Spécification d'un ascenseur

Up and Down The Temporal Way

H. BARRINGER*
Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL

A formal specification of a multiple-lift system is constructed. The example illustrates and justifies one of many possible system specification styles based on temporal techniques.

Received September 1985

1. INTRODUCTION

Over the last decade there has been widespread research directed at obtaining techniques for the analysis, specification and development of concurrent systems. Several of these lines of research have led to the belief that temporal logic is a useful tool for reasoning about such systems.[1–4] The use of temporal logic enables, in particular, analysis of both safety and liveness properties in a single uniform logical framework (see Ref. 5 for extensive examples). More recently, techniques have been developed for achieving compositional temporal proof systems.[6,7] Compositionality is an essential requirement for the hierarchical development of implementations from formal specifications. Without compositionality, the check on consistency of a development step would be delayed until all interactions between the developed components are known, essentially, at the implementation level; clearly, it could be rather costly if such a consistency check then showed that the system did not achieve the overall specification. In general, compositionality can be achieved by realising that a specification of any component must include assumptions about the behaviour of the environment in which the component will reside. In the temporal framework, this requires that one can distinguish actions made by a component from those made by its environment; in Refs 6 and 7 the coarse technique of labelling actions is used for just that purpose. Although it is sometimes possible

5 presents the multiple-lift system. Final comments are made in Section 6. For convenience, an appendix contains the semantics of the logic used in this article.

2. INFORMAL REQUIREMENTS

The following is a description of the lift-control system problem as set by Neil Davis.

A Lift-Control System

An n-lift system is to be installed in a building with m floors. The lifts and the control mechanism are supplied by a manufacturer. The internal mechanisms of these are assumed (given) in this problem.

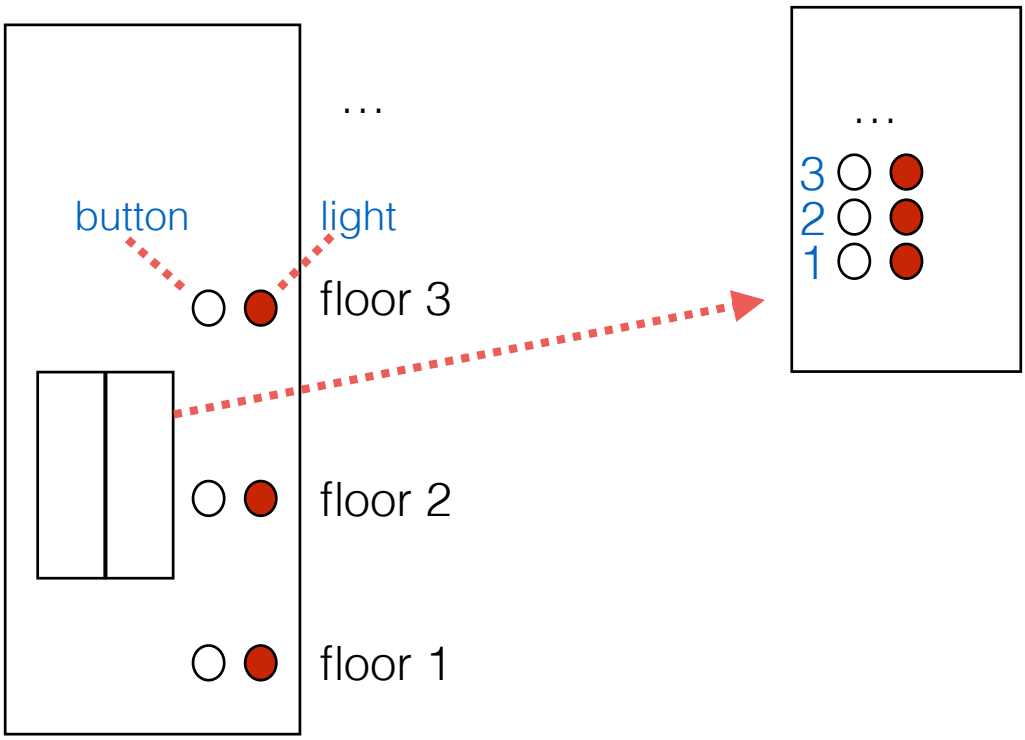
Design the logic to move lifts between floors in the building according to the following rules.

(1) Each lift has a set of buttons, one button for each floor. These illuminate when pressed and cause the lift to visit the corresponding floor. The illumination is cancelled when the corresponding floor is visited (i.e. stopped at) by the lift.

(2) Each floor has two buttons (except ground and top), one to request an up-lift and one to request a down-lift. These buttons illuminate when pressed. The buttons are cancelled when a lift visits the floor and is either travelling in the desired direction, or visiting the floor with no requests outstanding.

In the latter case, if both floor-request buttons are illuminated, only one should be cancelled. The algorithm used to decide which to service should minimise the waiting time for both requests.

