

# Model-Checker-Based Testing of LTL Specifications

Luis García and Steve Roach  
*The University of Texas at El Paso*  
*garciala@us.ibm.com, sroach@utep.edu*

## 1. Introduction

Formal verification techniques for software typically start with formal specifications of software behavior. Several automated tools use Linear Temporal Logic (LTL) to specify behavioral properties of software. A significant hurdle to the use of formal approaches is the development of formal specifications.

The Specification Pattern System (SPS) by Dwyer [1,2] is a set of recurring types of specifications identified in industrial applications. SPS introduces *patterns* and *scopes*. A pattern is a specification template describing the software behavior, and scope is the extent of software execution over which the pattern must hold. Patterns are categorized into two major groups: *occurrence* and *order*. An example of a pattern is the *Response* pattern, which states that some event is a response to some trigger. An example scope is the *AfterL* scope, which indicates that the specified property must hold after some enabling event or condition.

Mondragon extended SPS to include composite propositions (CPs) [3,4], which describe the relationships between two or more propositions. This extension enables a user to specify, for example, that two atomic propositions must hold in a given state or that a sequence of propositions must hold in a given order. Mondragon defined eight CP classes to describe sequential and concurrent software. CP classes have been defined representing *conditions* and *events*. Conditions describe concurrent behavior, and events describe synchronization. For example, the *AtLeastOne(g)* class indicates that *g* is a set of propositions, at least one of which must hold.

The *Prospec* [3,5] software tool guides practitioners in the creation of formal specifications by assisting the user in the identification of appropriate specification patterns, the scope over which a property must hold, and the identification of events or conditions that are used to specify behavior.

Pattern-based specifications that contain multiple propositions are called *complex pattern-based*

*specifications*. It is not possible to use direct substitution of CPs into simple pattern-based specification to generate complex pattern-based specifications using SPS. Salamah [6] generated a new set of general pattern/scope templates like the templates generated by Dwyer that allow direct substitution of CPs into pattern-based templates to generate LTL formulas. Salamah provides informal proofs of the correctness of these formula templates. The work described here further verifies the templates through *model checker-based testing*, a general method for testing software specifications based on SPS patterns and composite propositions.

## 2. Execution Traces

An execution trace is a sequence of states represented here by a sequence of positions, read left to right, where each position indicates a state. The spaces are filled with the atomic propositions that are true in that state. A dash indicates no proposition is true. If more than one proposition is true, they are written between parentheses. Table 1 lists the usual LTL operators with example and counter example traces.

**Table 1: Linear Temporal Logic Operators**

Operator	Example Trace	Counter Example
$\neg a$	-----	a-----
$a \wedge b$	(ab)-----	-----ab-----
$a \vee b$	a-----	-----
$a U b$	aaaab-----	Aaa----b----
$b X a$	ba-----	b----a-----
$\diamond a$	-----a-----	-----
$\square a$	aaaaaaaa	aaaa—aaaa

## 3. Model-Checker-Based Specification Verification

Ordinarily, a model checker is used to verify that a computation system satisfies a specification. However, in this work, it is necessary to verify that the concrete specifications generated by the *Prospec* tool accurately

reflect the intent of the user by matching the natural language descriptions given for the patterns. To achieve this, we will assume that the model is correct and use the model checker to verify the formula.

This paper introduces the *Property Testing Tool (Protest)*, which automatically generates and tests formulas representing software specifications, in particular, specifications based on SPS and composite propositions. *Protest* takes an execution trace and builds a model of the trace. It then uses the NuSMV model checker to test whether the given formula is satisfied by the model. The output of the model checker is compared to the expected result to determine if the test case passes or fails.

The key ideas here are that the generated models are simple enough that they can be validated by inspection and that the test cases can be reused. The model generated by *Protest* is a simple, linear model. For example, given this execution trace,

```
---P1P2P3---R1R2R3---
```

*Protest* generates the following NuSMV model:

```
MODULE main
VAR Q : seq();
DEFINE
    P1 := (Q.State = 3);  P2 := (Q.State = 4);
    P3 := (Q.State = 5);  R1 := (Q.State = 9);
    R2 := (Q.State = 10); R3 := (Q.State = 11);
MODULE seq()
VAR State: 0..14;
ASSIGN init(State):= 0;
    next(State):= case
        (State != 14) : {State + 1};
        (State = 14)  : {14};
    esac;
```

This model might be used to test this specification of the absence of P before R, where P and R are both CPs in the *AtLeastOne* class:  $!((!(R1 | R2 | R3)) \cup (((P1 | P2 | P3) \& !(R1 | R2 | R3)) \& (F (R1 | R2 | R3))))$

## 4. Results

While Salamah has provided informal proofs of the correctness of the templates, *Protest* was used to test a subset of the templates, namely the *Absence* pattern and the *Before R* scope, and two choices of composite propositions classes, *AtLeastOne<sub>C</sub>* and *Eventually<sub>E</sub>*. A test suite was developed that included example and counter example traces using boundary value analysis and equivalence class testing. In all, 87 tests were generated and executed.

The initial results of the testing indicated a number of discrepancies between the result computed by the

model checker and the expected value. These were trace to three root causes: (1) errors in interpretations of the templates; (2) typographic and conceptual errors in the expected results (the test cases themselves were in error); and (3) errors in the templates used. In the third case, the difficulties were traced to the use of an early version of the templates, and Salamah had identified the problems with the templates when constructing the informal proofs.

## 5. Future Uses

The *Protest* tool provides the formal methods community with three principle uses. First, the execution traces used to test the templates will be integrated into *Prospec* to assist users in the specification of properties and in the testing of the properties generated by *Prospec*. Second, the *Protest* approach is a valuable tool for teaching LTL. It can be used to experiment with LTL formulas so that students can gain an understanding of the semantics of LTL and test formulas they write. Third, the *Protest* approach can be extended to other logic formalisms such as CTL.

## 10. References

- [1] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C., "A System of Specification Patterns," Proc.of the 2nd Workshop on Formal Methods in Software Practice, Sept.1998.
- [2] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C., "Patterns in Property Specification for Finite-State Verification," Proc.of the 21st Intl. Conference on Software Engineering, Los Angeles, CA, USA, 1999, 411-420.
- [3] Mondragon, O.A., "Elucidation and Specification of Software Properties through Patterns and Composite Propositions to Support Formal Verification Techniques" Dissertation, Computer Engineering Department, University of Texas at El Paso, 2004.
- [4] Mondragon, O. and Gates, A.Q., "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," Intl. Journal Software Engineering and Knowledge Engineering, 14(1), Feb. 2004.
- [5] Mondragon, O., Gates, A., and Roach, S., "Prospec: Support for Elicitation and Formal Specification of Software Properties," in Proc. of Runtime Verification Workshop, ENTCS, 89(2), 2004.
- [6] Salamah, S.I., "Generating Linear Temporal Logic Formulas For Complex Pattern-Based Specifications" Dissertation, Computer Science Department, University of Texas at El Paso, 2007.