Verifying Pattern-Generated LTL Formulas: A Case Study

Salamah Salamah[1], Ann Gates, Steve Roach, and Oscar Mondragoon[2]

[1] Computer Science Department, University of Texas at El Paso, El Paso TX 79902, USA,
salamah, agates, sroach@cs.utep.edu,
[2] ESICenter, Guadalajara, Mexico
oscar.mondragon@itesm.mx,

Abstract. The Specification Pattern System (SPS) and the Property Specification (Prospec) tool assist a user in generating formal specifications in Linear Temporal Logic (LTL), as well as other languages, from property patterns and scopes. Patterns are high-level abstractions that provide descriptions of common properties, and scopes describe the extent of program execution over which the property holds. The purpose of the work presented in this paper is to verify that the generated LTL formulas match the natural language descriptions, timelines, and traces of computation that describe the pattern and scope. The LTL formulas were verified using the spin model checker on numerous test cases. A test case is an LTL formula and a sequence of Boolean valuations. The LTL formulas were those generated from SPS and Prospec. The Boolean valuations of propositions in the LTL formula are generated by a deterministic, single-threaded Promela program that was run using the software model-checker spin. For each pattern, a suite of test cases was developed using boundary value analysis and equivalence class testing strategies. The experiments uncovered several errors in both the SPS-generated and the Prospec-generated formulas.

1 Introduction

Tools that simplify the creation of Linear Temporal Logic LTL formulas are of interest to the model checking as well as other communities, but only if the formulas produced match the intent of the specifier. This paper describes the verification of LTL formulas generated by two specification approaches: the Property Specification Patterns system (SPS)[2, 3] and Property Specification (Prospec)[11, 12].

These two approaches assist in the specification of properties and assist users in generating formal specifications in LTL, as well as other languages, from property patterns and scopes. Patterns are high-level abstractions that provide descriptions of common properties, and scopes describe the extent of program execution over which the property holds. Previous verification of the Prospec LTL formulas was obtained using the spin model checker on various test cases,

but without a disciplined strategy for test case selection. Previous verification of the SPS LTL formulas utilized peer reviewers and verification through Finite State Verification tools[14].

The verification described here used the software model checker spin on test cases that were systematically developed based on equivalence class analysis and boundary value analysis. This use of spin differs from traditional model checking. The model used here produces a simple finite state automaton with exactly one possible execution trace and only a few states. This model is used as a test bed for the analysis of LTL formulas. The simplicity of the model makes inspection feasible. The approach to validation of the generated LTL formulas is to create test cases for which the desired output is obvious, even if the meaning of the LTL formula is not immediately so. A test case is an LTL formula and a sequence of Boolean valuations. The LTL formulas were those generated from SPS and Prospec. The Boolean valuations of propositions in the LTL formula are generated by a deterministic, single-threaded Promela program that was run using spin. For each pattern, a suite of test cases was developed using boundary value analysis and equivalence class testing strategies. The experiments uncovered several errors in both the SPS-generated and the Prospec-generated formulas.

This paper first presents a background on SPS and Prospec. These sections present the verbatim documentation that is made available to the user. The next section describes the testing process that was used to verify the formulas. Last, the results are described followed by a discussion. Appendix A gives the complete set of test cases for the $Absence\text{-}Before$ R and $Precedence\text{-}Between$ L *and* R patterns respectively. While Appendix B gives couple of screen-shots of the Prospec tool, and finally Appendix C gives examples of failed test cases for both SPS and Prospec.

2 Background

2.1 Property Specification Patterns System

SPS defines patterns and scopes to assist the practitioner in formally specifying software properties. Patterns capture the expertise of developers by describing solutions to recurrent problems [4]. Each pattern describes the structure of specific behavior, defines the patterns relationship with other patterns, and defines the scope over which the property holds. The SPS website provides descriptions of the patterns, including intent, relationships, and known uses. After the user selects a pattern and a specification language, the website displays a mapping for each scope in the chosen language. The target specification languages include: Linear Temporal Logic (LTL) [7], Computational Tree Logic (CTL) [6], and Graphical Interval Logic (GIL) [1, 5]. In the definitions given in this section, P, T, L, and R represent propositions.