Specification of a lift



H. Barringer ("Up and down, the temporal way", 1985)

Specification of a lift

Hypothesis:

- A floor door is open or closed.
- A button is pressed or depressed.
- An indicator light is on or off.
- The cabin is present at floor i, or it is absent.

Specification of a lift

P1. Safe doors:
A floor door is never opened if the cabin is not present at the given floor.

P2. Indicator lights:
The indicator lights correctly reflect the current requests.

P3. Services:
All requests are eventually satisfied.

P4. Smart service:
The cabin only services the requested floors and does not move when there is no request.

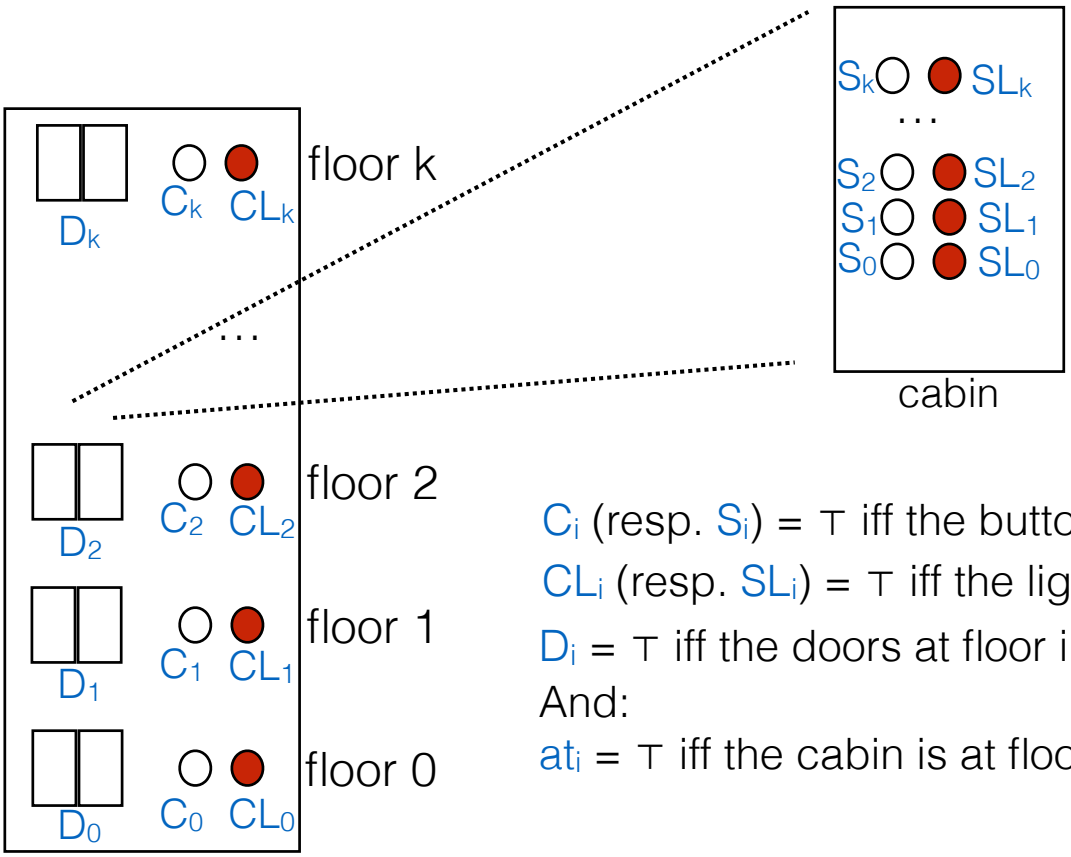
Specification of a lift

P5. Diligent service:
The cabin does not pass by a floor for which it has a request without servicing it.

P6. Direct movements:
The cabin always moves directly from previous to next serviced floor.

P7. Priorities:
The cabin services in priority requests that do not imply a change of direction (upward or downward).

Specification of a lift: the atomic prop.



C_i (resp. S_i) = $\top$ iff the button is pressed

CL_i (resp. SL_i) = $\top$ iff the light is on

D_i = $\top$ iff the doors at floor i are opened

And:

at_i = $\top$ iff the cabin is at floor i

Specification of a lift

P1. Safe doors:

A floor door is never opened if the cabin is not present at the given floor.

$$\mathbf{G}\,(D_i \Rightarrow at_i)$$

Porte i ouverte

cabine à l'étage i

Specification of a lift

P2. Indicator lights:
The indicator lights correctly reflect the current requests.

