1 Before you start this assignment

The requirement of this assignment is to extend an event B model of a lift control system to add floor buttons, by which passengers requests a lift on a particular floor to go either up or down.

1.1 What is a lift?

It may be more difficult than you think to decide exactly “what is a lift”. For example:

1. Are buttons essential to a lift system? It is possible to imagine a lift system that doesn’t need buttons: it traverses all floors of the building and stops at each floor.


1.2 Refinement

Our aim in this development of a model of a lift control system is to build layers of refinements, each of which deals with one aspect of a lift.

2 Overview of assignment

This assignment extends the lecture example of a simple lift controller, that you can extract from an archive (see 2.9). The archive contains:

- Lift_ctx: the basic lift context;
- BasicLift: the machine that models all the basic lift movements;
- Door_ctx: the context for lift and floor doors;
- LiftPlusDoors: basic lift extended with lift doors;
- Button_ctx: context for lift and floor buttons;
LiftButtons: lift extended by buttons in lifts;

FloorButtons: further extended by the addition of buttons on floors.

2.1 Requirements

Note: The following requirements still apply, but the annotations of the machines and contexts have been lost in a revision.

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.

Req 10: If a lift is moving then all floor doors for that lift must be closed.

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

Req 12: Lift and floor doors may be open for a stopped lift.

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

Req 14: The floor door closes before the corresponding lift door.

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 lift schedule that contains all the floors at which the lift must stop.
Req 21a: After stopping a lift must continue on its current direction of travel if there are floors in that direction of travel that are in the lift schedule. This strategy is intended to ensure that each lift passenger—that is passengers in the lift—experiences the smallest number of lift stops before their requested stop.

Req 21b: After stopping, a lift must change direction if the lift schedule 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.

2.2 The Current State

BasicLift and 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.

2.3 Lift Buttons and lift scheduling

Each lift has a set of buttons, by which passengers select the floor that they wish the lift to stop at. Introducing lift and floor buttons requires that lift movement is no longer completely random, and there must be some scheduling policy that ensures that passenger requests are satisfied. 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 automatically assigned to that lift’s schedule.

2.4 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.
2.5 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.

Req 24: Every activated floor request must be scheduled. That is, at least one list must 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.

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 a passenger in a lift selects a floor—by pressing one of the lift buttons—the lift will deliver the passenger to that floor in the shortest journey, subject to the context of the lift when the button is pressed.

2.6 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 direction. 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 sake the extreme floors can also be associated with two buttons, one of which is always OFF.

2.7 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 bottom floor, or it may exhaust schedule requests before that happens;

• floor requests could be assigned randomly to lifts;
2.8 Scheduling for Floor requests

The simple strategy of a single liftschedule works very well and while it obviously will not work (without some modification) for floor button scheduling, it would seem a good strategy to use the same principles for floor button scheduling. It would seem an even better idea to merge the scheduling of lift buttons and floor buttons.

The obvious complication with floor buttons is that a lift that stops in order to service a floor request must be travelling the requested direction. This would appear to require two schedules: an UP schedule and a DOWN schedule.

2.9 Floor button scheduling solution

An elegant solution is to data refine liftschedule as follows:

1. To each lift assign an upschedule and a downschedule. The upschedule is a set of floors on which a lift should stop if the lift is travelling UP. Similarly for downschedule.

2. Determine a refinement relation that relates upschedule and downschedule to liftschedule.

3. This should allow the current scheduling to be reused to schedule both lift- and floor- requests.

- 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 multiple lifts may be assigned to service the request, with only one lift actually carrying out the task.

IMPORTANT Floor requests must be assigned immediately to a lift, otherwise there can be no guarantee that the request will be serviced eventually.