The main patterns defined by SPS are: *universality*, *absence*, *existence*, *precedence*, and *response*. The descriptions given below are taken verbatim from the SPS website.

- *Absence*: To describe a portion of a system's execution that is free of certain events or states.
- *Universality*: To describe a portion of a system's execution which contains only states that have a desired property. Also known as Henceforth and Always.
- *Existence*: To describe a portion of a system's execution that contains an instance of certain events or states. Also known as Eventually.
- *Precedence*: To describe relationships between a pair of events/states where the occurrence of the first is a necessary pre-condition for an occurrence of the second. We say that an occurrence of the second is enabled by an occurrence of the first.
- *Response*: To describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause, must be followed by an occurrence of the second, the effect. Also known as Follows and Leads-to.

Precedence and *response* patterns include $chain - precedence$ and $chain - response$ patterns that permit specifying a sequence of propositions. SPS restricts the specification of sequences to *precedence* and *response* patterns. In SPS, each pattern is associated with a scope that defines the extent of program execution over which a property pattern is considered. There are five types of scopes defined in SPS, as shown in Figure 1

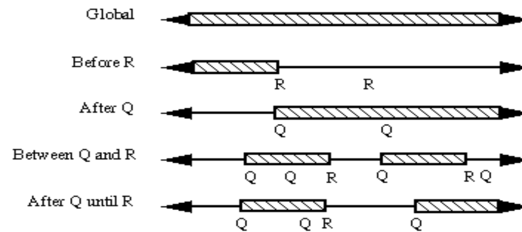

Fig. 1. Scopes in SPS.

The description of the scopes is described verbatim from the scopes section of the SPS website [15]:

> Each pattern has a scope, which is the extent of the program execution over which the pattern must hold. There are five basic kinds of scopes: global (the entire program execution), before (the execution up to a given state /event), after (the execution after a given state/event), between (any part of the execution from one given state/event to another given state/event) and after-until (like between but the designated part of the execution continues even if the second state/event does not occur). The scope is determined by specifying a starting and an ending

> state/event for the pattern: the scope consists of all states/events beginning withthe starting state/event and up to but not including the ending state/event.
>
> The figure above illustrates the portions of an execution that are designated by the different kinds of scopes. We note that a scope itself should be interpreted as optional; if the scope delimiters are not present in an execution then the specification will be true.
>
> Before and after scopes for our patterns are interpreted relative to the first occurrence of the designated state/event. We have done this because it matches our experience with real specifications. Note, however,that we could just as easily interpret these scopes relative to the last occurrence of the designated state/event (the mappings given in the patterns are easily transformed to match this interpretation). At present we do not see the need for supporting both first and last occurrence scopes, but as we gain experience applying the patterns we may wish to extend scopes in this way.
>
> Scope operators are not present in most specification formalisms (interval logics are an exception). Nevertheless, our experience strongly indicates that most informal requirements are specified as properties of program executions or segments of program executions. Thus a pattern system for properties should mirror this view to enhance usability.

Note that to facilitate discussions about SPS and Prospec, we use L to denote Q in the $After\ Q$, $Between\ Q\ and\ R$, and $After\ Q\ until\ R$ scopes.

2.2 Prospec

The Property Specification tool (Prospec) [16, 17, 18] builds on SPS by facilitating the identification of SPS patterns and scopes as well as validation of specifications. Prospec displays traces of computation to illustrate the subtle differences among different scopes and patterns, and it displays a decision tree to guide the user through a series of decisions to support the user in selecting an appropriate scope or pattern. Given a computation represented as a sequence of states, and a finite set of events E, a trace of computation is a list indicating, for each moment of time t, which events from the set E occur at t. Figure 4 in Appendix B shows the Prospec window for selecting a pattern and the traces of computation that are given to elucidate each pattern. Tables 1 and 2 give the characteristics for pattern and scope, respectively, that appear in the Prospec tool to assist the user in understanding the pattern and scope.

