Specification of an Elevator in Z

Safety-Critical Systems 3, WiSe'07/08

Jan Peleska (jp@tzi.de)
Christof Efkemann (chref@tzi.de)

5th version, Nov 06, 2007

Safety Requirements

Declaration of Constants

The number of the ground floor and of the highest floor:

$$
\begin{array}{|l}
topFloor : \mathbb{Z} \\
groundFloor : \mathbb{Z} \\
\hline
groundFloor < topFloor
\end{array}
$$

Declaration of Types

The set of all admissible floor numbers:

$$FLOORS == groundFloor \mathinner{\ldotp\ldotp} topFloor$$

An extra value outside of $FLOORS$:

$$
\begin{array}{|l}
noFloor : \mathbb{Z} \\
\hline
noFloor \notin FLOORS
\end{array}
$$

The set of floor request values:

$$FLOORREQS == FLOORS \cup \{noFloor\}$$

Possible states of a door:

$$DOOR ::= open \mid closed$$

Possible motion states for the elevator:

$$DIRECTION ::= up \mid down \mid stopped$$

Specification of the State Space

The elevator state space, as far as it is safety-relevant:

$$
\begin{array}{l}
\textit{ElevatorState} \\
\hline
motorState : DIRECTION \\
door : DOOR \\
thisFloor : FLOORS \\
\hline
thisFloor = topFloor \Rightarrow motorState \in \{stopped, down\} \\
thisFloor = groundFloor \Rightarrow motorState \in \{stopped, up\} \\
door \neq closed \Rightarrow motorState = stopped \\
\hline
\end{array}
$$

Specification of the Operations

The polled input *sensor*? returns the number of the current floor.

The following partial operations check for the sanity of the *sensor*? input. Currently, the specification assumes that the sensor always works. If we want to make the *Move* operation total, we need to specify more partial operations that handle the case of a detected sensor failure. Obviously, in that case the elevator should stop as soon as possible, and a technician should be called.

$$
\begin{array}{l}
\textit{MoveUp} \\
\hline
\Delta \textit{ElevatorState} \\
\textit{sensor}? : \textit{FLOORS} \\
\hline
\textit{motorState} = \textit{up} \\
\textit{sensor}? \in \{\textit{thisFloor}, \textit{thisFloor} + 1\} \\
\textit{thisFloor}' = \textit{sensor}? \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{MoveDown} \\
\hline
\Delta \textit{ElevatorState} \\
\textit{sensor}? : \textit{FLOORS} \\
\hline
\textit{motorState} = \textit{down} \\
\textit{sensor}? \in \{\textit{thisFloor} - 1, \textit{thisFloor}\} \\
\textit{thisFloor}' = \textit{sensor}? \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{StableState} \\
\hline
\Delta \textit{ElevatorState} \\
\textit{sensor}? : \textit{FLOORS} \\
\hline
\textit{motorState} = \textit{stopped} \\
\textit{sensor}? = \textit{thisFloor} \\
\textit{thisFloor}' = \textit{sensor}? \\
\hline
\end{array}
$$

$$\textit{Move} \mathrel{\widehat=} \textit{MoveUp} \lor \textit{MoveDown} \lor \textit{StableState}$$

User Requirements

Specification of the State Space

$UserState$

$$
\begin{array}{l}
ElevatorState \\
upQ : \mathrm{seq}\, FLOORS \\
downQ : \mathrm{seq}\, FLOORS \\
move : DIRECTION \\
\hline
\forall\, i : 1 \mathrel{..} (\#upQ - 1) \bullet upQ(i) < upQ(i+1) \\
\forall\, i : 1 \mathrel{..} (\#downQ - 1) \bullet downQ(i) > downQ(i+1) \\
\#upQ \geq 1 \Rightarrow thisFloor < head(upQ) \\
\#downQ \geq 1 \Rightarrow thisFloor > head(downQ)
\end{array}
$$

Specification of the Operations

The input *targetFloor*? contains the floor desired, if a button is pressed. It contains the value *noFloor*, if no button is pressed. As discussed in the lecture, after one round of processing, the interface resets the value to *noFloor*, even if the button is pressed for a longer time.

$RequestBase$

$$
\begin{array}{l}
\Delta\, UserState \\
\Xi ElevatorState \\
targetFloor? : FLOORREQS \\
\hline
targetFloor? \in FLOORS \\
move' = move
\end{array}
$$

$RequestUp$

$$
\begin{array}{l}
RequestBase \\
\hline
targetFloor? > thisFloor \\
\mathrm{ran}\ upQ' = \mathrm{ran}\ upQ \cup \{targetFloor?\} \\
downQ' = downQ
\end{array}
$$

RequestDown

$$
\begin{array}{l}
\mathit{RequestBase} \\
\hline
\mathit{targetFloor}? < \mathit{thisFloor} \\
\mathrm{ran}\ \mathit{downQ}' = \mathrm{ran}\ \mathit{downQ} \cup \{\mathit{targetFloor}?\} \\
\mathit{upQ}' = \mathit{upQ}
\end{array}
$$

RequestThis

$$
\begin{array}{l}
\mathit{RequestBase} \\
\hline
\mathit{targetFloor}? = \mathit{thisFloor} \\
\mathit{downQ}' = \mathit{downQ} \\
\mathit{upQ}' = \mathit{upQ}
\end{array}
$$

$$\mathit{Request} \mathrel{\widehat=} \mathit{RequestUp} \lor \mathit{RequestDown} \lor \mathit{RequestThis}$$

The operation *StartMove* actually starts the elevator to move, if there is a request pending.