There is no requirement that the servicing of requests is optimal, whatever that might mean.
CONTEXT \text{Lift}\_\text{ctx}

SETS

\text{DIRECTION}
\text{STATUS}
\text{LIFT}

CONSTANTS

\text{MAXFLOOR}
\text{FLOOR}
\text{UP}
\text{DOWN}
\text{IDLE}
\text{STOPPED}
\text{MOVING}
\text{CHANGE}

AXIOMS

\text{axm1: } \text{MAXFLOOR} \in \mathbb{N}_1
\text{axm2: } \text{MAXFLOOR} \geq 2
\text{axm3: } \text{FLOOR} = 0 .. \text{MAXFLOOR}
\text{axm4: } \text{finite(LIFT)}
\text{axm5: } \text{partition(DIRECTION, \{UP\}, \{DOWN\})}
\text{axm6: } \text{partition(STATUS, \{IDLE\}, \{STOPPED\}, \{MOVING\})}
\text{axm7: } \text{CHANGE} \in \text{DIRECTION} \rightarrow \text{DIRECTION}
\text{axm8: } \text{CHANGE} = \{ \text{UP} \rightarrow \text{DOWN}, \text{DOWN} \rightarrow \text{UP} \}

\text{thm1: } \text{FLOOR} \neq \emptyset
\text{thm2: } \text{finite(FLOOR)}
\text{thm3: } \text{finite(STATUS)}
\text{thm4: } \text{finite(DIRECTION)}
\text{thm5: } \text{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 discipline 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

There are no doors.

SEES Lift_ctx

VARIABLES

liftposition
liftstatus
liftdirection

INvariants

\( \text{inv1: } \text{liftposition} \in LIFT \to FLOOR \)

\( \text{thm1: } \text{finite(liftposition)} \)

\( \text{inv2: } \text{liftstatus} \in LIFT \to STATUS \)

\( \text{thm2: } \text{finite(liftstatus)} \)

\( \text{inv3: } \text{liftdirection} \in LIFT \to DIRECTION \)

\( \text{thm3: } \text{finite(liftdirection)} \)

\( \text{inv4: } \forall l \cdot l \in LIFT \land \text{liftposition}(l) = 0 \)
\( \Rightarrow \text{liftdirection}(l) = \text{UP} \)

\( \text{inv5: } \forall l \cdot l \in LIFT \land \text{liftposition}(l) = \text{MAXFLOOR} \)
\( \Rightarrow \text{liftdirection}(l) = \text{DOWN} \)

\( \text{thm4: } \forall l \cdot l \in LIFT \land \text{liftdirection}(l) = \text{DOWN} \)
\( \Rightarrow \text{liftposition}(l) \neq 0 \)

\( \text{thm5: } \forall l \cdot l \in LIFT \land \text{liftdirection}(l) = \text{UP} \)
\( \Rightarrow \text{liftposition}(l) \neq \text{MAXFLOOR} \)

\( \text{thm6: } \forall l \cdot l \in LIFT \land \text{liftdirection}(l) = \text{UP} \)
\( \Rightarrow \text{liftposition}(l) + 1 \leq \text{MAXFLOOR} \)

EVENTS
Initialisation

\begin{align*}
\text{begin} & \quad \text{act} 1: \quad \text{liftposition} := LIFT \times \{0\} \\
& \quad \text{act} 2: \quad \text{liftdirection} := LIFT \times \{UP\} \\
& \quad \text{act} 3: \quad \text{liftstatus} := LIFT \times \{IDLE\} \\
\text{end}
\end{align*}

Event \textit{IdleLift} $\triangleq$ Idle lifts cannot move

\begin{align*}
\text{any lift} \\
\text{when grd} 1: \quad \text{liftstatus}(\text{lift}) = STOPPED \\
\text{then act} 1: \quad \text{liftstatus}(\text{lift}) := IDLE \\
\text{end}
\end{align*}

Event \textit{ActivateLift} $\triangleq$ Ready an Idle lift to enable moving

\begin{align*}
\text{any lift} \\
\text{when grd} 1: \quad \text{liftstatus}(\text{lift}) = IDLE \\
\text{then act} 1: \quad \text{liftstatus}(\text{lift}) := STOPPED \\
\text{end}
\end{align*}

Event \textit{StartLift} $\triangleq$ Models the starting of a STOPPED lift, maintaining of previous direction

\begin{align*}
\text{any lift} \\
\text{when grd} 1: \quad \text{liftstatus}(\text{lift}) = STOPPED \\
\text{then act} 1: \quad \text{liftstatus}(\text{lift}) := MOVING \\
\text{end}
\end{align*}

Event \textit{ChangeDir} $\triangleq$ Models the changing of direction of a STOPPED lift

\begin{align*}
\text{any lift} \\
\text{when grd} 1: \quad \text{liftstatus}(\text{lift}) = STOPPED \\
\text{grd} 2: \quad \text{liftposition}(\text{lift}) \neq 0 \\
\text{grd} 3: \quad \text{liftposition}(\text{lift}) \neq \text{MAXFLOOR} \\
\text{then act} 1: \quad \text{liftdirection}(\text{lift}) := \text{CHANGE} (\text{liftdirection}(\text{lift})) \\
\text{end}
\end{align*}