Prospec extends SPS by introducing a classification for defining sequential and concurrent behavior. As described earlier, there are a number of interpretations of the property in the ATM example. For example, assume that a denotes "the user's account is updated," m denotes "money is dispensed," r denotes "the receipt is printed," and c denotes "the user's ATM card is returned." Interpretations of the property could be represented in LTL as follows:

$$\Diamond(a \wedge m \wedge r \wedge c) \tag{1}$$

Table 1. Summary of Prospec's patterns characteristics

PATTERN	**CHARACTERISTICS**
Absence of (P)	1) Event or condition P does not hold within the states defined by the scope of interest. 2) The *absence* property is also known as alarm.
Existence of (P)	1) Event or condition P holds at least once within the states defined by the scope of interest. 2) The *existence* property is also known as eventually.
Universality of (P)	1) Event or condition P holds in every state of the scope of interest. 2) The *universality* property is also known as safety or invariant.
(T) *Precedes* (P)	1) T holds before P holds, where T and P are events or conditions 2) T may hold several times before P holds 3) P does not hold before T holds 4) P does not hold at the same state at which T holds 5) If T holds, then P may or may not hold 6) If T holds, then T does not hold when P holds 7) The *precedence* property represents a cause-effect relation, where T denotes a cause and P denotes an effect 8) There is no effect P without a cause T 9) T precedes P is also know as T before P
(T) *Responds* to (P)	1) P must be followed by T, where P and T are events or conditions 2) Some T follows each time that P holds 3) The same state at which T holds may follow two or more states at which P holds 4) T may hold at the same state as P holds 5) If T holds, then P may or may not hold at a previous state 6) The *response* property represents a cause-effect relation, where P denotes a cause and T denotes an effect 7) If cause P holds, then at some future state effect T holds 8) T responds to P is also know as T follows P

$$(a \rightarrow \diamond c) \wedge (c \rightarrow \diamond r) \wedge (r \rightarrow m) \tag{2}$$

$$a \wedge \mathbf{O} \diamond (c \wedge \mathbf{O} \diamond (r \wedge \mathbf{O} \diamond m)). \tag{3}$$

There are subtle differences among the formulas. Formula (1) asserts that the propositions are *true* simultaneously in some future state, but not necessarily at the same time in the computation. If one wants to specify an ordering in which the propositions hold (i.e., c follows a, r follows c, and m follows r), then formula (2) may appear to capture the ordering; however, this formula also states that if a, c, and r hold, then they must be true at the same state. Formula (3) captures the ordering as follows: proposition a holds; after a holds, then at some future state c holds; after c holds, then at some future state r holds; and after r holds, then at some future state m holds.

In SPS, the user would need to replace the parameters in the formula generated by the selected pattern and scope to include propositions a, r, c, and m. Clearly, the refinement of the specification can be challenging as illustrated by formulas (1) through (3). To address this, Prospec extends the functionality of SPS by including composite propositions (CP) as shown in Figure 5 in Appendix B. The CP taxonomy [9, 12] categorizes and defines the structure of multiple propositions to capture sequential and concurrent behavior. The taxonomy can be used in the property elicitation and specification process to guide practitioners in formally specifying properties, eliminating the subtleties associated with multiple events and conditions. When relations have not been carefully analyzed, composite propositions can expose incompleteness or ambiguities.

CP defined as conditions are used to describe concurrency, and those defined as events are used to describe activation or synchronization of processes or actions. The formal syntax and semantics for CP classes is given in Mondragon and Gates [11]. CP can be used to define boundaries of scopes and patterns with multiple propositions. For instance, an ordered sequence can define the left boundary of an after L scope, and multiple events can define the cause part of a response pattern.

