COMP2111 Assignment 4

Ken Robinson

10th May 2012

Name of assignment: LiftController
Assessment: 20 marks
Submission: give cs2111 ass4 LiftController.zip
Deadline: Friday June 1 (23:59:59)

1 Overview of assignment

This assignment extends the lecture example of a simple lift controller, that you can extract from an archive (see 2.2). The archive contains:

Lift_ctx: the basic lift context;

BasicLift: the machine that models all the basic lift movements;

Doors_ctx: the context for lift and floor doors;

LiftPlusDoors: *BasicLift* extended with lift doors;

Buttons_ctx: context for lift and door buttons;

LiftButtons: *LiftPlusDoors* extended by buttons in lifts;

1.1 Requirements

Req 1: There is a finite non-empty set of lifts.

Req 2: There is a finite non-empty set of floors.

Req 3: All lifts operate over the full set of floors

Req 4: Lifts may be one of *moving, stopped, idle*. An idle lift is inactive and must be activated before it can move.

Req 5: A moving lift may be moving *UP* or *DOWN*.

Req 5a: A lift at level 0 (lowest level) must be moving *UP*

Req 5b: A lift at the maximum level must be moving *DOWN*

Req 6: A *stopped* lift may change the direction of movement; a *moving* lift may not change direction, it must stop in order to change direction.

Req 7: Each lift has a door.

Req 8: On each floor there is a door for each lift.

Req 9: The lift door of a moving lift must be closed.

Req10: If a lift is moving then all floor doors for that lift must be closed.

Req 11: If a lift is stopped then the lift door for that lift may be open.

Req 12: If a lift is stopped and the lift door for that lift is open, then the floor door for that lift may be open.

Req 13: The lift door opens before the corresponding floor door, and the floor door closes before the corresponding lift door.

Req14: After the floor door opens there should be a delay before the floor door closes.

Req 15: Both the *lift* and *floor* doors are closed for an *idle* lift.

Req 16: Each lift is equipped with a set of *lift-buttons* corresponding to all floors for the building.

Req 17: Each lift button is either *on* or *off*.

Req 18: If a lift button is *on* then the lift must stop at the floor corresponding to that button.

Req 19: When the lift stops at a floor then the lift button corresponding to that floor must be *off*.

Req 20: Each lift should be associated with a *liftschedule* that contains all the floors at which the lift must stop.

Req 21:

a) After stopping a lift must continue on its current direction of travel if there are floors in that direction that are in the liftschedule. This strategy is intended to ensure that all lift passengers —that is passengers in a lift— experience the smallest possible number of lift stops before the lift stops at the floor requested by the lift buttons.

b) After stopping, a lift must change direction if the *liftschedule* is not empty but does not contain floors in the direction of travel.

Req 22: When a lift stops at a requested stop, then the doors must open, and remain open for an interval to allow passengers to leave the lift.

1.2 The Current State

BasicLift, *LiftPlusDoors* contain basic lift movement and safety constraints. These machines concentrate on establishing the rules for lift and door movements consistent with the above requirements, so that any further refinements will be constrained by those rules. All activities are currently nondeterministic and don't describe a useful lift system. For example: there is no reliable way of entering a lift, and once in a lift there is no assurance you will be able to get out of the lift again, let alone get out at the desired floor.

1.3 Adding Lift Buttons

Lift buttons are added to each lift, providing a capability for lift passengers to request for the lift to stop at particular floors. The modelling of the servicing of lift button requests must guarantee that all requests are serviced in the shortest possible time, as assessed across all current requests. The requirements are set ouf in section 1.1, where a simple lift schedule that provides for simple scheduling of lift button requests is described.

The refinement for *LiftButtons* shown here consists of:

LiftButtons: add lift buttons to each lift. These buttons select floors and are either ON or OFF.

Lift scheduling: at the same time as the addition of lift buttons, a lift schedule is introduced. The lift schedule records the floors to stop assigned to each lift. The requests recorded in the lift buttons are assigned to that lift's schedule.

1.4 Adding Floor Buttons

Each floor needs to be associated with buttons, which we will call *Floor Buttons* that are used by people on a floor to request a lift that will take them either *UP* or *DOWN*. Floor buttons are not associated with a lift, only a floor. Excepting the top and bottom floors, each floor will have two buttons associated respectively with *UP* and *DOWN*. Each button will be either *ON* or *OFF*. For uniformity's sake, the extreme floors can also be associated with two buttons, one of which is always *OFF*.

LiftButtons and *FloorButtons* should enforce what might be described as "passenger satisfaction policies". For *LiftButtons*, that policy is an expression of the idea that when passengers in a lift select floors –by pressing lift buttons– the lift will deliver all of those passengers to the chosen floors in the shortest journeys, subject to the context of the lift when the buttons were pressed.

1.5 Requirements for floor buttons

Req 23: Each floor is equipped with buttons for requesting a lift travelling either up or down. Obviously the bottom and top floors do not need a button for requesting a lift travelling down or up, respectively. Note that floor buttons are associated with a floor. There is no association with any particular lift.

Req 24: Every activated floor request must be scheduled. That is, at least one lift must eventually be scheduled to service each floor request. Of course, there may be more than one lift that is able to service a particular floor request.