Event \textit{MoveUp} $\triangleq$ Models a lift moving up to the next floor

\begin{align*}
\text{any lift} \\
\text{when grd} 1: \quad \text{liftstatus}(\text{lift}) = STOPPED \\
\text{then act} 1: \quad \text{liftdirection}(\text{lift}) := \text{CHANGE} (\text{liftdirection}(\text{lift})) \\
\text{end}
\end{align*}
any lift

when grd1: liftdirection(lift) = UP
        
        grd2: liftdirection(lift) = DOWN

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

act2: liftdirection : |liftdirection' ∈ LIFT → DIRECTION
       \∧ (liftposition(lift) = 1
       \⇒ liftdirection' = liftdirection \Leftrightarrow \{lift \mapsto UP\})
       \∧ (liftposition(lift) \neq 1
       \⇒ liftdirection' = liftdirection)

act3: liftdirection : |liftdirection' ∈ LIFT → DIRECTION
       \∧ (liftposition(lift) = 1
       \⇒ liftdirection' = liftdirection \Leftrightarrow \{lift \mapsto UP\})
       \∧ (liftposition(lift) \neq 1
       \⇒ liftdirection' = liftdirection)

act3: liftdirection : |liftdirection' ∈ LIFT → DIRECTION
       \∧ (liftposition(lift) = 1
       \⇒ liftdirection' = liftdirection \Leftrightarrow \{lift \mapsto UP\})
       \∧ (liftposition(lift) \neq 1
       \⇒ liftdirection' = liftdirection)

end

Event MoveDown ≡

Models a lift moving down to the next floor
MACHINE LiftPlusDoors
REFINES BasicLift
SEES Lift_ctx, Door_ctx

VARIABLES

liftposition
liftstatus
liftdirection
floordoorstatus
liftdoorstatus
waiting

INVARIANTS

inv1: floordoorstatus ∈ LIFT → (FLOOR → DOOR)
inv2: liftdoorstatus ∈ LIFT → DOOR
inv3: ∀l, f ∈ FLOOR ∧ f ≠ liftposition(l) ⇒ floordoorstatus(l)(f) = CLOSED
Floor doors may be OPEN only on the floor that is the current position of the lift
inv4: ∀l · liftstatus(l) ∈ {MOVING, IDLE} ⇒ liftdoorstatus(l) = CLOSED
inv5: ∀l, f · liftstatus(l) ∈ {MOVING, IDLE} ∧ f ∈ FLOOR ⇒ floordoorstatus(l)(f) = CLOSED
If a lift is MOVING or IDLE then the lift door and all floor doors are CLOSED

thm1: ∀l, f · f ∈ FLOOR ∧ floordoorstatus(l)(f) = OPEN ⇒ f = liftposition(l)
thm2: ∀l · liftdoorstatus(l) = OPEN ⇒ liftstatus(l) = STOPPED
thm3: ∀l · floordoorstatus(l)(liftposition(l)) = OPEN ⇒ liftstatus(l) = STOPPED

inv6: waiting ⊆ LIFT
inv7: ∀l · l ∈ waiting ⇒ floordoorstatus(l)(liftposition(l)) = OPEN ∧ liftdoorstatus(l) = OPEN
waiting is used to provide a simple model of a pause before closing doors
EVENTS

Initialisation

begin
  act1: liftposition := LIFT × {0}
  act2: liftdirection := LIFT × {UP}
  act3: liftstatus := LIFT × {IDLE}
  act4: floordoorstatus := LIFT × {FLOOR × {CLOSED}}
  act5: liftdoorstatus := LIFT × {CLOSED}
  act6: waiting := ∅
end

Event OpenFloorDoor ≜

any lift
when grd1: liftstatus(lift) = STOPPED
  grd2: floordoorstatus(lift)(liftposition(lift)) = CLOSED
  grd3: liftdoorstatus(lift) = CLOSED
then
  act1: floordoorstatus(lift) := floordoorstatus(lift) ▷ {liftposition(lift) ↦ OPEN}
end

