Temporal Logic Case Study

William G. Wood
Carnegie Mellon University
Software Engineering Institute
Pittsburgh Pa 15213[1]

Abstract. This report is a case study applying temporal logic to specify the operation of a bank of identical elevators servicing a number of floors in a building. The goal of the study was to understand the application of temporal logic in a problem domain that is appropriate for the method, and to determine some of the strengths and weaknesses of temporal logic in this domain. The case study uses a finite state machine language to build a model of the system specification, and verifies that the temporal logic specifications are consistent using this model. The specification aspires to be complete, consistent, and unambiguous.

Introduction

In this paper temporal logic is used to specify the operation of a bank of elevators servicing a number of floors. The elevator is described only in terms of the actions the end-user takes, and the indicators that the system displays to him. Hence the users push summons buttons on each floor to get an elevator to move to the floor, and when it arrives and opens its doors, they step inside and push destination buttons to get it to move to their desired destination floor. When they depress the push buttons, they light up, and become unlit in an appropriate manner when the elevator travels between floors. The specification is produced at two levels. The first level is that of the end-users describing how they wish the elevator system to operate, and the second level is that of the system engineer adding behavior to provide equitability and efficiency in the performance.

The specification is defined from the point of view of an applications engineer, rather than that of an expert in temporal logic. Since the author was interested in the problem of specifying a complex problem, the elevator contains most of the signals and indicators seen in modern elevators. In addition to the temporal logic specification, an automated model of the specification was also built, using the State Machine Language (SML) developed by Ed Clarke and others at CMU, and described in [Clarke 87]. The SML model was then used to verify the consistency of the formulas, using a toolset called EMC, also described in the previous reference.

The complete elevator specification is given in [Wood 89], including the complete set of formulas, and the SML programs used to model the operation of a small elevator system. An abbreviated description of some portions of the system is given here, along with a discussion of the advantages of using this approach to specification.

To describe the elevator system from the user's viewpoint, the user's expectations on equality of service must be included in some way. This is an optimization problem, and depends on many factors, such as statistics of usage and current requests for service. The control of the elevators should be defined independently of the final scheduling strategies (which may vary with the time of day or the level of service, and may change over relatively short time periods, as the tenants of the building, or their elevator usage habits change). The approach taken in this paper is to treat all of the elevators as autonomous, with a few interfaces to a scheduler. The details of the scheduler are undefined except for its interfaces, and the constraints on the

[1]This work is sponsored by the Department of Defence.

fairness of that behavior. Hence each elevator will continue to provide its simple-minded service unless the scheduler issues a command to override this service to improve the efficiency of operation. One additional advantage to such an approach is that it is appropriate for distributed fault-tolerant operation. If the scheduler is not in service, or a processor loses communications with the scheduler, service will continue to be provided in a predictable, though sub-optimal, manner.

Barringer [Barringer 87] describes a multiple elevator system (using temporal logic) that is somewhat simpler than the one used here. Woodcock and others [Woodcock 87] use the theory of Communicating Sequential Processes (CSP) from [Hoare 85] to describe the operation of a single elevator system at a single floor. Schwartz and others [Schwartz 87] also use CSP to specify the elevator system for multiple lifts at many floors. In two of these cases (Barringer and Schwartz), where the elevator problem includes on-board signals and considers elevator motion, the authors considered primitive scheduling and coordination necessary to the definition of the problem.

Terminology

Temporal logic is a well-developed branch of modal logic, is thoroughly described in [Rescher 71], and has been proposed as applying both to the specification and verification of program behavior, and to the specification of system behavior. The first significant proposals for using temporal logic to describe program behavior came from Pnueli, and his later paper [Pnueli 85] is a good survey of the field. Another good general descriptions of the applicability of temporal logic is [Lamport 83]. There have been numerous papers from Clarke and others demonstrating this applicability; [Clarke 86] and [Clarke 87] describe the development of tools to automate the process. A more complete description of the specific temporal logic model used in this report is given in the appendix.

Temporal Operators

The expected logical operators used throughout are **and** ($\wedge$), **or** ($\vee$), **not** ($\neg$), **equivalence** ($\equiv$), and **implies** ($\rightarrow$).

The temporal operators to be used are listed below, and the bold word in the description of the operator is the word to be visualized when reading the formula.

