

# Elevator Control System (Version 0.1)

SE-381 Class

Mark J. Sebern

September 29, 2003

## 1 Introduction

The purpose of the elevator control system is to manage movement of an elevator in response to user requests.

### 1.1 Basic elements

The elevator system has the following basic elements and parameters.

#### 1.1.1 Number of elevators

Number of elevators in the system.

|  $numElevators : \mathbb{N}_1$

#### 1.1.2 Floors

Floors are the building levels serviced by the elevator system. Internally, floors are numbered starting at one, with floor “1” being the lowest floor in the elevator system. All elevators in the system are assumed to service the same range of floors.

*Floors* is modeled as a finite set, since we may need to apply the cardinality operator, which does not work with infinite sets.

|  $nFloors : \mathbb{N}$

|  $Floors : \mathbb{FZ}$

|  $\langle\langle \text{grule TopFloorGE2} \rangle\rangle$

|  $nFloors \geq 2$

|  $\langle\langle \text{rule FloorsDef} \rangle\rangle$

|  $Floors = 1 .. nFloors$

Proof note:

The *TopFloorGE2* label marks the  $nFloors \geq 2$  predicate as a theorem that the Z/EVES prover can assume to be true.

The *FloorsDef* label marks the  $Floors = 1 .. nFloors$  equality predicate so it can be used as a substitution rule. In other words, when the prover sees *Floors*, this rule means that it can rewrite that part of the expression to be  $1 .. nFloors$  instead.

These proof rules are needed for use in later proofs.

In some circumstances, floors are labeled with numbers or other designators (e.g., “B” or “G”) that do not correspond to the internal floor numbers. (Have you noticed that many hotels and other buildings don’t have a floor that is labeled “13”, for example?)

To model these externally visible floor labels, we introduce a new type, which is a string (sequence) of characters.

[*CHAR*]

*STRING* == seq *CHAR*

*FLOOR\_LABEL\_LENGTH\_MAX* == 4

$FLOOR\_LABEL : \mathbb{F} \textit{STRING}$
$\forall fl : FLOOR\_LABEL \bullet \#fl \geq 1 \wedge \#fl \leq FLOOR\_LABEL\_LENGTH\_MAX$

Proof note:

The following proof steps demonstrate how the domain check for the above axiomatic declaration can be proved.

**proof**  
*reduce*;  
 ■

Now, we would like to associate floor labels with internal floor numbers.

One way to do this would be to use the Z “abbreviation notation” to define a set of all possible floor-to-label mappings, and then define a global set of floor mappings for a specific elevator system.

*FLOOR\_LABEL\_MAPPING* == *Floors* × *FLOOR\_LABEL*

$FloorLabelMap1 : \mathbb{F} \textit{FLOOR\_LABEL\_MAPPING}$
--

This would define a set (*FloorLabelMap1*) of floor-to-label pairs, but we still have the problem of specifying that there must be exactly one pair for each floor, with the first member of the ordered pair equal to the floor number. Also, we want to make sure that one floor number can’t map to more than one label, and that one label cannot be associated with more than one floor.

Another way to define the floor-label association would be to use a Z relation, which is a shorthand way of defining a set of ordered pairs.

$FloorLabelMap2 : \textit{Floors} \leftrightarrow \textit{FLOOR\_LABEL}$
--

But, since this is only a shorthand for the abbreviation definition and set definition above, we still have the same problem.

What we really need is a function, as shown below. In defining the function, we have a number of alternatives:

---

	<i>FloorLabelMapPartialFunction</i> : Floors $\leftrightarrow$ FLOOR_LABEL
	<i>FloorLabelMapTotalFunction</i> : Floors $\rightarrow$ FLOOR_LABEL
	<i>FloorLabelMapPartialInjection</i> : Floors $\nrightarrow$ FLOOR_LABEL
	<i>FloorLabelMapTotalInjection</i> : Floors $\rightarrow$ FLOOR_LABEL
	<i>FloorLabelMapPartialSurjection</i> : Floors $\twoheadrightarrow$ FLOOR_LABEL
	<i>FloorLabelMapTotalSurjection</i> : Floors $\twoheadrightarrow$ FLOOR_LABEL
	<i>FloorLabelMapBijection</i> : Floors $\xrightarrow{\sim}$ FLOOR_LABEL
	<i>FloorLabelMapFinitePartialFunction</i> : Floors $\leftrightarrow$ FLOOR_LABEL
	<i>FloorLabelMapFinitePartialInjection</i> : Floors $\nrightarrow$ FLOOR_LABEL

---

We model the association between floor numbers and floor labels with a global function.

	<i>FloorLabels</i> : Floors $\rightarrow$ FLOOR_LABEL
--	---

---

Note that the function *FloorLabels* is defined as a total injection. Because *FloorLabels* is total, its domain is equal to the set of floors in the elevator system. Because *FloorLabels* is injective, its inverse is also a function. This allows us to map unambiguously from floor numbers to floor labels or from floor labels to floor numbers.

If we want to know the label for a specified floor number, we can write an expression that applies the function.

$$floorlabel = FloorLabels\ floornum$$

Similarly, we can apply the function's inverse to get the floor number for a specified label.

$$floornum = FloorLabels\sim floorlabel$$


---

### 1.1.3 Elevator direction

An elevator may be stopped, or it may be moving up or down.

$$Direction ::= DirUp \mid DirDown \mid DirHalt$$

Proof note:

The following theorem is defined to specify the enumerated directions, so the theorem prover can know that these are the only possible direction values.

**theorem** frule DirectionDef

$$\forall d : Direction \bullet d = DirUp \vee d = DirDown \vee d = DirHalt$$

The following proof steps demonstrate how the *DirectionDef* theorem can be proved.