Req 25: Scheduling must ensure that every floor request will be serviced, eventually; not necessarily optimally.

Req 26: Scheduling of floor requests must not compromise the scheduling strategy for lift button requests described in **Req 21**.

1.6 Rules for Lift System

The following rules are required to model a lift system that provides an acceptable level of service.

Buttons remain ON until the request is serviced: a button, whether a *lift button* or a *floor button* remains ON until the request implied by that button is serviced. In the case of a lift button this means the lift stops at the selected floor, and in the case of a floor button it means a lift travelling in the required direction stops at the floor.

Lifts continue as long as possible in the same direction: having stopped at a floor a lift should continue in the same direction it was travelling, if there are scheduled requests in that direction. If there are no such requests, but there are scheduled requests in the other direction then the lift will change direction and proceed using this same rule.

Lifts without scheduled requests should become IDLE: having stopped, if a lift has no scheduled requests remaining then the lift should close its doors and enter the IDLE state.

Floor requests are given to the nearest lift: as far as possible requests for a lift initiated by a floor request are assigned to the *nearest* lift, see below.

1.7 Scheduling floor buttons

Scheduling floor buttons is more difficult than scheduling lift buttons.

- when stopping at a floor to service a floor request the lift must not only be at the floor on which the request was made, but must be travelling in the required direction;
- of course, any lift stopping at a floor may service a floor request and this may render redundant the earlier scheduling of a lift to service the request, and in some circumstances the floor may be safely removed from the schedule;
- it would appear that a floor request should be assigned to the "nearest lift", but that concept is quite difficult to measure, and, given the dynamic nature of the system, is likely to be less effective than expected. Remember that the extent of a lift's travel in a particular direction is not deterministic. A lift could travel to the topmost floor or to the bottommost floor, or it may exhaust schedule requests before that happens;
- floor requests could be assigned randomly to lifts;

1.8 Policy for Floor requests

While we can construct a reasonable policy for lift requests that is capable of being implemented reasonably simply, it is more difficult for floor requests. The minimum policy is:

floor requests are scheduled in finite time and the scheduling must guarantee that the request is serviced.

The scheduling should allow for the possibility that the request could be serviced by a lift other than the intended lift.

Another possibility is that servicing of a request ma be assigned to multiple lifts, with only one lift actually carrying out the task. Scheduling for requests that are no longer required can be removed.

2 What you should do

1. First, download LiftController.zip from the class archive. A link will be found on the class webpage.
2. Create a refinement of *LiftButtons* using the Event-B explorer. Name it *Floorbuttons*.
3. Add the following to the refinement:
   New variable
   floorbuttons

Invariant
$floorbuttons \in FLOOR \rightarrow (DIRECTION \rightarrow BUTTON)$
$floorbuttons(0)(DOWN) = OFF$
$floorbuttons(MAXFLOOR)(UP) = OFF$

4. Next, read the given parts of the model, especially the *LiftButtons* machine, and understand how the model achieves the requirements. Special attention should be paid to understanding the role of the invariants and guards.

5. Devise a scheme to schedule the floor requests and develop a model based on that scheme.

6. Special attention should be paid to developing invariants that will assist in assuring that the model is behaving as your expect it to.

7. Read and consider carefully the following item on data refinement.

8. Animate: you will almost certainly find it useful to animate in order to to get an appreciation of whether your scheduling is following the type of discipline you require. Remember that animation can't be used to verify your model. Animation will not be assessed.

9. Discharge as many POs as you can.

2.1 Data refinement of liftschedule

It is almost certain that the best refinement will be achieved by data refinement. The reason is that while the current scheduling, using *liftschedule* as it stands, will not be able to be used to schedule the floor button requests, because it is not sensitive to *direction*, the fact is that you do have a scheduling strategy. What is required is to be able to extend the direction insensitive *liftschedule* to incorporate the direction sensitive requirements.

That is data refinement

If you can do that then the current framework only needs to be adapted to incorporate the data refinement.

2.2 Initiating the data refinement

Data refinement requires the following steps:

Delete liftschedule the variable *liftschedule* must be deleted. Simply delete the variable from the variables.

Do not make any other changes at the moment.

Decide how you are going to use the liftschedule strategy this involves creating new variables that you can use to schedule both *liftbutton* and *floorbutton* requests, *and*

Decide how you can extract the value of liftschedule this forms the refinement relation that relates the new set of variables to the deleted *liftschedule*.

The above exercise involves determining how you can use information from the floor requests that can be used at the appropriate time to drive the same scheduling model.

Since *liftschedule* has been deleted most of the places where you will need to *do something* will be highlighted with error diagnostics. What is required at these points is a description of how the current state relates to the deleted liftschedule. That is the verification of the refinement relation.

The above shows one of the big advantages of using data refinement: the structure of the floorbuttons machine remains significantly the same as the structure of liftbuttons. To a significant degree you only have to supply the information that allows the refinement relation to be verified.

Of course, there will be new cases that apply specifically to the function of the floor buttons.