1. $\Diamond$ **a** : This means that **eventually a** will be true.
2. []**a** : This means that **henceforth a** is true.
3. °**a** : This means that at the **next** state (instant in time) **a** will be true.
4. •**a** : This means that at the **previous** state (instant in time) **a** was true.
5. **a** $\Rightarrow$ **b** : This is read as **strictly implies**, and it means that henceforth, if **a** is true, then **b** is true. **(a $\Rightarrow$ b) $=_{df}$ [](a $\rightarrow$ b)**.
6. **a U b** : This means that **a** is true **until** the state (instant in time) w hen **b** occurs. The strong **until** implies that **b** will eventually occur. In addition, **a** must always be true in the present.
7. **a = b**: This means that **a** is strictly equivalent to **b**. **(a = b) $=_{df}$ [] (a $\equiv$ b)**.

Sets and Sequences

The sets in the system are: elevators **I**; floors **M**; directions **J = {up, down}**;
destination pushbuttons **IM $\equiv$ I*M**; on-floor summons buttons **MJ $\equiv$ M*J - (1, down) - (|M|, up)**;
and on-floor arrival lights **IMJ $\equiv$ I*M*J - ($\forall$ i $\in$ I, (i,1,down))- ($\forall$ i $\in$ I, (i,|M|,up))**.

Variable Definitions

The indicators and signals visible to the end user are each described below, and the variable definitions for each are given.

1. On each floor there is a summons button in each direction, with the obvious exception of the first and last floors. They are depressed to indicate that the traveler wishes an elevator to come to

floor **m**, and take him in the chosen direction **j**, $\mathbf{fl\_summ\_depr}_{m,j}$. Each summons push button is backlit to indicate that it has been depressed, $\mathbf{fl\_summ\_lit}_{m,j}$

2. On each floor, there are two indicators next to each elevator. These indicate that an elevator is stopping or is stopped at the floor. These are defined for all floors; though the indicators for the top floor going up, and the bottom floor going down are not ever set or reset, they are defined for consistency of terminology, $\mathbf{fl\_direction}_{l,m,j}$
3. On each floor **m** there are doors for each elevator **l**, $\mathbf{fl\_doors\_closed}_{l,m}$
4. For each elevator **l** there is a light for each floor **m**, indicating that the elevator is in the vicinity of that floor, $\mathbf{el\_position}_{l,m}$.
5. For each elevator there is a light signaling that the elevator is traveling up, and another that it is traveling down, $\mathbf{el\_direction}_{l,j}$ where **j= {up, down}**
6. Each elevator **l** also contains a set of destination pushbuttons (one for each floor **m**), each of which backlights when depressed. This indicates what floor the user wishes to go to. These are denoted by $\mathbf{el\_dest\_depr}_{l,m}$ and $\mathbf{el\_dest\_lit}_{l,m}$.

Safety Conditions

The safety conditions describe conditions of the operation which are disallowed. Most of the safety conditions in this context are necessary to describe conditions that are mutually inconsistent, such as the fact that the elevator cannot be going up and down at the same time. There are a few, however, forbidding absurd operations such as lights coming on of their own volition. Two of the many safety conditions are shown below

1. For each elevator only one arrival light at one floor can be on at any time. Having no arrival light on is always valid. The light can only be on when the elevator is in the vicinity of the floor.

$$\forall\ (\mathbf{l,m1,j1}) \in \mathbf{IMJ},\ \mathbf{fl\_direction}_{l,m1,j1} \Rightarrow \mathbf{el\_position}_{l,m1} \wedge (\forall\ (\mathbf{m,j}) \in (\ \mathbf{MJ}\ \mathbf{-(m1,j1)}),\ \neg\ \mathbf{fl\_direction}_{l,m,j})$$

2. The floor summons lights must not be lit, unless at the previous state they were lit or the push button was depressed.

$$\forall\ (\mathbf{m,j}) \in \mathbf{MJ},\ \mathbf{fl\_summ\_lit}_{m,j} \Rightarrow \bullet\ (\mathbf{fl\_summ\_lit}_{m,j} \vee \mathbf{fl\_summ\_depr}_{m,j})$$

User's Viewpoint

Some of the formula describing the operation of the elevator are shown below.