Event OpenLiftDoor ≜

any lift
when grd1: liftstatus(lift) = STOPPED
  grd2: floordoorstatus(lift)(liftposition(lift)) = OPEN
  grd3: liftdoorstatus(lift) = CLOSED
then
  act1: liftdoorstatus(lift) := OPEN
  act2: waiting := waiting ∪ {lift}
end

Event CloseFloorDoor ≜

any lift
when grd1: liftstatus(lift) = STOPPED
  grd2: floordoorstatus(lift)(liftposition(lift)) = OPEN
  grd3: liftdoorstatus(lift) = OPEN
  grd4: lift /∈ waiting
then
  act1: floordoorstatus(lift) := floordoorstatus(lift) ▷ {liftposition(lift) ↦ CLOSED}
end

Event CloseLiftdoor ≜
any lift
when grd1: liftstatus(lift) = STOPPED
  grd2: floordoorstatus(lift)(liftposition(lift)) = CLOSED
  grd3: liftdoorstatus(lift) = OPEN
then act1: liftdoorstatus(lift) := CLOSED
end

Event Release ≜ Models pausing between opening and closing lift doors

any lift
when grd1: lift ∈ waiting
then act1: waiting := waiting \ {lift}
end

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

refines MoveUp

any lift
when grd1: liftstatus(lift) = MOVING
  grd2: liftdirection(lift) = UP
then act1: liftposition(lift) := liftposition(lift) + 1
  act2: liftdirection : |liftdirection'| ∈ LIFT → DIRECTION
        ∧ (liftposition(lift) + 1 = MAXFLOOR
            ⇒ liftdirection' = liftdirection \ {lift↦DOWN})
        ∧ (liftposition(lift) + 1 ≠ MAXFLOOR
            ⇒ liftdirection' = liftdirection)
  act3: liftstatus : |liftstatus'| ∈ LIFT → STATUS ∧
        (liftstatus' = liftstatus \ {lift↦MOVING})
end

Event MoveUpAndStop ≜ Models a lift moving up to the next floor and stopping

refines MoveUp

any lift
when grd1: liftstatus(lift) = MOVING
  grd2: liftdirection(lift) = UP
then act1: liftposition(lift) := liftposition(lift) + 1
act2: \[ \text{liftdirection} : |\text{liftdirection}' \in \text{LIFT} \to \text{DIRECTION} \\
\land \left( \text{liftposition(lift)} + 1 = \text{MAXFLOOR} \right) \\
\Rightarrow \\
\text{liftdirection}' = \text{liftdirection} \Leftrightarrow \{\text{lift} \mapsto \text{DOWN}\} \\
\land \left( \text{liftposition(lift)} + 1 \neq \text{MAXFLOOR} \right) \\
\Rightarrow \\
\text{liftdirection}' = \text{liftdirection} \]

act3: \[ \text{liftstatus(lift)} := \text{STOPPED} \]

\text{end}

\text{Event \ MoveDown} \equiv \text{Models a lift moving down to the next floor and continuing to move}

\text{refines \ MoveDown}

\text{any lift}

\text{when \ grd1: liftstatus(lift) = MOVING}

\text{grd2: liftdirection(lift) = DOWN}

\text{then act1: liftposition(lift) := liftposition(lift) - 1}

\text{act2: liftdirection : |liftdirection' \in \text{LIFT} \to \text{DIRECTION} \\
\land (\text{liftposition(lift)} = 1) \\
\Rightarrow \\
\text{liftdirection}' = \text{liftdirection} \Leftrightarrow \{\text{lift} \mapsto \text{UP}\} \\
\land (\text{liftposition(lift)} \neq 1) \\
\Rightarrow \\
\text{liftdirection}' = \text{liftdirection} \]

\text{act3: liftstatus(lift) := STOPPED}

\text{end}

\text{Event \ MoveDownAndStop} \equiv \text{Models a lift moving down to the next floor and stopping}

\text{refines \ MoveDown}

\text{any lift}

\text{when \ grd1: liftstatus(lift) = MOVING}

\text{grd2: liftdirection(lift) = DOWN}

\text{then act1: liftposition(lift) := liftposition(lift) - 1}

\text{act2: liftdirection : |liftdirection' \in \text{LIFT} \to \text{DIRECTION} \\
\land (\text{liftposition(lift)} = 1) \\
\Rightarrow \\
\text{liftdirection}' = \text{liftdirection} \Leftrightarrow \{\text{lift} \mapsto \text{UP}\} \\
\land (\text{liftposition(lift)} \neq 1) \\
\Rightarrow \\
\text{liftdirection}' = \text{liftdirection} \]

