---------------------------------------------------------------------------

-- AN ELEVATOR CONTROLLER --

---------------------------------------------------------------------------

-- This SMV program describes an elevator system for a 4-floors building.

-- It includes modules both for the physical system (reservation buttons,

-- cabin, door), and for the controller.

---------------------------------------------------------------------------

-- BUTTON --

---------------------------------------------------------------------------

-- For each floor there is a button to request service, that can be

-- pressed. A pressed button stays pressed unless reset by the

-- controller. A button that is not pressed can become pressed

-- nondeterministically.

MODULE Button(reset)

VAR

pressed : boolean;

ASSIGN

init(pressed) := FALSE;

next(pressed) :=

case

pressed & reset : FALSE;

pressed & !reset : TRUE;

!pressed : {FALSE,TRUE};

esac;

-- REQ: The controller must not reset a button that is not pressed.

INVARSPEC (reset -> pressed)

---------------------------------------------------------------------------

-- CABIN --

---------------------------------------------------------------------------

-- The cabin can be at any floor between 1 and 4. It is equipped with an

-- engine that has a direction of motion, that can be either standing, up

-- or down. The engine can receive one of the following commands: nop, in

-- which case it does not change status; stop, in which case it becomes

-- standing; up (down), in which case it goes up (down).

MODULE Cabin(move\_cmd)

VAR

floor : { 1,2,3,4 };

direction : { standing, moving\_up, moving\_down };

ASSIGN

init(direction) := standing;

next(direction) :=

case

move\_cmd = stop : standing;

move\_cmd = move\_up : moving\_up;

move\_cmd = move\_down : moving\_down;

move\_cmd = nop : direction;

esac;

next(floor) :=

case

next(direction) = standing : floor;

next(direction) = moving\_up : case

floor = 4 : 4;

TRUE : floor + 1;

esac;

next(direction) = moving\_down : case

floor = 1 : 1;

TRUE : floor - 1;

esac;

esac;

-- REQ: The controller can issue a stop command only if the direction

-- is up or down.

INVARSPEC (move\_cmd = stop -> direction in {moving\_up,moving\_down})

-- REQ: The controller can issue a move command only if the

-- direction is standing.

INVARSPEC (move\_cmd in {move\_up,move\_down} -> direction = standing)

-- REQ: The cabin can move up only if the floor is not 4.

SPEC AG (floor = 4 -> AX(direction != moving\_up))

-- REQ: The cabin can move down only if the floor is not 1.

SPEC AG (floor = 1 -> AX(direction != moving\_down))

---------------------------------------------------------------------------

-- DOOR --

---------------------------------------------------------------------------

-- The cabin is also equipped with a door, that can be either open

-- or closed. The door can receive either open, close or nop commands

-- from the controller, and it responds opening, closing, or

-- preserving the current state.

MODULE Door(door\_cmd)

VAR

status : { open, closed };

ASSIGN

next(status) :=

case

door\_cmd = open : open;

door\_cmd = close : closed;

door\_cmd = nop : status;

esac;

-- REQ: The controller can issue an open command only if the door is closed.

INVARSPEC (door\_cmd = open -> status = closed)

-- REQ: The controller can issue a close command only if the door is open.

INVARSPEC (door\_cmd = close -> status = open)

---------------------------------------------------------------------------

-- CONTROLLER --

---------------------------------------------------------------------------

-- The controller takes in input (as sensory signals) the floor and the

-- direction of motion of the cabin, the status of the door, and the

-- status of the four buttons. It decides the controls to the engine, to

-- the door and to the buttons.

MODULE CTRL(floor, dir, door, pressed\_1, pressed\_2, pressed\_3, pressed\_4)

VAR

move\_cmd : {move\_up, move\_down, stop, nop};

door\_cmd : {open, close, nop};

reset\_1 : boolean;

reset\_2 : boolean;

reset\_3 : boolean;

reset\_4 : boolean;

-- Button N is reset only if it is pressed, we are at floor N, and

-- the door is open.

ASSIGN

reset\_1 := (pressed\_1 & floor = 1 & door = open);

reset\_2 := (pressed\_2 & floor = 2 & door = open);

reset\_3 := (pressed\_3 & floor = 3 & door = open);

reset\_4 := (pressed\_4 & floor = 4 & door = open);

-- Check whether there are pending requests at the current floor,

-- at a higher floor, and at a lower floor.