CONTEXT Lift_ctx

SETS

$DIRECTION$
$STATUS$
$LIFT$

CONSTANTS

$MAXFLOOR$
$FLOOR$
UP
$DOWN$
$IDLE$
$STOPPED$
$MOVING$
$CHANGE$

AXIOMS

axm1 : $MAXFLOOR \in \mathbb{N}_1$
axm2 : $MAXFLOOR \geq 2$
axm3 : $FLOOR = 0 \mathinner{..} MAXFLOOR$
axm4 : $finite(LIFT)$
axm5 : $partition(DIRECTION, \{UP\}, \{DOWN\})$
axm6 : $partition(STATUS, \{IDLE\}, \{STOPPED\}, \{MOVING\})$
axm7 : $CHANGE \in DIRECTION \rightarrowtail\!\!\!\!\!\twoheadrightarrow DIRECTION$
axm8 : $CHANGE = \{UP \mapsto DOWN, DOWN \mapsto UP\}$
thm1 : $FLOOR \neq \varnothing$
thm2 : $finite(FLOOR)$
thm3 : $finite(STATUS)$
thm4 : $finite(DIRECTION)$
thm5 : $finite(CHANGE)$

END

MACHINE BasicLift

This machine models the basic lift movements,
and establishes the basic lift constraints.
The behaviour is non-deterministic:
there is no attempt to express any sort of lift control or scheduling.
A discpline of lift direction is established:
* level 0: direction is UP
* level MAXFLOOR: direction is DOWN
* other floors: either direction is valid
A lift at any time has one of the following statuses:
IDLE: not currently an active lift
STOPPED: not moving
MOVING: moving between floors
The status of a lift must be STOPPED before it becomes IDLE;
and must be STOPPED before it becomes MOVING
There are no doors.

SEES Lift_ctx

VARIABLES

$liftposition$
$liftstatus$
$liftdirection$

INVARIANTS

inv1 : $liftposition \in LIFT \rightarrow FLOOR$
thm1 : $finite(liftposition)$
inv2 : $liftstatus \in LIFT \rightarrow STATUS$
thm2 : $finite(liftstatus)$
inv3 : $liftdirection \in LIFT \rightarrow DIRECTION$
thm3 : $finite(liftdirection)$
inv4 : $\forall l \cdot l \in LIFT \wedge liftposition(l) = 0$
$\Rightarrow liftdirection(l) = UP$
inv5 : $\forall l \cdot l \in LIFT \wedge liftposition(l) = MAXFLOOR$
$\Rightarrow liftdirection(l) = DOWN$
thm4 : $\forall l \cdot l \in LIFT \wedge liftdirection(l) = DOWN$
$\Rightarrow liftposition(l) \neq 0$
thm5 : $\forall l \cdot l \in LIFT \wedge liftdirection(l) = UP$
$\Rightarrow liftposition(l) \neq MAXFLOOR$
thm6 : $\forall l \cdot l \in LIFT \wedge liftdirection(l) = UP$
$\Rightarrow liftposition(l) + 1 \leq MAXFLOOR$

EVENTS

Initialisation

begin

act1 : $liftposition := LIFT \times \{0\}$
act2 : $liftdirection := LIFT \times \{UP\}$
act3 : $liftstatus := LIFT \times \{IDLE\}$

end

Event *IdleLift* $\widehat{=}$

Idle lifts cannot move

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$

then

act1 : $liftstatus(lift) := IDLE$

end

Event *ActivateLift* $\widehat{=}$

Ready an Idle lift to enable moving

any

$lift$

where

grd1 : $liftstatus(lift) = IDLE$

then

act1 : $liftstatus(lift) := STOPPED$

end

Event *StartLift* $\widehat{=}$

Models the starting of a STOPPED lift,
maintaining previous direction

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$

then

act1 : $liftstatus(lift) := MOVING$

end

Event *ChangeDir* $\widehat{=}$

Models the changing of direction of a STOPPED lift

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $liftposition(lift) \neq 0$
grd3 : $liftposition(lift) \neq MAXFLOOR$

then

act1 : $liftdirection(lift) := CHANGE(liftdirection(lift))$

end

Event *MoveUp* $\widehat{=}$

Models a lift moving up to the next floor
where its status is either MOVING or STOPPED

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = UP$

then

act1 : $liftposition(lift) := liftposition(lift) + 1$
act2 : $liftdirection : |liftdirection' \in LIFT \to DIRECTION$
$\land (liftposition(lift) + 1 = MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto DOWN\})$
$\land (liftposition(lift) + 1 \neq MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection)$
act3 : $liftstatus : |liftstatus' \in LIFT \to STATUS \land$
$((liftstatus' = liftstatus \lhd \{lift \mapsto MOVING\})$
$\lor$
$(liftstatus' = liftstatus \lhd \{lift \mapsto STOPPED\}))$

end

Event *MoveDown* $\widehat{=}$

Models a lift moving down to the next floor
where it can stop is either MOVING or STOPPED