StartMoveBase

$$
\begin{array}{l}
\Delta \mathit{UserState} \\
\hline
\mathit{upQ}' = \mathit{upQ} \\
\mathit{downQ}' = \mathit{downQ} \\
\mathit{thisFloor}' = \mathit{thisFloor} \\
\mathit{motorState}' = \mathit{move}'
\end{array}
$$

StartMoveUp

$$
\begin{array}{l}
\mathit{StartMoveBase} \\
\hline
\mathit{move} = \mathit{stopped} \\
\#\mathit{upQ} > 0 \\
\mathit{move}' = \mathit{up}
\end{array}
$$

StartMoveUpGoOn

$$
\begin{array}{l}
\mathit{StartMoveBase} \\
\hline
\mathit{move} = \mathit{up} \\
\mathit{motorState} = \mathit{stopped} \\
\mathit{move}' = \mathit{move}
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{StartMoveDown}} \\
\hline
\textit{StartMoveBase} \\
\hline
\textit{move} = \textit{stopped} \\
\#\textit{downQ} > 0 \\
\textit{move}' = \textit{down} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{StartMoveDownGoOn}} \\
\hline
\textit{StartMoveBase} \\
\hline
\textit{move} = \textit{down} \\
\textit{motorState} = \textit{stopped} \\
\textit{move}' = \textit{move} \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{StartMove} \mathrel{\widehat=} \textit{StartMoveUp} \lor \textit{StartMoveUpGoOn} \lor \\
\textit{StartMoveDown} \lor \textit{StartMoveDownGoOn}
\end{array}
$$

The operation *UserMove* decides what happens if the elevator reaches another floor. Here, we assume that the safety-relevant parts are already handled by the operation *Move*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{UserMoveUpBase}} \\
\hline
\Delta \textit{UserState} \\
\textit{sensor?} : \textit{FLOORS} \\
\hline
\textit{sensor?} = \textit{thisFloor} + 1 \\
\textit{downQ}' = \textit{downQ} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{UserMoveUpSkip}} \\
\hline
\textit{UserMoveUpBase} \\
\hline
\textit{sensor?} \neq \textit{upQ}(1) \\
\textit{upQ}' = \textit{upQ} \\
\textit{motorState}' = \textit{motorState} \\
\textit{move}' = \textit{move} \\
\hline
\end{array}
$$

$UserMoveUpArrive$

$UserMoveUpBase$

$sensor? = upQ(1)$
$upQ' = tail\ upQ$
$motorState' = stopped$

$UserMoveUpArriveGoOn$

$UserMoveUpArrive$

$\#upQ > 1$
$move' = move$

$UserMoveUpArriveStop$

$UserMoveUpArrive$

$\#upQ = 1$
$\#downQ = 0$
$move' = stopped$

$UserMoveUpArriveTurn$

$UserMoveUpArrive$

$\#upQ = 1$
$\#downQ > 0$
$move' = down$

$UserMoveUp \mathrel{\widehat{=}} UserMoveUpSkip \lor UserMoveUpArriveGoOn \lor UserMoveUpArriveStop \lor UserMoveUpArriveTurn$

$UserMoveDownBase$

$\Delta UserState$
$sensor? : FLOORS$

$sensor? = thisFloor - 1$
$upQ' = upQ$

$$
\begin{array}{|l}
\hline
\textit{UserMoveDownSkip} \\
\textit{UserMoveDownBase} \\
\hline
sensor? \neq downQ(1) \\
downQ' = downQ \\
motorState' = motorState \\
move' = move \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{UserMoveDownArrive} \\
\textit{UserMoveDownBase} \\
\hline
sensor? = downQ(1) \\
downQ' = tail\ downQ \\
motorState' = stopped \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{UserMoveDownArriveGoOn} \\
\textit{UserMoveDownArrive} \\
\hline
\# downQ > 1 \\
move' = move \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{UserMoveDownArriveStop} \\
\textit{UserMoveDownArrive} \\
\hline
\# downQ = 1 \\
\# upQ = 0 \\
move' = stopped \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{UserMoveDownArriveTurn} \\
\textit{UserMoveDownArrive} \\
\hline
\# downQ = 1 \\
\# upQ > 0 \\
move' = up \\
\hline
\end{array}
$$

$$
\begin{array}{l}
UserMoveDown \mathrel{\widehat=} UserMoveDownSkip \lor UserMoveDownArriveGoOn \lor \\
UserMoveDownArriveStop \lor UserMoveDownArriveTurn
\end{array}
$$

$$
\begin{array}{l}
\textit{UserStableState} \\
\hline
\Delta \textit{UserState} \\
\textit{sensor}? : \textit{FLOORS} \\
\hline
\textit{sensor}? = \textit{thisFloor} \\
\textit{upQ}' = \textit{upQ} \\
\textit{downQ}' = \textit{downQ} \\
\textit{motorState}' = \textit{motorState} \\
\textit{move}' = \textit{move} \\
\hline
\end{array}
$$

$$\textit{UserMove} \mathrel{\widehat{=}} \textit{UserMoveUp} \lor \textit{UserMoveDown} \lor \textit{UserStableState}$$

And finally the composition of all the safety and the user requirements operations of the elevator:

$$\textit{ElevatorOp} \mathrel{\widehat{=}} \textit{Move} \land (\textit{Request} \lor \textit{StartMove} \lor \textit{UserMove})$$