tasks: Safety/liveness properties

The following properties have to be specified in temporal logic. Properties in *italics* may require multiple temporal logic formulas to be fully captured (one for the inner and one for the outer door). You are always free to choose whether to write multiple properties or a conjunction of properties, but the former is recommended for readability:

1. Both doors are never open together.
2. *A door only opens if a button is pressed.*
3. If both buttons are pressed, the inner door should take precedence unless it was just previously opened.
4. *Both doors must eventually open (i.e., for each door, there must eventually occur a state where it is open).*
5. *No button can reach a state where it remains pressed forever.*

You are free to choose LTL or CTL (where applicable).

3 Tasks for higher grades

Note: It is not necessary to implement all optional features in the *same* model; you can also create multiple models that each implement one or more optional features.

3.1 Advanced properties

Specify these additional properties (again, *italics* indicates that using multiple properties may be easier). The first two properties have to be specified for each door. The third property needs to combine the behavior of both doors in the same property.

1. *No pressed button can be reset until the door opens.*
2. *A button must be reset as soon as the door opens.*
3. If both buttons are pressed, then first one door opens, followed by the other one.

3.2 Door safety

The door contains a sensor, which is triggered if something obstructs the door and may physically prevent it from closing.

Modifications: Add two non-deterministic variables *inner_sensor* and *outer_sensor* to the controller (**main** module), and update the door module to fulfill the new property (below).

Property: *The doors never close when the sensor is on.*

Fairness: States where neither sensor is triggered and no door button is pressed occur infinitely often.

Analysis: What happens without the new fairness property? Explain **one** of the resulting error traces.

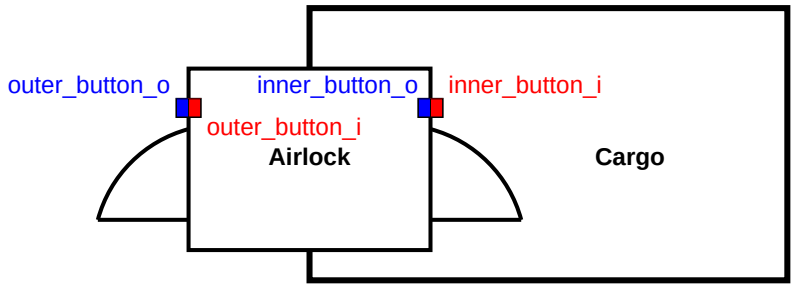

Figure 2: Extended button model to distinguish inside and outside buttons.

3.3 Temperature control

The ship has a thermostat to regulate the cooling. With this feature, the controller (module **main**) maintains two states, "normal" and "overheating". When the temperature is too high, the controller switches from "normal" to "overheating". If the cargo hold is overheated, requests to open the doors from the outside of each airlock door are ignored until the temperature drops again and the mode goes back to "normal". (Requests from inside are allowed so people do not get trapped inside.)

This requires several modifications of the original model:

1. Buttons: You have to instantiate four buttons overall now (two related buttons for each door; see Figure 2). Depending on how you extend your model, a button may now be reset even if it was not pressed before. You are allowed to simplify/generalize the way a button works, and rewrite the button invariant (even specifying a similar property in a different part of the model) if needed.
   Resetting a button if there was only a request on the other side of the same door will simplify the model (e.g., you can now reset *inner_button_i* if the door opened only because of a request from *inner_button_o* and the former button was never pressed).

2. Airlock: The airlock interface becomes more complex:
   `MODULE Airlock(inner_door, outer_door, inner_button_i, inner_button_o, outer_button_i, outer_button_o, mode)`

3. Door functionality: The door (eventually) opens after the push of either button in normal mode, as before, but in "overheat" mode, it only answers to the *inside* buttons (_i). Make sure that these key properties refer to the exact button state (e.g., *inner_button_i*) and not the value of a macro, as to avoid carrying over mistakes of an incorrect macro to the property checking behavior related to it.

4. Mode: There are two new states, "normal" and "overheating"; the mode can change non-deterministically between them.

5. Design: It is recommended to add a macro (`DEFINE`) to define an alias for combining the inside and outside buttons of each door. This lets you reuse the properties with a minor syntactical change, and avoids requiring a complex conjunct in every property that refers to buttons. You can also define "helper variables" that combine the values of multiple inputs or state variables to minimize changes in your logic.

6. Fairness properties: Change the existing fairness properties such that the *inside* button on each door is eventually pressed.

7. Requirement 3: Rewrite the requirement 3 (about precedence) as follows:
   If both *inside* buttons are pressed, the inner door should take precedence.

8. Advanced requirement 1: Change it to
   A request from a pressed button remains active until the door opens *or overheating occurs.*

3.4 Validation, documentation

Extra points can be obtained by validating and documenting the model. Commented error traces can be pasted and explained at the bottom of the submission, as a multi-line comment between `/--` and `--/`.

Description	Feature	Property	Trace/doc	Points
Extra properties		medium–difficult		+0.75
The in-door sensors are implemented correctly, with a correct property, fairness condition (if needed) and reasoning about the fairness condition.	easy	easy	easy	+0.75
Temperature control	difficult	medium		+1.5
Documented error trace of incorrect model.	easy			+0.25
Documented error trace of incorrect property		easy		+0.25
Simulation			easy	+0.25
Modeling/environment			easy	+0.25

Table 1: Extra points for advanced features/properties.

Trace: You document an example error trace with a faulty (older?) or mutated *model*. (+0.25)

Trace: You document an example error trace with a faulty (older?) or mutated *property*. (+0.25)

Trace: You carry out a simulation with the correct model, to 6 steps from an initial state, and document the outcome. (+0.25)

Analysis: What kind of feature or property do you think is missing in this model but would make sense in this setting? Explain briefly in natural language (without a formal specification) what the feature or property would be, and how it interacts with or relates to the current system. (+0.25)

4 Grading criteria: Points for correct extension and properties

Pass (E):

- The submitted work is your group's own original work. You may look at examples from the web for inspiration, but you are not allowed to copy code from the web.
- All mandatory properties are correct.
- The model correctly implements all mandatory properties (no error traces found by NuSMV).
- No additional assumptions about the initial state of physical components are made.

Extra points are awarded according to Table 1.

Reduced points for late submissions! The maximal number of points is reduced by 1 for each missed deadline. There are two deadlines: for the small lab exercises (tutorials) and for this exercise.