Graded lab assignment: Biosafety lab

1 Overview

Figure 1: Inside a biosafety lab.

The NuSMV model for this exercise describes a Biosafety lab (see Figure 1) with an airlock and its controller. The airlock has two doors, which may never be open at the same time. The doors are controlled by buttons on either side of the door. Furthermore, the air in the airlock itself can be "cleaned" by ensuring that higher air pressure is re-instated after a door has been opened, so harmful particles cannot escape the lab. The goal of this exercise is 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 2 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 "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, and activates cleaning for the airlock space between the doors. It takes as input (sensor data) the status of each door, the status of all buttons, and its current state. The state is "clean" after cleaning, and "dirty" as soon as the inner door has been opened.

Tasks: Specify the controller logic for the inner and outer door, and for the correct state of the airlock ("clean" or "dirty").

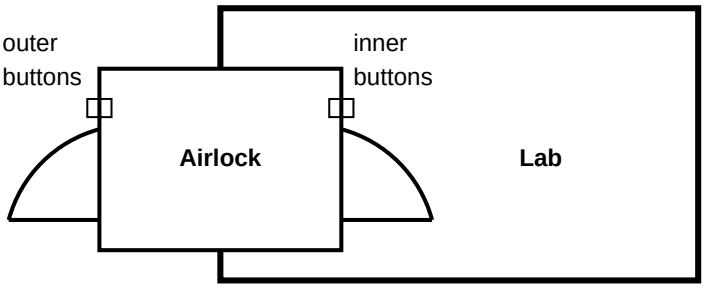

Figure 2: Schematics of the lab with the airlock.

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. 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.
4. *Either door must eventually open (i.e., for either door, there must eventually occur a state where it is open).*
5. The state of the airlock is always "dirty" if the inner door is open.
6. The next state of the airlock is always "dirty" if the inner door is open.
7. The outer door never opens for a dirty airlock.
8. *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).

1. *No pressed button can be reset until the door opens.*
2. *A button must be reset as soon as the door opens.*
3. A dirty airlock remains dirty if one of the doors is open or "cleaning" is *false*.

3.2 Automated cleaning

The airlock should know by itself when to "clean" itself. To this end, implement the following:

1. Remove the (non-deterministic) variable "cleaning" from module Airlock, and the fairness property related to it.
2. Add a variable "cleaning" to module Airlock (inside it), and specify its initial state and next state by implementing the right controller logic for it.

Furthermore, to ensure the safety and efficiency of this feature, add two properties:

1. Cleaning only happens (in the next state) if the airlock is dirty.
2. Cleaning only happens if both doors are closed.

3.3 Evacuation mode

Add an "access" mode to the airlock. The airlock can have normal access, or be in evacuation mode. In evacuation mode, nobody can enter the lab anymore. This means that **button presses from the *outside* button of each door are ignored.** This requires several modifications of the original model:

1. Buttons: You have to instantiate four buttons overall now (two related buttons for each door). 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 the button works, and remove the invariant.
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)`
   This example assumes you auto-cleaning (above); it uses "_i" and "_o" to denote the button on the *inside* and *outside* of each door, respectively. Furthermore, you need to define an access mode:
   `access: { normal, evac }; -- non-deterministic input`
3. Door functionality: The door (eventually) opens after the push of either buttons in normal mode, as before, but in "evac" mode, it only answers to the *inside* buttons (_i).
4. Access: The access mode can change non-deterministically from either mode to the other one.
5. Fairness properties: Change the existing fairness properties such that the *inside* button is eventually pressed. You may also have to introduce additional fairness properties.
6. Requirements: 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.
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 access is restricted.*
9. New requirement: The doors never open on a push of an outside button as long as the mode of the airlock is "evac".

Note: It is recommended to define auxiliary variables "inner_buttons" and "outer_buttons", whose boolean value depends on "inner_button_i" (and _o) and "outer_button_i" (and _o), respectively, and also a macro for the properties relating to the buttons. This allows you to reuse much of your code.

3.4 Lockdown mode

In addition to the mode above, define also a "lockdown" mode that closes all doors temporarily. Add "lockdown" as a third possible value of the "access" state. In this mode, **all button presses are ignored.**

1. State transitions: The state transitions for "access = normal" remain, but a possible transition from "evac" to "lockdown" has to be added (so any state is reachable from "evac"); from state "lockdown", "normal" and "lockdown" are reachable. See Figure 3.
2. Updated new requirement: The new requirement above changes to
   The doors never open on a push of an outside button as long as the mode of the airlock is "evac" or "lockdown".
3. Additional requirement: The doors never open in lockdown mode.

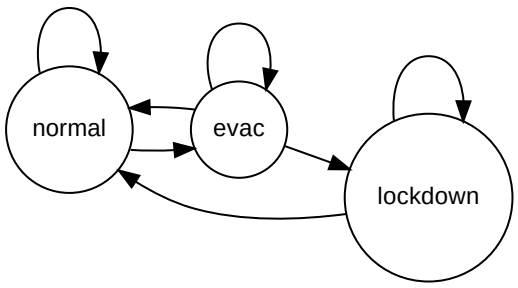

Figure 3: State transitions for the access mode.

Description	Feature	Property	Points
Extra properties		medium	+0.5
Auto-cleaning	easy	medium	+0.75
Evacuation mode	difficult	medium	+1.5
Lockdown mode	easy	easy	+0.5
Documented error trace of incorrect model.	easy		+0.25
Documented error trace of incorrect property		easy	+0.25
Simulation			+0.25

Table 1: Extra points for advanced features/properties.

You can also implement this functionality without the one above, in which case the assignment description above still applies, except that no access state "evac" is used, and both states "normal" and "lockdown" are reachable from each other.

3.5 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 `--/`.

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

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

Optional: You carry out a simulation with the correct model, to 6 steps from an initial state, and document the outcome. (+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 three deadlines: for the small lab exercises, its peer review, and for this exercise.