2.3 LTL Formulas of SPS and Prospec

Table 3 presents the SPS LTL mappings for each pattern and scope combination and those generated by Prospec. In the table, the parameter for the left boundary is referred to as L rather than Q as in SPS. It is important to note that the SPS formulas for the *absence* pattern were rewritten to facilitate the specification of sequences of events in Prospec. The SPS formula for the *absence* pattern ($[](\neg p)$) holds if P does not hold at any state of the computation. If P is an event, then it is not possible to assert that P does not hold at any state of the computation. By using the LTL equivalence ($[]p \equiv \neg \diamond (\neg p)$), the absence pattern is expressed as: $(\neg \diamond (P))$. The latter formula holds when it is not the case that P holds at some state of the computation. This formula provides a more intuitive specification of properties such as the absence of a sequence of events.

Table 2. Summary of scopes characteristics

SCOPE	CHARACTERISTICS
Global	1) The scope denotes the entire computation. 2) The scope includes all the states in the computation. 3) The interval defined by the scope occurs once in a computation
Before R	1) The scope denotes a subsequence of states or events (an interval) that begins with the start of computation and ends with the state or event immediately preceding the event or state at which R holds for first time in the computation. 2) The interval does not include the state or event associated with R. 3) The interval defined by the scope occurs once in a computation. 4) One or more events (conditions) may be associated with R; a condition is a proposition and an event is a change in value of the proposition from one state to the next.
After L	1) The scope denotes a subsequence of states or events (an interval) that begins with the first event or state at which L holds and ends with termination of computation. 2) The interval includes the state or event associated with L 3) The interval defined by the scope occurs once in a computation. 4) One or more events (conditions) may be associated with L; a condition is a proposition and an event is a change in value of the proposition from one state to the next.
Between L and *R*	1) The scope denotes a subsequence of states or events (an interval) that begins when L holds and ends with the state or event immediately preceding the event or state at which R holds. 2) Event or condition L must hold and, at a different event or state in the future, R must hold. 3) The interval includes the state or event associated with L 4) The interval does not include the state or event associated with R. 5) The interval defined by the scope may occur more than once in a computation. 6) Multiple intervals may be defined within an interval when L holds more than once before R holds 7) One or more events (conditions) may be associated with L and R
After L Until R	1) The scope denotes a subsequence of states or events (an interval) that begins when L holds and ends either with the state or event immediately preceding the event or state at which R holds, or begins when L holds and ends with the termination of computation. 2) The interval includes the state or event associated with L 3) The interval does not include the state or event associated with R 4) The interval may repeat during a computation. 5) If L holds and R does not hold, the interval ends with termination of a computation. 6) The interval defined by the scope may occur more than once in a computation. 7) Multiple intervals may be defined within an interval when L holds more than once before R holds 8) One or more events (conditions) may be associated with L and R