\text{act3: liftstatus(lift) := STOPPED
Event \textit{StartLift} \equiv \text{Models the starting of a STOPPED lift, maintaining of previous direction}

\textbf{extends} \textit{StartLift}

\textbf{any} lift

\textbf{when} \textbf{grd1}: liftstatus(lift) = STOPPED
\hspace{1em} \textbf{grd2}: liftdoorstatus(lift) = CLOSED
\hspace{1em} \textbf{grd3}: floordoorstatus(lift)(liftposition(lift)) = CLOSED
\textbf{then} \textbf{act1}: liftstatus(lift) := MOVING

end

Event \textit{ChangeDir} \equiv \text{Models the changing of direction of a STOPPED lift}

\textbf{extends} \textit{ChangeDir}

\textbf{any} lift

\textbf{when} \textbf{grd1}: liftstatus(lift) = STOPPED
\hspace{1em} \textbf{grd2}: liftposition(lift) \neq 0
\hspace{1em} \textbf{grd3}: liftposition(lift) \neq MAXFLOOR
\textbf{then} \textbf{act1}: liftdirection(lift) := CHANGE(liftdirection(lift))

end

Event \textit{IdleLift} \equiv \text{Idle lifts cannot move}

\textbf{extends} \textit{IdleLift}

\textbf{any} lift

\textbf{when} \textbf{grd1}: liftstatus(lift) = STOPPED
\hspace{1em} \textbf{grd2}: floordoorstatus(lift)(liftposition(lift)) = CLOSED
\hspace{1em} \textbf{grd3}: liftdoorstatus(lift) = CLOSED
\textbf{then} \textbf{act1}: liftstatus(lift) := IDLE

end

Event \textit{ActivateLift} \equiv \text{Ready an Idle lift to enable moving}

\textbf{extends} \textit{ActivateLift}

\textbf{any} lift

\textbf{when} \textbf{grd1}: liftstatus(lift) = IDLE
\hspace{1em} \textbf{then} \textbf{act1}: liftstatus(lift) := STOPPED
end

END
CONTEXT Buttons_ctx

SETS

| BUTTONS |

CONSTANTS

| ON   |
| OFF  |

AXIOMS

\[ axm1: \text{partition}(BUTTONS, \{ON\}, \{OFF\}) \]  

END

Req 17
MACHINE LiftButtons
This machine extends the model to
1) add buttons within each lift by which passengers indicate the floor to which they want to
tavel;
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.

REFINES LiftPlusDoors
SEES Lift_ctx, Door_ctx, Button_ctx

VARIABLES

liftposition
liftstatus
liftdirection
liftdoorstatus
floordoorstatus
liftbuttons
liftschedule
waiting

INVARINANTS

\begin{align*}
inv1: & \quad \text{liftbuttons} \in \text{LIFT} \rightarrow (\text{FLOOR} \rightarrow \text{BUTTON}) \\
inv2: & \quad \text{liftschedule} \in \text{LIFT} \rightarrow \mathcal{P}(\text{FLOOR}) \\
inv3: & \quad \forall l,f \cdot l \in \text{LIFT} \land f \in \text{FLOOR} \\
& \quad \Rightarrow \\
& \quad (\text{liftbuttons}(l)(f) = \text{ON} \Rightarrow f \in \text{liftschedule}(l))
\end{align*}

EVENTS

Initialisation

\begin{align*}
\text{extended} \\
\text{begin} & \quad \text{act1: liftposition} := \text{LIFT} \times \{0\} \\
& \quad \text{act2: liftdirection} := \text{LIFT} \times \{\text{UP}\} \\
& \quad \text{act3: liftstatus} := \text{LIFT} \times \{\text{IDLE}\} \\
& \quad \text{act4: floordoorstatus} := \text{LIFT} \times (\text{FLOOR} \times \{\text{CLOSED}\})
\end{align*}
act5: liftdoorstatus := LIFT \times \{CLOSED\}
act6: waiting := \emptyset
act7: liftbuttons := LIFT \times \{FLOOR \times \{OFF\}\}
act8: liftschedule := LIFT \times \{\emptyset\}