1. When a summons push button is depressed at floor **m** for direction **j**, it is also backlit, and will remain backlit until a floor directional light shows that any elevator **l** is stopping at floor **m** in the direction **j**. The button is always lit while depressed, indicating to the user that the system is responding to the request for service even if there is an elevator already at the floor with its arrival light set in the requested direction.

$$\forall\ (\mathbf{m,j}) \in \mathbf{MJ},\ \mathbf{fl\_summ\_depr}_{m,j} \Rightarrow \mathbf{fl\_summ\_lit}_{m,j}\ \mathbf{U}\ (\ \exists\ \mathbf{l} \in \mathbf{I},\ \mathbf{fl\_direction}_{l,m,j})$$

$$\forall\ (\mathbf{m,j}) \in \mathbf{MJ},\ (\neg\ \mathbf{fl\_summ\_depr}_{m,j} \wedge (\ \exists\ \mathbf{l} \in \mathbf{I},\ \mathbf{fl\_direction}_{l\ ,m,j}) \Rightarrow \neg\ \mathbf{fl\_summ\_lit}_{m,j}$$

2. When the floor directional light is on, the elevator is at the floor, and eventually its doors will open. The floor directional light comes on shortly before the doors open, to give the user time to move to the appropriate elevator. This formula explicitly states that after turning a floor directional light on, the elevator cannot change floors and the floor directional light stays on until the doors have opened.

$$\forall\ (\mathbf{l,m,j}) \in \mathbf{IMJ},\ \mathbf{fl\_direction}_{l,m,j} \Rightarrow (\ \mathbf{fl\_direction}_{l,m,j} \wedge \mathbf{el\_position}_{l,m})\ \mathbf{U}\ (\bullet\ \neg\ \mathbf{fl\_doors\_closed}_{l,m} \wedge \mathbf{fl\_doors\_closed}_{l,m})$$

System Engineer's Viewpoint

The user's viewpoint holds under all conditions, but there are many refinements to be added to make the system work effectively. For example, having every elevator going from bottom to top and back continuously, stopping at every floor, would satisfy the user conditions described in the previous chapter, including treating all users in a fairly equitable manner. Indeed, by maintaining constant "headways" between the elevators as they move up and down, the service would be very equitable. This, however, would be an inefficient way to operate the elevator system. The systems engineer has to add formulas to further refine the operation of the system, and one of these is described below, after the preliminary definition of a useful function.

The elevator is referred to as being dormant if certain conditions hold, indicating that the elevator has nothing to do just now. An elevator **I**, dormant at floor **m** is represented by the variable $\mathbf{dormant}_{l,m}$. If the elevator is dormant at a floor, and a summons button at that floor is lit, the doors will reopen, with the floor indicator reset to the new direction. If the up and down buttons are pushed simultaneously, we specify a nondeterministic choice as to which directional indicator will be lit.

$$\forall (l,m) \in IM, \mathbf{dormant}_{l,m} = \mathbf{el\_doors\_closed}_l \wedge \mathbf{el\_position}_{l,m} \wedge$$
$$(\forall j \in J, \neg\, \mathbf{el\_direction}_{l,j} \wedge \neg\, \mathbf{fl\_direction}_{l,m,j})$$

$$\forall (l,m) \in IM, (\mathbf{dormant}_{l,m} \wedge \mathbf{fl\_summ\_lit}_{m,down} \wedge \mathbf{fl\_summ\_lit}_{m,up}) \Rightarrow$$
$$\circ((\mathbf{fl\_direction}_{l,m,down} \vee \mathbf{fl\_direction}_{l,m,up}) \wedge \neg \mathbf{el\_doors\_closed})$$

Fairness Conditions

The fairness conditions represent restrictions to the manner in which nondeterministic conditions are handled. The fairness conditions assert that in such cases, the same path will not always be taken. In this specification, there is only one such applicable condition, namely when the system is dormant, and the up and down summons push buttons are backlit simultaneously, the system should not always select one of the directions. The fairness condition must ensure that each time this condition arises, the same direction is not always chosen.

$$\forall (l,m) \in IM, \Box \Diamond (\mathbf{dormant}_{l,m} \wedge \mathbf{fl\_summ\_lit}_{m,down} \wedge \mathbf{fl\_summ\_lit}_{m,up}) \rightarrow \circ\, \mathbf{fl\_direction}_{l,m,down}$$

$$\forall (l,m) \in IM, \Box \Diamond (\mathbf{dormant}_{l,m} \wedge \mathbf{fl\_summ\_lit}_{m,down} \wedge \mathbf{fl\_summ\_lit}_{m,up}) \rightarrow \circ\, \mathbf{fl\_direction}_{l,m,up}$$

If these fairness conditions are not imposed, for example by always preferring the up direction, then the single elevator can be prevented from turning on the floor directional light down, by pushing the up summons button as the doors close, causing the up directional light to come on, and the doors to re-open. Obviously this can continue forever.