Table 3. LTL Formulas for patterns and scopes

PATTERN	SCOPE	PROSPEC	SPS
Absence	*Global*	$\neg(\diamond P)$	$[](\neg P)$
	Before R	$\diamond R \rightarrow \neg(\neg(R) U P)$	$\diamond R \rightarrow (\neg P U R)$
	After L	$\neg(L) W (L \wedge \neg(\diamond P))$	$[](L \rightarrow [](\neg P))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow \neg(\neg(R) U P))$	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(P) U R))$
	After L Until R	$[](L \wedge \neg(R) \rightarrow \neg(\neg(R) U P))$	$[]((L \wedge \neg(R)) \rightarrow (\neg(P) W R))$
Existence	*Global*	$(\diamond P)$	$(\diamond P)$
	Before R	$\diamond R \rightarrow (\neg(R) U (P \wedge \neg(R)))$	$\neg R W (P \wedge \neg R)$
	After L	$\neg(L) W (L \wedge (\diamond P))$	$[](\neg L) \vee \diamond(L \wedge \diamond P))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(R) U (P \wedge \neg(R))))$	$[]((L \wedge \neg(R)) \rightarrow (\neg(R) W (P \wedge \neg R)))$
	After L Until R	$[]((L \wedge \neg(R)) \rightarrow (\neg(R) U (P \wedge \neg(R))))$	$[]((L \wedge \neg(R)) \rightarrow (\neg(R) U (P \wedge \neg R)))$
Universality	*Global*	$[]$P	$[]$P
	Before R	$\diamond R \rightarrow (P U R)$	$\diamond R \rightarrow (P U R)$
	After L	$\neg(L) W (L \wedge []P)$	$[](L \rightarrow [](P))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (P U R))$	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (P U R))$
	After L Until R	$[]((L \wedge \neg(R)) \rightarrow (P W R))$	$[]((L \wedge \neg(R)) \rightarrow (P W R))$
Presedence	*Global*	$\neg(P) W T$	$\neg(P) W T$
	Before R	$\diamond R \rightarrow (\neg(P) U (T \vee R))$	$\diamond R \rightarrow (\neg(P) U (T \vee R))$
	After L	$\neg(L) W (L \wedge (\neg(P) W (T)))$	$[]\neg L \vee \diamond(L \wedge (\neg P W T))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(P) U ((T \vee R)))$	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(P) U ((T \vee R)))$
	After L Until R	$[](L \wedge \neg(R) \rightarrow (\neg(P) W (T \vee R)))$	$[](L \wedge \neg(R) \rightarrow (\neg(P) W (T \vee R)))$
Response	*Global*	$[](P \rightarrow \diamond T)$	$[](P \rightarrow \diamond T)$
	Before R	$\diamond R \rightarrow ((P \rightarrow (\neg(R) U (T \wedge \neg(R))))) U R)$	$\diamond R \rightarrow ((P \rightarrow (\neg(R) U (T \wedge \neg(R))))) U R)$
	After L	$(\neg L) W (L \wedge [](P \rightarrow \diamond T))$	$[](L \rightarrow [](P \rightarrow \diamond T))$
	*Between L and*R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (P \rightarrow (\neg(R) U (T \wedge \neg(R)))) U R$	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (P \rightarrow (\neg(R) U (T \wedge \neg(R)))) U R$
	After L Until R	$[]((L \wedge \neg(R) \rightarrow (P \rightarrow (\neg(R) U (T \wedge \neg(R)))) W R)$	$[]((L \wedge \neg(R) \rightarrow (P \rightarrow (\neg(R) U (T \wedge \neg(R)))) W R)$
Strict Presedence	*Global*	$\neg(P) W (T \wedge \neg(P))$	N/A
	Before R	$\diamond R \rightarrow (\neg(P) U ((T \wedge \neg(P)) \vee R))$	N/A
	After L	$\neg(L) W (L \wedge (\neg(P) W (T \wedge \neg(P))))$	N/A
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(P) U ((T \wedge \neg(P)) \vee R)))$	N/A
	After L Until R	$[](L \wedge \neg(R) \rightarrow (\neg(P) W ((T \wedge \neg(P)) \vee R)))$	N/A

In addition, the formulas for the *Precedence* pattern were rewritten to define a *strict-precedence* pattern, i.e., S *strictly precedes* P, where S and P are events or conditions. Unlike the SPS *precedence* pattern, S and P cannot hold at the same state in this pattern. While the *precedence* pattern ($\neg PWS$) holds even when S and P hold at the same state, the Prospec formula ($\neg PW(SU\neg P)$) stays true to the SPS description that states "Precedence says that some cause precedes each effect." Use of *Strict Precedence* enforces that S and P cannot hold at the same state.

Prospec formulas for patterns with the $After$ scope differ from SPS formulas. While the SPS formula for the *universality* pattern within $After$ scope considers every state at which L holds, the Prospec formula only considers the first state at which L holds.

3 Verification Methodology

Tools that automatically generate specifications require assurance that the results are correct. According to the SPS website, the SPS specifications were verified by reviews conducted with experts in the language. The Prospec specifications were verified using the spin model checker on numerous test cases, where a test case used the generated LTL formula and a sequence of Boolean valuations. The Boolean valuations of propositions in the LTL formula are generated by a deterministic, single-threaded Promela program that was run using spin. For each pattern, a suite of test cases was developed. The work presented in this paper followed the Prospec approach to verifying the formulas, but it used a more systematic approach to defining the test cases by incorporating boundary value analysis and equivalence class testing strategies. This section describes the verification process.