**proof**

apply *Direction*\$member to predicate  $d \in Direction$ ;

reduce;

■

### 1.1.4 Valid elevator calls

An elevator call is a summons from a specific floor, which indicates that a user has signaled a desire to travel in a specified direction (up or down) from that floor.

The requested direction uses the same type as that used for an elevator’s direction of travel, but the “halt” direction is excluded.

$$CallDirection ::= \{DirUp, DirDown\}$$

The following theorem allows the Z/EVES prover to assume the correct type of *CallDirection*; it is needed for later proofs.

**theorem** grule CallDirectionType  
 $CallDirection \in \mathbb{F} Direction$

All that is necessary to prove this theorem is to expand the definition of *CallDirection*.

**proof**  
*invoke CallDirection;*  
*prove;*  
 ■

A call is represented by a pair that contains the originating floor and the desired direction of travel. The bottom floor has no “down” button and the top floor has no “up” button.

$$\begin{array}{|l} ValidCalls : \mathbb{F}(Floors \times CallDirection) \\ \hline \langle\langle \text{rule ValidCallsDef} \rangle\rangle \\ ValidCalls = (Floors \times CallDirection) \setminus \{(nFloors, DirUp), (1, DirDown)\} \end{array}$$

Proof note:

The following theorem specifies the type of the domain of *ValidCalls* so that the Z/EVES theorem prover can assume this fact.

**theorem** grule ValidCallsDomType  
 $\forall c : \mathbb{F} ValidCalls \bullet \text{dom } c \in \mathbb{F}(1 .. nFloors)$

### 1.1.5 Elevator status

An elevator may be in service or out of service.

$$ServiceStatus ::= InSvc \mid OutSvc$$

Proof note:

The following theorem is defined so that Z/EVES knows that the service status is binary; an elevator is either in service or out of service. This permits the theorem prover to infer, for example, that if an elevator is not in service it must be out of service. The theorem might seem obvious from the type definition, but Z/EVES doesn’t automatically know this fact about free types.

**theorem** frule ServiceStatusDef  
 $\forall s : ServiceStatus \bullet s = InSvc \vee s = OutSvc$

The following proof steps demonstrate how the *ServiceStatusDef* theorem can be proved.

**proof**

*apply ServiceStatus\$member to predicate  $s \in ServiceStatus$ ;*  
*reduce;*

■

## 1.2 Elevator calls

A schema is used to model the set of pending elevator calls, to make it easier to define operations.

<i>Calls</i> <i>calls : <math>\mathbb{F}</math> ValidCalls</i>
---

The following schema describes the initial state of the elevator calls.

<i>InitCalls</i> <i>Calls</i>
<i>calls = <math>\emptyset</math></i>

The following theorem asserts that the elevator calls can be successfully initialized.

**theorem** InitCallsOK

$\exists$  *Calls* • *InitCalls*

Proof note:

The following proof steps demonstrate how the *InitCallsOK* theorem can be proved. A single step of *prove by reduce* would also work.

**proof**

*reduce;*  
*invoke Calls;*  
*prove;*

■

## 1.3 Elevator

An elevator has a current location (floor) and direction of movement. It also has a set of floor requests that correspond to the floor buttons currently selected inside the elevator.

A finite set is used to model *requests*.

<i>Elevator</i> <i>curFloor : Floors</i> <i>status : ServiceStatus</i> <i>curDir : Direction</i> <i>requests : <math>\mathbb{F}</math> Floors</i>
---

The following schema describes the initial state of an elevator.

*InitElevator*

*Elevator*

$curFloor = 1$   
 $status = InSvc$   
 $curDir = DirHalt$   
 $requests = \emptyset$

The following theorem asserts that an elevator can be successfully initialized.

**theorem** *InitElevatorOK*

$\exists Elevator \bullet InitElevator$

Proof note:

The following proof steps demonstrate how the *InitElevatorOK* theorem can be proved. A faster alternative would be to use a single step of *prove by reduce*, which in effect combines all the steps into one operation.

**proof**

*reduce*;  
*invoke Elevator*;  
*apply FloorsDef to expression Floors*;  
*prove*;

■

## 2 Elevator system operations

A number of operations are specified for the elevator system. Some apply to a single elevator, with or without information on elevator calls, and others apply to the elevator system as a whole.

### 2.1 Operation status

Operations return a status code to indicate their success or failure. The following set of status codes represents the values defined so far.

$OpStatusCode ::= StatusOK \mid$   
 $StatusOutOfService \mid$   
 $StatusInvalidMovement$

### 2.2 Handling calls

Calls from waiting passengers are associated with the elevator system as a whole, rather than with a particular elevator.

The *HandleCalls* schema describes the operation that adjusts the system state to reflect new calls originated by waiting passengers.

*HandleCalls*

$\Delta Calls$

$newCalls? : \mathbb{F} ValidCalls$

$calls' = calls \cup newCalls?$

The following theorem asserts that the *HandleCalls* operation covers all possible cases of system state and input.

**theorem** HandleCallsIsTotal

$\forall Calls; newCalls? : \mathbb{F} \text{ ValidCalls} \bullet \text{pre } HandleCalls$

Proof note:

The theorem is proved by simple reduction.

**proof**

*reduce;*

■

### 2.3 Elevator movement

In this model, elevator movement is broken down into the following components:

- Movement up or down by one floor.
- When at a floor, performing a “visit” (opening doors, exchanging passengers, closing doors) or deciding not to do so, and accepting new floor requests from passengers.
- Choosing (calculating) an updated direction of movement, taking into account the pending requests and calls.

To be continued ...