Event  SelectFloor \equiv  Select floor to stop using lift buttons; also adds floor to lift schedule

any lift
floor
when \text{grd1}:  \text{floor} \in \text{FLOOR}
  \text{grd2}:  \text{liftbuttons}(\text{lift})(\text{floor}) = \text{OFF}
  \text{grd3}:  \text{floor} \neq \text{liftposition}(\text{lift})
then act1:  \text{liftbuttons}(\text{lift}) := \text{liftbuttons}(\text{lift}) \ominus \{\text{floor} \mapsto \text{ON}\}
  act2:  \text{liftschedule}(\text{lift}) := \text{liftschedule}(\text{lift}) \cup \{\text{floor}\}

Event  StartLift \equiv  Models the starting of a STOPPED lift, maintaining of previous direction

extends StartLift

any lift
when \text{grd1}:  \text{liftstatus}(\text{lift}) = \text{STOPPED}
  \text{grd2}:  \text{liftdoorstatus}(\text{lift}) = \text{CLOSED}
  \text{grd3}:  \text{floordoorstatus}(\text{lift})(\text{liftposition}(\text{lift})) = \text{CLOSED}
  \text{grd4}:  \text{liftschedule}(\text{lift}) \neq \emptyset
  \text{grd5}:  \text{liftdirection}(\text{lift}) = \text{DOWN}
  \quad \Rightarrow
    \text{liftposition}(\text{lift}) > \min(\text{liftschedule}(\text{lift}))
  \text{grd6}:  \text{liftdirection}(\text{lift}) = \text{UP}
  \quad \Rightarrow
    \text{liftposition}(\text{lift}) < \max(\text{liftschedule}(\text{lift}))
  \text{grd7}:  \text{liftposition}(\text{lift}) \notin \text{liftschedule}(\text{lift})
then act1:  \text{liftstatus}(\text{lift}) := \text{MOVING}

Event  ChangeDir \equiv  Models the changing of direction of a STOPPED lift

extends ChangeDir

any lift
when \text{grd1}:  \text{liftstatus}(\text{lift}) = \text{STOPPED}
\textbf{grd2:} lift\textit{position}(\textit{lift}) \neq 0 \\
\textbf{grd3:} lift\textit{position}(\textit{lift}) \neq \text{MAXFLOOR} \\
\textbf{grd4:} lift\textit{schedule}(\textit{lift}) \neq \emptyset \\
\textbf{grd5:} (lift\textit{direction}(\textit{lift}) = \text{UP})  \\
\textbf{grd6:} (lift\textit{direction}(\textit{lift}) = \text{DOWN}) \\
\textbf{then act1:} lift\textit{direction}(\textit{lift}) := \text{CHANGE}(lift\textit{direction}(\textit{lift})) \\
\textbf{end} \\

**Event** \textit{IdleLift} $\triangleq$ Idle lifts cannot move \\
\textbf{extends} \textit{IdleLift} \\
\textbf{any} \textit{lift} \\
\textbf{when grd1:} lift\textit{status}(\textit{lift}) = \text{STOPPED} \\
\textbf{grd2:} floordoor\textit{status}(\textit{lift})(\textit{lift\textit{position}(\textit{lift})) = \text{CLOSED} \\
\textbf{grd3:} liftdoor\textit{status}(\textit{lift}) = \text{CLOSED} \\
\textbf{grd4:} lift\textit{schedule}(\textit{lift}) = \emptyset \\
\textbf{then act1:} lift\textit{status}(\textit{lift}) := \text{IDLE} \\
\textbf{end} \\

**Event** \textit{ActivateLift} $\triangleq$ \\
\textbf{extends} \textit{ActivateLift} \\
\textbf{any} \textit{lift} \\
\textbf{when grd1:} lift\textit{status}(\textit{lift}) = \text{IDLE} \\
\textbf{then act1:} lift\textit{status}(\textit{lift}) := \text{STOPPED} \\
\textbf{act2:} lift\textit{schedule}(\textit{lift}) := lift\textit{schedule}(\textit{lift}) \cup \{\textit{lift\textit{position}(\textit{lift})}\} \\
\textbf{end} \\