The SML Model

The elevator model was built using the State Machine Language (SML) developed by Ed Clarke and associates, and described in [Clarke 87] and other publications. The model consists of four major parts.

1. A first floor, which turns the elevator around, and causes it to wait for a request at the second or third floor.
2. A second floor, which is modeled completely, allowing the elevator to bypass it in either direction if there is no reason to stop, or to stop in response to a summons button or destination button.

3. A third floor, which turns the elevator around, and causes it to wait for a request at the second or first floor.
4. And, of course, the elevator itself.

Model Validation

Obviously the model cannot validate the complete generality of the temporal logic statements, since it could not handle multiple elevators and many floors. It did, however, model the complete operation of a single floor, and its interfaces to the adjacent floors, for a single elevator. The temporal statements were developed before the author was thoroughly familiar with the logic used in the SML/EMC tools, and used a slightly different logic, hence in validating the model, the temporal logic statements in the document were somewhat modified to account for the differences in the logics.

Discussion

The temporal logic specification was produced by a specifier (the author), and reviewed throughly by a single reviewer, who reviewed the specification through many cycles over a period of months.

1. The temporal logic specification proved to be a good communication vehicle between the specifier and the reviewer, even though, in this case, both people were somewhat unfamiliar with temporal logic, and were struggling with the underlying mathematics as well as the elevator specification. The specification was written and went through about three review and rewrite cycles. The first review changed the organization of the report as well as some of the formulas. The remaining reviews concentrated on the formulas and their consistency. Not only could we argue over details of the specification, but we were able to resolve these areas of disagreement precisely.
2. The safety formulas were not only lumped into a single early grouping in the report, but were also expressed early in the derivation process, thus simplifying the expression of the liveness conditions.
3. At one point I considered the systems specification to be an overworking of the problem, which could have been avoided, since the temporal operators used were exclusively *next* and *previous*, rather than *eventually* or *until*. However, building an SML model to satisfy these conditions consistently was a challenging endeavor, so in the end I was convinced that the further detail was not only desirable, but absolutely necessary for success.
4. The early handcrafting of a state machine revealed some underspecification, which would not necessarily have been found in the modeling the system using the SML/EMC tools. Care has to be taken that such underspecification is not simply ignored, but that the specifier (and reviewers) seek out conditions that have not been specified.
5. The model checking revealed a few inconsistencies in the formulas, but, in general the bulk of the work was in changing the state machine model until the formulas were satisfied.
6. Having a verified state machine assists the next stages of design considerably.

Conclusion

The specification of the elevator problem, the building of models describing its behavior, and the checking of the consistency of the specification, all led me to believe in the usefulness of this approach, and convinced me not only that such systems can be specified, but also that they should be specified. Such specifications allow for consistent and thorough reviewing of the details of how the system is to operate, and the modeling allows for consistency checking in fine detail, and leaves one with a system model for developing the next steps in the design. The availability of a supporting toolset allowed me to gain confidence in my ability to apply the technique, and to find flaws in the logic and the model. The specification was understandable to a knowledgeable reviewer, who approached the review with energy and enthusiasm.

Appendix A - Temporal Logic

Temporal logic is a modal logic and each temporal formula is interpreted as true or false against a model of the world. I chose to check the validity of a formula against a state machine in a manner similar to that of (Clarke 84). The state machine SM =(S,R,P) is defined over a set of atomic propositions AP (logical variables), and has the elements described below.

1. **S** is a finite set of states;
2. **R** is a binary relationship on S, defining the possible transitions between states; the arc $(\mathbf{s_i}, \mathbf{s_j}) \in$ **R** indicates that there is a direct path between the states $\mathbf{s_i}$ and $\mathbf{s_j}$. Obviously if there is no such relationship in **R**, then there is no arc connecting the two states. Every state must be connected to at least one other state.
3. **P** assigns to each state the set of atomic propositions true in that state.

A path $\alpha(s_i)$ is an infinite sequence of states starting at state s_i; $\alpha(\mathbf{s_0}) =_{\mathbf{df}} (\mathbf{s_0}, \mathbf{s_1}, \mathbf{s_2}, \ldots)$, such that each adjacent tuple of states $(\mathbf{s_i}, \mathbf{s_{i+1}})$ is in the relationship **R**. $\forall i \ni \mathbf{i \geq 0}, (\mathbf{s_i}, \mathbf{s_{i+1}}) \in$ **R**.