3.1 Test Process

Each pattern is associated with a test suite. The test suite consists of a set of test cases for each scope, where a test case is a set of assignments for the propositions in the LTL formula under test such that a particular trace of computation is created. In other words, each test case will result in a particular trace of computation for an LTL formula associated with a pattern and scope combination. The test suites for both SPS and Prospec are identical, although for some patterns and scopes the expected results may differ.

The Promela code consists of a do-loop that begins with the initial value of i set to zero and terminates when i reaches a predefined value called *limit*. Setting the value of *limit* to 20 supports construction of a variety of test cases that are dependant on the value of i to create a desired trace of computation. The Promela Code follows:

```
#define limit 20
byte i = 0;
```

```
proctype seq( )
{
  do
  ::  (i < limit) -> i = i+1;
  ::  (i == limit) -> break;
  od;
  progress: assert ( i == limit )
}
init
{
  run seq()
}
```

Execution of a test case consists of assigning conditions to propositions, defining an LTL formula, and model checking the Promela code. The Promela code remains the same in all test cases. Assignments of conditions to propositions are done in the Xspin 3.4.2's LTL Property Manager. For example, in the test case where L is true in the second state, P is true in the seventh state, and R is true in the twelfth state, the following definitions are made: # define L (i == 1), # define P (i == 6), and # define R (i == 11). Figure 2 presents an example test case for the pattern $Existence(P)$ with scope *Between* L and R. In this example, the test number is the number of the test in the test suite. The documentation includes a representation of the trace of computation to be tested. For example, in the trace "- - - - - L - - - - - P - - - R - - - - -" each character in the string represents a state. A dash (-) implies that none of the propositions is *true* at that state, a letter symbol, e.g., L, P, and R in this example, denotes that the proposition is *true* in the designated state. For example, in the trace above, L holds in state 5, and P holds in state 11. Displaying two letters between parenthes implies that both propositions are valid at that state. For instance, in trace "- - (pq) - - r ", p and q both hold in state 2. The test case documentation also includes the name of the pattern and scope being tested as well as the LTL formula that was given by SPS or generated from Prospec.

The LTL formula is tested by the spin model checker. Because spin does not support the weak until (W) operator, the LTL equivalence was used instead. For example, $aWb \equiv []a \vee (aUb)$. Next, the documentation gives the assignment for the propositions in the formula. These assignments are based on variable i in the Promela code and are selected so that the trace of computation is generated. In the example in Figure 2, L is defined as (i == 3), i.e., proposition L holds when variable i becomes 3 within the iteration. The Interval designation is used to document whether an interval(s) can be built in the trace of computation given by the test cases. This is relevant for all scopes except *Global*. Last, the test case documentation shows the expected results based on the natural language description and timelines provided by the respective system or tool and the actual results that were returned by the spin model checker.

The test cases were constructed using two testing strategies: equivalence class analysis and boundary value analysis. Equivalence class analysis partitions test

Test 2 : - - -L - - P - - R- - - - -
Pattern : Existence (P)
Scope : Between L and R
Formula: [] ((L && $(\neg R)\&\&(\diamond R)) \rightarrow (\neg((\neg R)U(P\&\&(\neg R))))$)
L: (i == 3)
P: (i == 6)
R: (i == 9)
Interval: Yes
Expected Result: No violation
Actual Result: No violation