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = DOWN$

then

act1 : $liftposition(lift) := liftposition(lift) - 1$
act2 : $liftdirection : |liftdirection' \in LIFT \to DIRECTION$
$\land (liftposition(lift) = 1$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto UP\})$
$\land (liftposition(lift) \neq 1$
$\Rightarrow$
$liftdirection' = liftdirection)$
act3 : $liftstatus : |liftstatus' \in LIFT \to STATUS \land$
$((liftstatus' = liftstatus \lhd \{lift \mapsto MOVING\})$
$\lor$
$(liftstatus' = liftstatus \lhd \{lift \mapsto STOPPED\}))$

end

END

CONTEXT Doors_ctx
SETS

$DOORS$
CONSTANTS

$OPEN$

$CLOSED$
AXIOMS

axm1 : $partition(DOORS, \{OPEN\}, \{CLOSED\})$
END

MACHINE LiftPlusDoors

Add doors to the floors and also the lifts.
Floor doors requirements:
When a lift stops at a floor the status of the doors moves through the sequence:
CLOSED OPEN, CLOSED.
1) the lift door may be opened only if the lift status is STOPPED, and the liftdoorstatus is CLOSED
2) the floor door may be opened only if the lift status is STOPPED, the liftdoorstatus is OPEN and the floordoorstatus is CLOSED
3) the floor door may be closed only if the lift status is STOPPED, the liftdoorstatus and floordoorstatus are both OPEN
4) the lift door may be closed only if the lift status is STOPPED, the floordoorstatus is CLOSED and liftdoorstatus is OPEN
5) the lift floordoor may be OPEN only on the floor on which the lift is stopped
6) it is clear that the above door opening/closing sequence can cycle; this will be prevented by scheduling
Lift doors requirements:
1) a Lift door may be open only if
a) the lift is STOPPED, and
b) the floor door is OPEN
The lift door opens AFTER the floor door
and closes AFTER the floor door
A new event OpenDoors is introduced.
This event can only be activated when the lift status is STOPPED and will
initiate a cycle through the door opening sequence.

REFINES BasicLift
SEES Lift_ctx, Door_ctx
VARIABLES

$liftposition$
$liftstatus$
$liftdirection$
$floordoorstatus$
$liftdoorstatus$
$waiting$

INVARIANTS

inv1 : $floordoorstatus \in LIFT \rightarrow (FLOOR \rightarrow DOOR)$
inv2 : $liftdoorstatus \in LIFT \rightarrow DOOR$
inv3 : $\forall l, f \cdot f \in FLOOR \wedge f \neq liftposition(l)$
$\Rightarrow$
$floordoorstatus(l)(f) = CLOSED$
Floor doors may be OPEN only on the floor that is the current position of the lift
inv4 : $\forall l \cdot liftstatus(l) \in \{MOVING, IDLE\}$
$\Rightarrow$
$liftdoorstatus(l) = CLOSED$

inv5 : $\forall l, f \cdot liftstatus(l) \in \{MOVING, IDLE\} \land f \in FLOOR$
$\Rightarrow$
$floordoorstatus(l)(f) = CLOSED$
If a lift is MOVING or IDLE
then the lift door and all floor doors are CLOSED

inv8 : $\forall l, f \cdot f = liftposition(l) \land floordoorstatus(l)(f) = OPEN$
$\Rightarrow$
$liftdoorstatus(l) = OPEN$
Floor door may be OPEN only if the lift door is OPEN

thm1 : $\forall l, f \cdot f \in FLOOR \land floordoorstatus(l)(f) = OPEN$
$\Rightarrow$
$f = liftposition(l)$

thm2 : $\forall l \cdot liftdoorstatus(l) = OPEN$
$\Rightarrow$
$liftstatus(l) = STOPPED$

thm3 : $\forall l \cdot floordoorstatus(l)(liftposition(l)) = OPEN$
$\Rightarrow$
$liftstatus(l) = STOPPED \land liftdoorstatus(l) = OPEN$

inv6 : $waiting \subset LIFT$

inv7 : $\forall l \cdot l \in waiting$
$\Rightarrow$
$floordoorstatus(l)(liftposition(l)) = OPEN \land liftdoorstatus(l) = OPEN$
waiting is used to provide a simple model
of a house before closing doors

EVENTS
Initialisation
begin
act1 : $liftposition := LIFT \times \{0\}$
act2 : $liftdirection := LIFT \times \{UP\}$
act3 : $liftstatus := LIFT \times \{IDLE\}$
act4 : $floordoorstatus := LIFT \times \{FLOOR \times \{CLOSED\}\}$
act5 : $liftdoorstatus := LIFT \times \{CLOSED\}$
act6 : $waiting := \varnothing$
end

Event *OpenFloorDoor* $\widehat{=}$
any
$lift$
where
grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$
grd3 : $liftdoorstatus(lift) = OPEN$
then
act1 : $floordoorstatus(lift) := floordoorstatus(lift) \lhd \{liftposition(lift) \mapsto OPEN\}$
act2 : $waiting := waiting \cup \{lift\}$
end