**Event** \textit{MoveUp} $\triangleq$ Models a lift moving up to the next floor Next floor is not MAXFLOOR \\
\textbf{refines} \textit{MoveUp} \\
\textbf{any} \textit{lift} \\
\textbf{when grd1:} lift\textit{status}(\textit{lift}) = \text{MOVING} \\
\textbf{grd2:} lift\textit{direction}(\textit{lift}) = \text{UP} \\
\textbf{grd3:} lift\textit{schedule}(\textit{lift}) \neq \emptyset \\
\textbf{grd4:} lift\textit{position}(\textit{lift}) < \text{max}(lift\textit{schedule}(\textit{lift})) \\

19
\begin{align*}
\text{grd5: } & \text{liftposition}(lift) + 1 \notin \text{liftschedule}(lift) \\
\text{then act1: } & \text{liftposition}(lift) := \text{liftposition}(lift) + 1 \\
\text{act2: } & \text{liftdirection}(lift) = \text{liftdirection'} \in \text{LIFT} \rightarrow \text{DIRECTION} \\
& \quad \land (\text{liftposition}(lift) + 1 = \text{MAXFLOOR}) \\
& \quad \Rightarrow \text{liftdirection'} = \text{liftdirection} \leftarrow \{\text{lift} \mapsto \text{DOWN}\} \\
& \quad \land (\text{liftposition}(lift) + 1 \neq \text{MAXFLOOR}) \\
& \quad \Rightarrow \text{liftdirection'} = \text{liftdirection} \\
\text{act3: } & \text{liftstatus}(lift) := \text{STOPPED}
\end{align*}

\text{Event \textit{MoveUpAndStop} = Models a lift arriving at a floor and stopping}

\text{extends \textit{MoveUpAndStop}}

\begin{align*}
\text{any lift} \\
\text{when grd1: } & \text{liftstatus}(lift) = \text{MOVING} \\
& \text{grd2: } \text{liftdirection}(lift) = \text{UP} \\
& \text{grd3: } \text{liftposition}(lift) + 1 \in \text{liftschedule}(lift) \\
\text{then act1: } & \text{liftposition}(lift) := \text{liftposition}(lift) + 1 \\
\text{act2: } & \text{liftdirection}(lift) = \text{liftdirection'} \in \text{LIFT} \rightarrow \text{DIRECTION} \\
& \quad \land (\text{liftposition}(lift) + 1 = \text{MAXFLOOR}) \\
& \quad \Rightarrow \text{liftdirection'} = \text{liftdirection} \leftarrow \{\text{lift} \mapsto \text{DOWN}\} \\
& \quad \land (\text{liftposition}(lift) + 1 \neq \text{MAXFLOOR}) \\
& \quad \Rightarrow \text{liftdirection'} = \text{liftdirection} \\
\text{act3: } & \text{liftstatus}(lift) := \text{STOPPED}
\end{align*}

\text{Event \textit{MoveDown} = Models a lift moving down to the next floor The next floor is not floor 0}

\text{refines \textit{MoveDown}}

\begin{align*}
\text{any lift} \\
\text{when grd1: } & \text{liftstatus}(lift) = \text{MOVING} \\
& \text{grd2: } \text{liftdirection}(lift) = \text{DOWN} \\
& \text{grd3: } \text{liftschedule}(lift) \neq \emptyset \\
& \text{grd4: } \text{liftposition}(lift) > \text{min}\left(\text{liftschedule}(lift)\right) \\
& \text{grd5: } \text{liftposition}(lift) - 1 \notin \text{liftschedule}(lift) \\
\text{then act1: } & \text{liftposition}(lift) := \text{liftposition}(lift) - 1
\end{align*}
\( \text{act2: } \) liftdirection : \( \{ \text{liftdirection}' \in LIFT \to DIRECTION \) \\
\( \land (\text{liftposition}(\text{lift}) - 1 \neq 0) \) \\
\( \Rightarrow \text{liftdirection}' = \text{liftdirection} \) \\
\( \land (\text{liftposition}(\text{lift}) - 1 = 0) \) \\
\( \Rightarrow \text{liftdirection}' = \text{liftdirection} \Leftrightarrow \{ \text{lift} \mapsto \text{UP} \} \) \\