$\bigwedge_i \mathbf{G}\,(\,C_i \Rightarrow (LC_i \vee \text{servicing}_i\,))$: turned on when necessary

with: $\text{servicing}_i = at_i \wedge D_i$

$\bigwedge_i \mathbf{G}\,(\,LC_i \Rightarrow (\,LC_i\ \mathbf{W}\ \text{servicing}_i\,)\,)$: stay lit when necessary

$\bigwedge_i \mathbf{G}\,(\,\text{servicing}_i \Rightarrow (\neg\, LC_i \wedge \neg\, SLi))$: turned off when necessary

$\bigwedge_i \mathbf{G}\,(\,(\neg\, LC_i\,) \Rightarrow (\,(\neg LC_i)\ \mathbf{W}\ C_i)\,)$:only turned on when necessary

or $\bigwedge_i \mathbf{G}\,(\,LC_i \Rightarrow (\,LC_i\ \mathbf{S}\ C_i)\,)$

(and the same for S_i and SL_i)

Specification of a lift

P2. Indicator lights:
The indicator lights correctly reflect the current requests.

An *alternative* is:

$\bigwedge_i \mathbf{G}\,(\,LC_i \iff (\,(\neg\text{servicing}_i\,)\ \mathbf{S}\ (C_i \wedge \neg\ \text{servicing}_i))\,)$

(and the same for S_i and SL_i)

Specification of a lift

P3. Services:
All requests are eventually satisfied.

$$\bigwedge_i \mathbf{G}\,(\,\text{request}_i \Rightarrow \mathbf{F}\,\text{servicing}_i\,))$$

with: $\text{request}_i = C_i \vee S_i$

P4. Smart service:
The cabin only services the requested floors and does not move when there is no request.

$$\bigwedge_i \mathbf{G}\,(\;\text{servicing}_i \Rightarrow [\,\text{servicing}_i\;\mathbf{S}\;(CL_i \vee SL_i)\,]\,)$$

$$\bigwedge_i \mathbf{G}\,(\;at_i \Rightarrow (\,at_i\;\mathbf{W}\,(\;\bigvee_{j\neq i}(CL_j \vee SL_j)\;)\;)\,)$$

Specification of a lift

P5. Diligent service:
The cabin does not pass by a floor for which it has a request without servicing it.

$$\bigwedge_i \mathbf{G}\,([(\,LC_i \vee LS_i) \wedge at_i\,] \Rightarrow (at_i\;\mathbf{U}\;\text{servicing}_i\,))$$

Specification of a lift

P6. Direct movements:
The cabin always moves directly from previous to next serviced floor.

$$\bigwedge_{i<j} \mathbf{G}\,(\,From\_i\_to\_j \Rightarrow (at_i \vee betw\_floors)\ \mathbf{U}\ (at_{i+1} \wedge (at_{i+1} \vee betw\_floors)\ \mathbf{U}\ (at_{i+2} \ldots (at_{j-1} \wedge (at_{j-1} \vee betw\_floors)\ \mathbf{U}\ at_j))))$$

$$\neg at_0 \wedge \ldots \wedge \neg at_n$$

$$servicing_i \wedge [\,(servicing_i \vee \neg\, service)\ \mathbf{U}\ servicing_j\,]$$

$$service = servicing_0 \vee servicing_1 \vee \ldots \vee servicing_k$$

Specification of a lift

P7. Priorities:
The cabin services in priority requests that do not imply a change of direction (upward or downward).

$$Up = \bigvee_{i=1..k} [\,(at_i \vee betw\_floors\,)\ \mathbf{S}\ at_{i-1} \ \wedge\ (at_i \vee betw\_floors)\ \mathbf{U}\ at_i\,]$$

$$Down = \bigvee_{i=0..k-1} [\,(at_i \vee betw\_floors\,)\ \mathbf{S}\ at_{i+1} \ \wedge\ (at_i \vee betw\_floors)\ \mathbf{U}\ at_i\,]$$

$$\mathbf{G} \bigwedge_{i=0..k-1} [(servicing_i \wedge Down \wedge \bigvee_{j<i} (CL_j \vee SL_j)\,) \Rightarrow \bigvee_{n<i} From\_i\_to\_n]$$

$$\mathbf{G} \bigwedge_{i=1..k} [(servicing_i \wedge Up \wedge \bigvee_{j>i} (CL_j \vee SL_j)\,) \Rightarrow \bigvee_{n>i} From\_i\_to\_n]$$

Exercice

* Un feu de circulation:
  - le feu n'est jamais rouge et vert en même temps
  - l'ordre des couleurs est rouge vert orange rouge etc.
  - le feu est vert infiniment souvent

* Il existe au moins un état vérifiant *a* le long de l'exécution.

* Il existe un unique état vérifiant *a* le long de l'exécution.

* Il existe deux états (et seulement deux) vérifiant *a* le long de l'exécution.

* tout état vérifiant *a* est précédé par un état vérifiant *b*.

* si *a* est vérifié trois fois, alors on passera par un état vérifiant *b*.