Event *OpenLiftDoor* $\widehat{=}$
any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = OPEN$
grd3 : $liftdoorstatus(lift) = CLOSED$

then

act1 : $liftdoorstatus(lift) := OPEN$

end

Event *CloseFloorDoor* $\widehat{=}$

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = OPEN$
grd3 : $liftdoorstatus(lift) = OPEN$
grd4 : $lift \notin waiting$

then

act1 : $floordoorstatus(lift) := floordoorstatus(lift) \Leftarrow \{liftposition(lift) \mapsto CLOSED\}$

end

Event *CloseLiftdoor* $\widehat{=}$

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$
grd3 : $liftdoorstatus(lift) = OPEN$

then

act1 : $liftdoorstatus(lift) := CLOSED$

end

Event *Release* $\widehat{=}$

Models pausing between opening and closing lift doors

any

$lift$

where

grd1 : $lift \in waiting$

then

act1 : $waiting := waiting \setminus \{lift\}$

end

Event *MoveUp* $\widehat{=}$

Models a lift moving up to the next floor and continuing to move

refines *MoveUp*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = UP$

then

act1 : $liftposition(lift) := liftposition(lift) + 1$
act2 : $liftdirection : |liftdirection' \in LIFT \rightarrow DIRECTION$
$\wedge (liftposition(lift) + 1 = MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection \lhd\!\!+ \{lift \mapsto DOWN\})$
$\wedge (liftposition(lift) + 1 \neq MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection)$
act3 : $liftstatus : |liftstatus' \in LIFT \rightarrow STATUS \wedge$
$(liftstatus' = liftstatus \lhd\!\!+ \{lift \mapsto MOVING\})$

end

Event *MoveUpAndStop* $\widehat{=}$
Models a lift moving up to the next floor and stopping

refines *MoveUp*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = UP$

then

act1 : $liftposition(lift) := liftposition(lift) + 1$
act2 : $liftdirection : |liftdirection' \in LIFT \rightarrow DIRECTION$
$\wedge (liftposition(lift) + 1 = MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection \lhd\!\!+ \{lift \mapsto DOWN\})$
$\wedge (liftposition(lift) + 1 \neq MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection)$
act3 : $liftstatus(lift) := STOPPED$

end

Event *MoveDown* $\widehat{=}$
Models a lift moving down to the next floor and continueing to move

refines *MoveDown*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = DOWN$

then

act1 : $liftposition(lift) := liftposition(lift) - 1$

act2 : $liftdirection : |liftdirection' \in LIFT \to DIRECTION$
$\land (liftposition(lift) = 1$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto UP\})$
$\land (liftposition(lift) \neq 1$
$\Rightarrow$
$liftdirection' = liftdirection)$
act3 : $liftstatus : |liftstatus' \in LIFT \to STATUS \land$
$(liftstatus' = liftstatus \lhd \{lift \mapsto MOVING\})$

end

Event *MoveDownAndStop* $\widehat{=}$

Models a lift moving down to the next floor and stopping

refines *MoveDown*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = DOWN$

then

act1 : $liftposition(lift) := liftposition(lift) - 1$
act2 : $liftdirection : |liftdirection' \in LIFT \to DIRECTION$
$\land (liftposition(lift) = 1$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto UP\})$
$\land (liftposition(lift) \neq 1$
$\Rightarrow$
$liftdirection' = liftdirection)$
act3 : $liftstatus(lift) := STOPPED$

end

Event *StartLift* $\widehat{=}$

Models the starting of a STOPPED lift, maintaining of previous direction

extends *StartLift*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $liftdoorstatus(lift) = CLOSED$
grd3 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$

then

act1 : $liftstatus(lift) := MOVING$

end

Event *ChangeDir* $\widehat{=}$

Models the changing of direction of a STOPPED lift

extends *ChangeDir*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $liftposition(lift) \neq 0$
grd3 : $liftposition(lift) \neq MAXFLOOR$

then

act1 : $liftdirection(lift) := CHANGE(liftdirection(lift))$

end

Event *IdleLift* $\widehat{=}$

Idle lifts cannot move

extends *IdleLift*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$
grd3 : $liftdoorstatus(lift) = CLOSED$

then

act1 : $liftstatus(lift) := IDLE$

end

Event *ActivateLift* $\widehat{=}$

Ready an Idle lift to enable moving

extends *ActivateLift*

any

$lift$

where

grd1 : $liftstatus(lift) = IDLE$

then

act1 : $liftstatus(lift) := STOPPED$

end

END

MACHINE LiftPlusFloorDoors This machine completes the modelling of doors for a lift by introducing floor doors.
REFINES LiftPlusDoors
SEES Lift_ctx, Doors_ctx
VARIABLES

$liftposition$
$liftstatus$
$liftdirection$
$liftdoorstatus$
$floordoorstatus$

INVARIANTS

inv1 : $floordoorstatus \in LIFT \rightarrow (FLOOR \rightarrow DOORS)$
thm1 : $finite(floordoorstatus)$
inv2 : $\forall l \cdot l \in LIFT \wedge liftdoorstatus(l) = CLOSED$
$\Rightarrow$
$floordoorstatus(l)(liftposition(l)) = CLOSED$
Req 13: The floor door opens AFTER the lift door opens

inv3 : $\forall l, f \cdot l \in LIFT \wedge f \in FLOOR \setminus \{liftposition(l)\}$
$\Rightarrow$
$floordoorstatus(l)(f) = CLOSED$
Req 11: Floor doors may be OPEN only on the floor where a lift is stopped

thm2 : $\forall l, f \cdot l \in LIFT \wedge f \in FLOOR \wedge liftstatus(l) = MOVING$
$\Rightarrow$
$floordoorstatus(l)(f) = CLOSED$
Req 10: If a lift is MOVING then the floor door for that lift is CLOSED on all floors

thm3 : $\forall l \cdot l \in LIFT \wedge floordoorstatus(l)(liftposition(l)) = OPEN$
$\Rightarrow$
$liftdoorstatus(l) = OPEN$
Req 13, 14: Floor door OPEN implies lift door OPEN

inv4 : $\forall l \cdot l \in LIFT \wedge floordoorstatus(l)(liftposition(l)) = OPEN$
$\Rightarrow$
$liftstatus(l) = WAITING$
Req 11 (variant): Floor door may be OPEN only in WAITING state