DEFINE

pending\_here := (floor = 1 & pressed\_1) | (floor = 2 & pressed\_2) |

(floor = 3 & pressed\_3) | (floor = 4 & pressed\_4) ;

pending\_up := (floor = 1 & ( pressed\_2 | pressed\_3 | pressed\_4 )) |

(floor = 2 & ( pressed\_3 | pressed\_4 )) |

(floor = 3 & ( pressed\_4 )) ;

pending\_down := (floor = 4 & ( pressed\_1 | pressed\_2 | pressed\_3 )) |

(floor = 3 & ( pressed\_1 | pressed\_2 )) |

(floor = 2 & ( pressed\_1 )) ;

-- \* If the cabin is moving, do not send commands to the door.

-- \* If there is a pending request at the current floor and

-- the door is closed, open it.

-- \* If there are pending requests at different floors and the

-- door is open, close it.

-- \* Otherwise, do not send commands to the door.

ASSIGN

door\_cmd :=

case

dir != standing : nop;

pending\_here & door = closed : open;

pending\_up & door = open : close;

pending\_down & door = open : close;

TRUE : nop;

esac;

-- Variable "last\_dir" records the last movement direction of the cabin.

VAR

last\_dir : {moving\_up,moving\_down};

ASSIGN

next(last\_dir) :=

case

dir = standing : last\_dir;

TRUE : dir;

esac;

-- \* If the door is open, do not send move commands to the cabin.

-- \* If there is a pending request at the current floor

-- and the cabin is moving, stop it.

-- \* If there are pending requests both at higher and at lower floors,

-- keep moving in "last\_dir".

-- \* If there are pending requests at higher (lower) floors,

-- move up (down).

-- \* Otherwise, do not send commands to the cabin.

ASSIGN

move\_cmd :=

case

door = open : nop;

pending\_here : case

dir != standing : stop;

TRUE : nop;

esac;

pending\_up & pending\_down : case

dir != standing : nop;

last\_dir = moving\_up : move\_up;

last\_dir = moving\_down : move\_down;

esac;

pending\_up : case

dir != standing : nop;

TRUE : move\_up;

esac;

pending\_down : case

dir != standing : nop;

TRUE : move\_down;

esac;

TRUE : nop;

esac;

---------------------------------------------------------------------------

-- MAIN --

---------------------------------------------------------------------------

-- The main module shows the connection between modules.

MODULE main

VAR

cabin : Cabin(ctrl.move\_cmd);

door : Door(ctrl.door\_cmd);

button\_1 : Button(ctrl.reset\_1);

button\_2 : Button(ctrl.reset\_2);

button\_3 : Button(ctrl.reset\_3);

button\_4 : Button(ctrl.reset\_4);

ctrl : CTRL(cabin.floor, cabin.direction, door.status,

button\_1.pressed, button\_2.pressed,

button\_3.pressed, button\_4.pressed);

---------------------------------------------------------------------------

-- REQUIREMENTS --

---------------------------------------------------------------------------

-- The controller must satisfy the following requirements.

-- REQ: No button can reach a state where it remains pressed forever.

SPEC AG AF ! button\_1.pressed

SPEC AG AF ! button\_2.pressed

SPEC AG AF ! button\_3.pressed

SPEC AG AF ! button\_4.pressed

-- REQ: No pressed button can be reset until the cabin stops at the

-- corresponding floor and opens the door.

SPEC AG (button\_1.pressed ->

A[button\_1.pressed U (cabin.floor = 1 & door.status = open)])

SPEC AG (button\_2.pressed ->

A[button\_2.pressed U (cabin.floor = 2 & door.status = open)])

SPEC AG (button\_3.pressed ->

A[button\_3.pressed U (cabin.floor = 3 & door.status = open)])

SPEC AG (button\_4.pressed ->

A[button\_4.pressed U (cabin.floor = 4 & door.status = open)])

-- REQ: A button must be reset as soon as the cabin stops at the

-- corresponding floor with the door open.

SPEC AG ((button\_1.pressed & cabin.floor = 1 & door.status = open) ->

AX ! button\_1.pressed)

SPEC AG ((button\_2.pressed & cabin.floor = 2 & door.status = open) ->

AX ! button\_2.pressed)

SPEC AG ((button\_3.pressed & cabin.floor = 3 & door.status = open) ->

AX ! button\_3.pressed)

SPEC AG ((button\_4.pressed & cabin.floor = 4 & door.status = open) ->

AX ! button\_4.pressed)

-- REQ: The cabin can move only when the door is closed.

INVARSPEC (door.status = open -> cabin.direction = standing)

-- REQ: If no button is pressed, the controller must issue no commands

-- and the cabin must be standing.

INVARSPEC (((! button\_1.pressed) & (! button\_2.pressed) &

(! button\_3.pressed) & (! button\_4.pressed))

-> (ctrl.door\_cmd = nop & ctrl.move\_cmd = nop &

cabin.direction = standing))

---------------------------------------------------------------------------