\( \text{act3: } \) liftstatus(\text{lift}) := MOVING \\

end

\textbf{Event } MoveDownAndStop \equiv \\
\textbf{extends } MoveDownAndStop \\
\textbf{any } \text{lift} \\
\textbf{when } \text{grd1: } \text{liftstatus}(\text{lift}) = MOVING \\
\text{grd2: } \text{liftdirection}(\text{lift}) = DOWN \\
\text{grd3: } \text{liftposition}(\text{lift}) - 1 \in \text{liftschedule}(\text{lift}) \\
\text{grd4: } \text{liftposition}(\text{lift}) - 1 \in \text{liftschedule}(\text{lift}) \\
\textbf{then } \text{act1: } \text{liftposition}(\text{lift}) := \text{liftposition}(\text{lift}) - 1 \\
\text{act2: } \text{liftdirection} : \{ \text{liftdirection}' \in LIFT \to DIRECTION \) \\
\( \land (\text{liftposition}(\text{lift}) = 1) \) \\
\( \Rightarrow \) \\
\( \text{liftdirection}' = \text{liftdirection} \Leftrightarrow \{ \text{lift} \mapsto \text{UP} \} \) \\
\( \land (\text{liftposition}(\text{lift}) \neq 1) \) \\
\( \Rightarrow \) \\
\( \text{liftdirection}' = \text{liftdirection} \) \\
\text{act3: } \text{liftstatus}(\text{lift}) := STOPPED \\

end

\textbf{Event } IdleToWaiting \equiv \text{Ready an Idle lift to enable moving} \\
\textbf{extends } ActivateLift \\
\textbf{any } \text{lift} \\
\textbf{when } \text{grd1: } \text{liftstatus}(\text{lift}) = IDLE \\
\textbf{then } \text{act1: } \text{liftstatus}(\text{lift}) := STOPPED \\
\text{act2: } \text{liftschedule}(\text{lift}) := \text{liftschedule}(\text{lift}) \cup \{ \text{liftposition}(\text{lift}) \} \\

end

\textbf{Event } OpenFloorDoor \equiv \\
\textbf{extends } OpenFloorDoor \\
\textbf{any } \text{lift} \\
\textbf{when } \text{grd1: } \text{liftstatus}(\text{lift}) = STOPPED \\
\text{grd2: } \text{floordoorstatus}(\text{lift})(\text{liftposition}(\text{lift})) = CLOSED
grd3: liftdoorstatus(lift) = CLOSED
grd4: liftposition(lift) ∈ liftschedule(lift)

then act1: floordoorstatus(lift) := floordoorstatus(lift) \ {liftposition(lift) \mapsto OPEN}
end

Event CloseFloorDoor ≡
extends CloseFloorDoor

any lift
when grd1: liftstatus(lift) = STOPPED
    grd2: floordoorstatus(lift)(liftposition(lift)) = OPEN
    grd3: liftdoorstatus(lift) = OPEN
    grd4: lift \notin waiting
    grd5: liftposition(lift) ∈ liftschedule(lift)

then act1: floordoorstatus(lift) := floordoorstatus(lift) \ {liftposition(lift) \mapsto CLOSED}
end

Event OpenLiftDoor ≡
extends OpenLiftDoor

any lift
when grd1: liftstatus(lift) = STOPPED
    grd2: floordoorstatus(lift)(liftposition(lift)) = OPEN
    grd3: liftdoorstatus(lift) = CLOSED
    grd4: liftposition(lift) ∈ liftschedule(lift)

then act1: liftdoorstatus(lift) := OPEN
    act2: waiting := waiting ∪ {lift}
end

Event CloseLiftdoor ≡
extends CloseLiftdoor

any lift
when grd1: liftstatus(lift) = STOPPED
    grd2: floordoorstatus(lift)(liftposition(lift)) = CLOSED
    grd3: liftdoorstatus(lift) = OPEN
    grd4: liftposition(lift) ∈ liftschedule(lift)

then act1: liftdoorstatus(lift) := CLOSED
    act2: liftbuttons(lift) := (liftbuttons(lift) \ {liftposition(lift) \mapsto OFF})
    act3: liftschedule(lift) := liftschedule(lift) \ {liftposition(lift)}
end
Event  Release \( \cong \)
extends  Release
  any  lift
  when  \text{grd1}:  \text{lift} \in \text{waiting}
  then  \text{act1}:  \text{waiting} := \text{waiting} \setminus \{\text{lift}\}
end

END