EVENTS
Initialisation
extended
begin

act1 : $liftposition := LIFT \times \{0\}$
act2 : $liftdirection := LIFT \times \{UP\}$
act3 : $liftstatus := LIFT \times \{IDLE\}$
act4 : $liftdoorstatus := LIFT \times \{CLOSED\}$
act5 : $floordoorstatus := LIFT \times \{FLOOR \times \{CLOSED\}\}$

end

Event *OpenFloorDoor* $\widehat{=}$ Req 11: The floor door opens when the status of the lift is WAITING

any

$lift$
$floor$

where

grd1 : $floor = liftposition(lift)$
grd2 : $liftdoorstatus(lift) = OPEN$
grd3 : $floordoorstatus(lift)(floor) = CLOSED$
grd4 : $liftstatus(lift) = WAITING$

then

act1 : $floordoorstatus(lift) := floordoorstatus(lift) \lhd \{floor \mapsto OPEN\}$

end

Event *CloseFloorDoor* $\widehat{=}$ Req 14
refines *ChangeStatus*

any

$lift$

where

grd1 : $(floordoorstatus(lift))(liftposition(lift)) = OPEN$
grd2 : $liftstatus(lift) = WAITING$

then

act1 : $floordoorstatus(lift) := floordoorstatus(lift) \lhd \{liftposition(lift) \mapsto CLOSED\}$
act2 : $liftstatus(lift) := STOPPED$

end

Event *OpenLiftDoor* $\widehat{=}$
extends *OpenLiftDoor*

any

$lift$

where

grd1 : $liftstatus(lift) = WAITING$
grd2 : $liftdoorstatus(lift) = CLOSED$

then

act1 : $liftdoorstatus(lift) := OPEN$

end

Event *CloseLiftdoor* $\widehat{=}$
extends *CloseLiftdoor*

any

$lift$

where

grd1 : $liftdoorstatus(lift) = OPEN$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$

then

act1 : $liftdoorstatus(lift) := CLOSED$
act2 : $liftstatus(lift) := STOPPED$

end

Event *StartLift* $\widehat{=}$ Models the starting of a STOPPED lift, maintaining of previous direction

extends *StartLift*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$

grd2 : $liftdoorstatus(lift) = CLOSED$

Req 9

then

act1 : $liftstatus(lift) := MOVING$

end

Event *ChangeDir* $\widehat{=}$ Models the changing of direction of a STOPPED lift

extends *ChangeDir*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$

grd2 : $liftposition(lift) \neq 0$

grd3 : $liftposition(lift) \neq MAXFLOOR$

then

act1 : $liftdirection(lift) := CHANGE(liftdirection(lift))$

end

Event *IdleLift* $\widehat{=}$ Idle lifts cannot move

extends *IdleLift*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$

grd2 : $liftdoorstatus(lift) = CLOSED$

Req 15

then

act1 : $liftstatus(lift) := IDLE$

end

Event *ActiveateLift* $\widehat{=}$ Ready an Idle lift to enable moving

extends *ActiveateLift*

any

$lift$

where

grd1 : $liftstatus(lift) = IDLE$

then

act1 : $liftstatus : |liftstatus' \in LIFT \rightarrow STATUS$

$\wedge$

$((liftstatus' = liftstatus \lhd\!\!\!\!- \{lift \mapsto STOPPED\})$

$\vee$

$(liftstatus' = liftstatus \lhd\!\!\!\!- \{lift \mapsto WAITING\}))$

end

Event *MoveUp* $\widehat{=}$ Models a lift moving up to the next floor
extends *MoveUp*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = UP$

then

act1 : $liftposition(lift) := liftposition(lift) + 1$
act2 : $liftdirection : |liftdirection' \in LIFT \rightarrow DIRECTION$
$\wedge (liftposition(lift) + 1 = MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto DOWN\})$
$\wedge (liftposition(lift) + 1 \neq MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection)$
Req 5a and 5bReq 5a and 5b
act3 : $liftstatus : |liftstatus' \in LIFT \rightarrow STATUS \wedge$
$((liftstatus' = liftstatus \lhd \{lift \mapsto MOVING\})$
$\vee$
$(liftstatus' = liftstatus \lhd \{lift \mapsto WAITING\})$
$\vee$
$(liftstatus' = liftstatus \lhd \{lift \mapsto STOPPED\}))$

end

Event *MoveDown* $\widehat{=}$ Models a lift moving down to the next floor
extends *MoveDown*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = DOWN$