The standard notation for stating that the formula f holds at state s_i in the structure SM is: **SM, $\mathbf{s_i}$ |=f**. When the structure SM is understood, **$\mathbf{s_i}$ |= f** will suffice. The operators used within the body of the paper are from Branching Time Temporal Logic, and are enumerated below.

1. $\mathbf{s_i} \models \mathbf{p}$ **iff** $\mathbf{p} \in \mathbf{P(s_i)}$
2. $\mathbf{s_i} \models \neg\, \mathbf{p}$ **iff** $\mathbf{p} \notin \mathbf{P(s_i)}$
3. $\mathbf{s_i} \models \mathbf{f1} \vee \mathbf{f2}$ **iff** $(\mathbf{s_i} \models \mathbf{f1}) \vee (\mathbf{s_i} \models \mathbf{f2})$
4. $\mathbf{s_i} \models {}^\circ\, \mathbf{f}$ **iff** $\forall \mathbf{j}, (\mathbf{s_i}, \mathbf{s_j}) \in \mathbf{R}, \mathbf{s_j} \models \mathbf{f}$
5. $\mathbf{s_i} \models \bullet\, \mathbf{f}$ **fiff** $\forall \mathbf{j}, (\mathbf{s_j}, \mathbf{s_i}) \in \mathbf{R}, \mathbf{s_j} \models \mathbf{f}$
6. $\mathbf{s_0} \models \Diamond\, \mathbf{p}$ **iff** $\forall$ **paths** $\alpha(\mathbf{s_0})$ $(\exists\, \mathbf{i} \ni \mathbf{i} \geq \mathbf{0}, (\, \mathbf{s_i} \models \mathbf{p}\,)$
7. $\mathbf{s_0} \models [\,]\, \mathbf{p}$ **iff** $\forall$ **paths** $\alpha(\mathbf{s_0})$ $(\forall\, \mathbf{i} \ni \mathbf{i} \geq \mathbf{0}, (\, \mathbf{s_i} \models \mathbf{p}\,)$
8. $\mathbf{s_0} \models \mathbf{p\ U\ q}$ **iff** $\forall$ **paths** $\alpha(\mathbf{s_0})$ $(\exists\, \mathbf{i} \ni \mathbf{i} \geq \mathbf{0}, (\, \mathbf{s_i} \models \mathbf{q}\,) \wedge (\forall\, \mathbf{j} \ni \mathbf{0} \leq \mathbf{j} \leq \mathbf{i}, (\mathbf{s_j} \models \mathbf{p})))$

References

[Barringer 87]	Barringer, H. Up And Down the Temporal Way. *Computer Journal* 30(2):134-148, April, 1987.
[Clarke 86]	Clarke, E.M., Emerson, E.A., and Sistla, A.P. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244-263, April, 1986.
[Clarke 87]	Clarke, E. M. and Grumberg, O. Research on Automatic Verification of Finite-State Concurrent Systems. *Ann. Rev. Computer Science* (2):269-290, 1987.
[Hoare 85]	Hoare, C.A.R. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, New Jersey, 1985.
[Lamport 83]	Lamport, L. What Good is Temporal Logic. *Information Processing 83* :657-668, 1983.
[Pnueli 85]	Pnueli, A. Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends. *Lecture Notes in Computer Science #224* :510-584, 1985.
[Rescher 71]	Rescher, Nicholas and Urquhart, Alasdair. *Temporal Logic.* Springer-Verlag, Wien, New York, 1971.
[Schwartz 87]	Schwartz, M. D. and Delisle, N. M. Specifying A Lift Control System With CSP. In *Fourth International Workshop on Software Specification and Design*, pages 21-27. The Computer Society of the IEEE, April, 1987.
[Wood 89]	Wood, W. *A Temporal Logic Case Study.* Technical Report CMU/SEI-89-TR-24, Software Engineering Institute, 1989.
[Woodcock 87]	Woodcock, J. C. P., King, S., and Sorensen, I. H. Mathematics for Specification and Design: The Problem with Lifts. In *Fourth International Workshop on Software Specification and Design*, pages 265-268. The Computer Society of the IEEE, April, 1987.