David Pick
Eric Stokes
Pete Brousalis

Elevator Project, Part 2

Introduction

For the second part of the elevator modeling project, we were required to add additional behavior to our previous model, in particular the control system. In order to start modeling our system, we used a model provided by Dr. Curt Clifton. His model provided us with a working elevator system that would allow the elevator to go up and down to floors, open and close its doors and display a direction indicator. We had to add two additional parameters to our Lift model that would store the outside up and down requests. In addition, we added a control system constrained by predicates. Below is a detailing of our model.

Helper Functions

We have written a few helper functions in order to clean up the code and provide easier means of constraining the system by being able to be called from with predicates.

Good

- The upRequestsAbove function gives us any requests within outUpLiftRequests (Requests pressed from the outside the lift) for the current floor and above
- The downRequestsAbove function gives us any requests within outDownLiftRequests (Requests pressed from the outside the lift) for the current floor and below
- The requestsAbove function gives us ALL of the requested floors above the elevator's current position (including the current floor)
- The requests function gives us ALL of the requested floors above the elevator's current position (including the current floor)

Predicates

Our model contains several predicates in charge of the control algorithm. The predicates play as facts in order to constrain our system. Some predicates are called within others. There is also a set of predicates for viewing sample instances which allow us to test specific instances.

Constraint Predicates

- The initTime predicate initializes our time and ensures the elevator starts correctly.
- The someChange predicate states that there must be a change of some sort unless that elevator is out of service
- The noFloorChangeExcept predicate makes sure that a floor change only happens if there is a requested floor
- The noRequestChangeExcept predicate makes sure that a request change happens within the current lift only

- There are several more of the /no[A-Za-z]+Except/ requests that behave in the same manor.

regexs in the house?

er, manner?

Operation Predicates

- The move predicate constrains when our elevator is allowed to move, and calls other predicates depending on conditions that will allow us to set our lights.
- The changeService predicate will change whether or not the elevator is in service or out of service
- The requestFloor predicate will ensure that another floor is requested, and that that floor is not already in the list of requested floors for that type of request
- The /out(Up | Down)RequestFloor/ requests a floor in the same manor, but adds it to the appropriate request. These apply to buttons pressed outside of floors.
- The openDoors predicate opens the doors if and only if we are on a serviced floor. We can impose such a constraint, because even if a drunken woman/man wakes up in an elevator, they must request A floor, even if it is the same floor to get the doors to open.
- The closeDoors predicate can be much looser as most of the time we do want the doors to be closed

Assertions

There are a few assertions that check the correctness of the model.

- The OnlyValid assertion ensures that all transitions are valid, which also means our elevator behaves in the manor that it should.
- The OnlyValid_Move assertion checks that all of the moves are valid, ie: no teleportation.
- The OnlyValid_Open assertion checks that every time a door opens, it is servicing a floor.

Would have been best to add an assertion about requests eventually being serviced. That would have detected the missing planned direction changes (see comments in model).

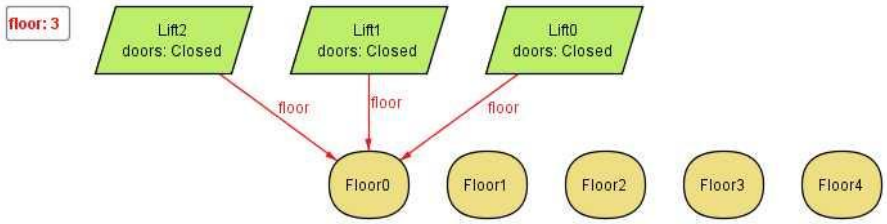
Visual Representations

We used the Alloy to preview our model and export the images for analysis.

Captures of our assertions need not be shown as no counterexamples are found.

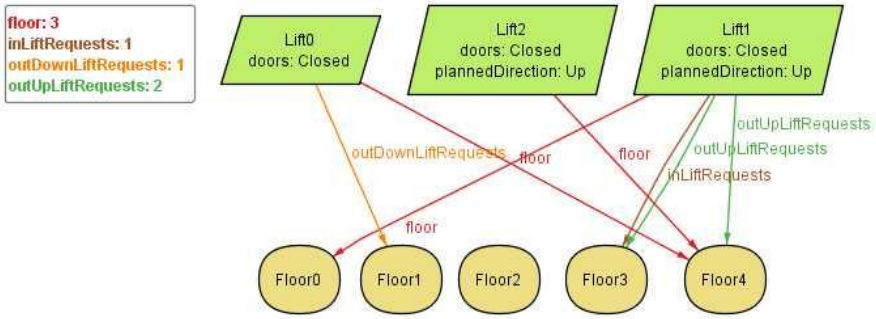
Show Predicate

Figure 1.



This diagram shows that all elevators initialize at floor0 and they all have closed doors

Figure 2.



Some paired images showing transitions would have been helpful.

This diagram shows all elevators at a later state in the show predicate

Time Spent

David Pick	18 hours
Pete Brousalis	12 hours
Eric Stokes	14 hours

Project description requested that code for model be included in the pdf. (See CONSIDER comments in model source in SVN.)

P2 Milestone

Grade: 88

Criteria (weight)	5 Exemplary	3 Satisfactory	1 Needs Improvement	Weighted Score
Organization of Report (x1)	Report is well organized with appropriate sections.	Report is mostly well organized with some sections.	Report is not well organized. There is no obvious logic to the order.	
Clarity and Conciseness of Prose (x2)	Written description is clear and unambiguous. It is not unnecessarily wordy.	Written description is mostly clear and unambiguous. Occasional instances of unclear or awkward writing.	Written description is unclear and ambiguous, or else missing altogether.	
Use of Images (x2)	Images are used appropriately to illustrate the model. Use of images is not excessive. All images are referenced in the text, described clearly, and have appropriate captions.	Use of images is generally appropriate. There may be an excessive number of images, the text may not describe the images, or images may lack captions.	Images are absent or insufficient.	
Professionalism (x1)	Report presents a professional tone. It could be shared with a "real-world" customer without changes.	Report largely presents a professional tone. It could be shared with a "real-world" customer with minor revisions.	Report is unprofessional. The majority of the prose would have to be rewritten before sharing the report with a "real-world" customer.	
Organization of Model (x2)	Model is well organized with appropriate sections.	Model is mostly well organized with some sections.	Model is not well organized. There is no obvious logic to the order.	
Depth of Analysis (x4)	Model covers all important corner cases. It demonstrates a deep understanding of the problem.	Model covers many important corner cases. Some cases might be treated in an unusual manner, but such treatment is documented.	Model treats few or no corner cases. It demonstrates just a superficial understanding of the problem.	
Logical Correctness (x4)	Model is free of logical and semantic errors. Formal model matches prose description.	Model is mostly free of logical and semantic errors. Such errors are limited to a very few different sorts.	Model has many logical errors of a variety of kinds.	
Clarity of Formalism (x4)	Model uses well-named signatures, paragraphs, and variables. Predicates and functions are introduced to improve readability.	Model mostly uses well-named signatures, paragraphs, and variables. Predicates and functions may not be introduced in all appropriate cases.	Signatures, paragraphs, and variables are poorly named; names are too short, ambiguous, or nonsensical.	
			Total Score:	88

-2, model code missing in pdf.