then

act1 : $liftposition(lift) := liftposition(lift) - 1$
act2 : $liftdirection : |liftdirection' \in LIFT \rightarrow DIRECTION$
$\wedge (liftposition(lift) = 1$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto UP\})$
$\wedge (liftposition(lift) \neq 1$
$\Rightarrow$
$liftdirection' = liftdirection)$
Req 5a and 5b

$$
\begin{array}{l}
\texttt{act3} : liftstatus : |liftstatus' \in LIFT \to STATUS \\
\land ((liftstatus' = liftstatus \lhd\!\!+ \{lift \mapsto MOVING\}) \\
\lor \\
(liftstatus' = liftstatus \lhd\!\!+ \{lift \mapsto WAITING\}) \\
\lor \\
(liftstatus' = liftstatus \lhd\!\!+ \{lift \mapsto STOPPED\}))
\end{array}
$$

end

END

CONTEXT Buttons_ctx
SETS

$BUTTONS$
CONSTANTS

ON

OFF
AXIOMS

axm1 : $partition(BUTTONS, \{ON\}, \{OFF\})$
Req 17
END

MACHINE LiftButtons

This machine extends the LiftPlusDoors model to

1) add buttons within each lift by which passengers indicate the floor to which they want to travel;

2) establish a lift schedule associated with each lift.
The lift schedule:
* is used to determine the direction of travel of a lift and the floors at which the lift should stop

* the lift adopts a strategy by which a lift keeps travelling in its current direction while the schedule
contains floors in that direction.
This strategy ensures satisfaction of Req21 a & b

REFINES LiftPlusDoors
SEES Lift_ctx, Door_ctx, Button_ctx
VARIABLES

$liftposition$
$liftstatus$
$liftdirection$
$liftdoorstatus$
$floordoorstatus$
$liftbuttons$
$liftschedule$
$waiting$

INVARIANTS

inv1 : $liftbuttons \in LIFT \rightarrow (FLOOR \rightarrow BUTTON)$
inv2 : $liftschedule \in LIFT \rightarrow \mathbb{P}(FLOOR)$
inv3 : $\forall l, f \cdot l \in LIFT \wedge f \in FLOOR$
$\Rightarrow$
$(liftbuttons(l)(f) = ON \Rightarrow f \in liftschedule(l))$

EVENTS
Initialisation
extended
begin

act1 : $liftposition := LIFT \times \{0\}$
act2 : $liftdirection := LIFT \times \{UP\}$
act3 : $liftstatus := LIFT \times \{IDLE\}$
act4 : $floordoorstatus := LIFT \times \{FLOOR \times \{CLOSED\}\}$
act5 : $liftdoorstatus := LIFT \times \{CLOSED\}$
act6 : $waiting := \varnothing$
act7 : $liftbuttons := LIFT \times \{FLOOR \times \{OFF\}\}$
act8 : $liftschedule := LIFT \times \{\varnothing\}$

end

Event *SelectFloor* $\widehat{=}$

Select floor to stop using lift buttons; also adds floor to liftschedule

any

$lift$
$floor$

where

grd1 : $floor \in FLOOR$
grd2 : $liftbuttons(lift)(floor) = OFF$
grd3 : $floor \neq liftposition(lift)$

then

act1 : $liftbuttons(lift) := liftbuttons(lift) \Leftarrow \{floor \mapsto ON\}$
act2 : $liftschedule(lift) := liftschedule(lift) \cup \{floor\}$

end

Event *StartLift* $\widehat{=}$

Models the starting of a STOPPED lift, maintaining of previous direction

extends *StartLift*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $liftdoorstatus(lift) = CLOSED$
grd3 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$
grd4 : $liftschedule(lift) \neq \varnothing$
grd5 : $liftdirection(lift) = DOWN$
$\Rightarrow$
$liftposition(lift) > min(liftschedule(lift))$
grd6 : $liftdirection(lift) = UP$
$\Rightarrow$
$liftposition(lift) < max(liftschedule(lift))$
grd7 : $liftposition(lift) \notin liftschedule(lift)$

then

act1 : $liftstatus(lift) := MOVING$

end

Event *ChangeDir* $\widehat{=}$

Models the changing of direction of a STOPPED lift

extends *ChangeDir*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $liftposition(lift) \neq 0$
grd3 : $liftposition(lift) \neq MAXFLOOR$
grd4 : $liftschedule(lift) \neq \varnothing$
grd5 : $(liftdirection(lift) = UP$
$\Rightarrow$
$liftposition(lift) > max(liftschedule(lift)))$
grd6 : $(liftdirection(lift) = DOWN$
$\Rightarrow$
$liftposition(lift) < min(liftschedule(lift)))$

then

act1 : $liftdirection(lift) := CHANGE(liftdirection(lift))$

end

Event *IdleLift* $\widehat{=}$

Idle lifts cannot move

extends *IdleLift*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$
grd3 : $liftdoorstatus(lift) = CLOSED$
grd4 : $liftschedule(lift) = \varnothing$

then

act1 : $liftstatus(lift) := IDLE$

end

Event *ActivateLift* $\widehat{=}$

extends *ActivateLift*

any

$lift$

where

grd1 : $liftstatus(lift) = IDLE$

then

act1 : $liftstatus(lift) := STOPPED$
act2 : $liftschedule(lift) := liftschedule(lift) \cup \{liftposition(lift)\}$

end

Event *MoveUp* $\widehat{=}$

Models a lift moving up to the next floor Next floor is not MAXFLOOR

refines *MoveUp*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = UP$
grd3 : $liftschedule(lift) \neq \varnothing$
grd4 : $liftposition(lift) < max(liftschedule(lift))$
grd5 : $liftposition(lift) + 1 \notin liftschedule(lift)$

then

act1 : $liftposition(lift) := liftposition(lift) + 1$
act2 : $liftdirection : |liftdirection' \in LIFT \to DIRECTION$
$\wedge (liftposition(lift) + 1 = MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto DOWN\})$
$\wedge (liftposition(lift) + 1 \neq MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection)$

act3 : $liftstatus(lift) := MOVING$

end

Event *MoveUpAndStop* $\widehat{=}$

Models a lift arriving at a floor and stopping

extends *MoveUpAndStop*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = UP$
grd3 : $liftposition(lift) + 1 \in liftschedule(lift)$

then

act1 : $liftposition(lift) := liftposition(lift) + 1$
act2 : $liftdirection : |liftdirection' \in LIFT \to DIRECTION$
$\wedge (liftposition(lift) + 1 = MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto DOWN\})$
$\wedge (liftposition(lift) + 1 \neq MAXFLOOR$
$\Rightarrow$
$liftdirection' = liftdirection)$
act3 : $liftstatus(lift) := STOPPED$

end

Event *MoveDown* $\widehat{=}$

Models a lift moving down to the next floor The next flloor is not floor 0

refines *MoveDown*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = DOWN$
grd3 : $liftschedule(lift) \neq \varnothing$
grd4 : $liftposition(lift) > min(liftschedule(lift))$
grd5 : $liftposition(lift) - 1 \notin liftschedule(lift)$

then

act1 : $liftposition(lift) := liftposition(lift) - 1$
act2 : $liftdirection : |liftdirection' \in LIFT \to DIRECTION$
$\wedge (liftposition(lift) - 1 \neq 0$
$\Rightarrow liftdirection' = liftdirection)$
$\wedge (liftposition(lift) - 1 = 0$
$\Rightarrow liftdirection' = liftdirection \lhd \{lift \mapsto UP\})$
act3 : $liftstatus(lift) := MOVING$

end

Event *MoveDownAndStop* $\widehat{=}$
extends *MoveDownAndStop*

any

$lift$

where

grd1 : $liftstatus(lift) = MOVING$
grd2 : $liftdirection(lift) = DOWN$
grd3 : $liftposition(lift) - 1 \in liftschedule(lift)$
grd4 : $liftposition(lift) - 1 \in liftschedule(lift)$

then

act1 : $liftposition(lift) := liftposition(lift) - 1$
act2 : $liftdirection : |liftdirection' \in LIFT \to DIRECTION$
$\wedge (liftposition(lift) = 1$
$\Rightarrow$
$liftdirection' = liftdirection \lhd \{lift \mapsto UP\})$
$\wedge (liftposition(lift) \neq 1$
$\Rightarrow$
$liftdirection' = liftdirection)$
act3 : $liftstatus(lift) := STOPPED$

end

Event *OpenFloorDoor* $\widehat{=}$
extends *OpenFloorDoor*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$
grd3 : $liftdoorstatus(lift) = OPEN$
grd4 : $liftposition(lift) \in liftschedule(lift)$

then

act1 : $floordoorstatus(lift) := floordoorstatus(lift) \lhd \{liftposition(lift) \mapsto OPEN\}$
act2 : $waiting := waiting \cup \{lift\}$

end

Event *CloseFloorDoor* $\widehat{=}$
extends *CloseFloorDoor*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = OPEN$
grd3 : $liftdoorstatus(lift) = OPEN$
grd4 : $lift \notin waiting$

then

act1 : $floordoorstatus(lift) := floordoorstatus(lift) \lhd \{liftposition(lift) \mapsto CLOSED\}$

end

Event *OpenLiftDoor* $\widehat{=}$
extends *OpenLiftDoor*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = OPEN$
grd3 : $liftdoorstatus(lift) = CLOSED$
grd4 : $liftposition(lift) \in liftschedule(lift)$

then

act1 : $liftdoorstatus(lift) := OPEN$

end

Event *CloseLiftdoor* $\widehat{=}$
extends *CloseLiftdoor*

any

$lift$

where

grd1 : $liftstatus(lift) = STOPPED$
grd2 : $floordoorstatus(lift)(liftposition(lift)) = CLOSED$
grd3 : $liftdoorstatus(lift) = OPEN$
grd4 : $liftposition(lift) \in liftschedule(lift)$

then

act1 : $liftdoorstatus(lift) := CLOSED$
act2 : $liftbuttons(lift) := (liftbuttons(lift) \lhd \{liftposition(lift) \mapsto OFF\})$
act3 : $liftschedule(lift) := liftschedule(lift) \setminus \{liftposition(lift)\}$

end

Event *Release* $\widehat{=}$
extends *Release*

any

$lift$

where

grd1 : $lift \in waiting$

then

act1 : $waiting := waiting \setminus \{lift\}$

end

END