Fig. 2. Sample Test Case

cases into sets from which exemplars are selected. It assumes that test cases from a partition are equally likely to expose an error. For example, the two traces of computation "L - P - - - - - -" and "L - - - - - - P - -" belong to the same equivalence class in the suite $existence(P)$ - $After\ L$. The test sets for the *Absence* (P) and *Existence* (P) patterns were partitioned at the upper level into two sets: 1) P does not hold in any state of the interval(s) and 2) P holds in some state of the interval(s). The test sets for the T *precedes* P (T *responds* to P) scopes were partitioned at the upper level into two sets: 1) T does not precede P (T does not respond to P) in any state of the interval(s) and 2) T precedes P (T responds to P) in the interval (s). Note that the partitions for the $Universality$ pattern and for *Global* scope are slight variations of the above partitions. The next level of partitions is based on the intervals defined by the scope as follows:

- $Before\ R$ ($After\ L$)-Partition 1:
  - R (L) holds in the first state
  - R (L)holds in the last state
  - R (L)holds in other states
  - the interval is not built
- $Before\ R$ ($After\ L$)-Partition 2:
  - L holds in the first state (not applicable for R)
  - R (L) holds in the last state
  - R (L) holds in other states
- *Between* L *and* R ($After\ L\ Until\ R$)-Partition 1:
  - the interval is not made
  - a single interval is made and
    - L holds in the first state
    - R holds in the last state
  - multiple intervals are made
  - nested intervals are made
- *Between* L *and* R ($After\ L\ Until\ R$)-Partition 2:
  - a single interval is made and
    - L holds in the first state

  * R holds in the last state
  * L and R hold in other states
- multiple intervals are made
- nested intervals are made

The test cases defined in each sub-partition are based on boundary value analysis strategies. For example, consider the following three test cases that are associated with *Absence* - *After* L and the sub-partition "L holds in other states" under Partition 2.

1. - - - - - - - - - P L - - - - - - - - - -
2. - - - - - - - - - - (LP) - - - - - - - - - -
3. - - - - - - - - - - L P - - - - - - - - -

Test case 1 is valid and the last two test cases are not valid. Test case 2 checks that P holds in the first state in which L becomes true. Test case 2 is also used for a $Before$ R test case, where L is replaced by R. In this case, the test would be valid. Tables 4 and 5 in Appendix A give the complete set of test cases for the *Absence-Before R* and *Precedence-Between L and R* patterns respectively. The complete set of test cases for all patterns and scopes combinations can be found in Salamah [13]. Test cases that check conditions that can be verified through other test cases were eliminated. For example, in the *T precedes P* pattern for the *Between L and R* scope, the following test case was eliminated: $---L(PT)---R$. There are two things that are being checked by this test case: it checks that strict precedence is upheld, and it checks that the interval is still built when R occurs in the last state of the computation. These two conditions are covered by test cases 2(a)(vi) and 2(b)(iii) as given in Table 5 of Appendix A.

4 Results and Discussion

Of the 25 formulas created using SPS, 16 were verified. Of the 30 formulas created using Prospec, 27 were verified based on interpretation of the documentation provided with each of these tools. Since the documentation is written in English, the potential for ambiguity is always present. Prospec has now been modified to generate formulas that align with the documentation. The following subsections discuss the errors detected by the verification effort and suggests ways to align SPS formulas to the documentation.

4.1 Prospec Errors

The testing of the Prospec generated formulas revealed errors in three formulas, all of which are in the *Absence* (p) pattern. The affected scopes are: $Before$ R, *Between L and R*, and *After L until R*. The test cases, which identified the errors, created an execution trace in which P holds at the same state as the right boundary R. In each case, spin reported a violation. According to Prospec's documentation, scopes consist of all the states beginning with, and including, the left boundary and ending with, but *not* including,

the right boundary. Figure 5 shows an example test case where a Prospec's generated formula failed. The complete list is included in Appendix C.
To address the Prospec's errors, the affected formulas were changed as follows:

- *Absence Before R*: $\diamond R \rightarrow (\neg(\neg(R)U(P \wedge \neg R)))$
- *Absence Between L and R*: $[]((L \wedge (\neg R) \wedge (\diamond R)) \rightarrow (\neg(\neg(R)U(P \wedge \neg R))))$
- *Absence After L until R*: $[]((L \wedge (\neg R)) \rightarrow (\neg(\neg(R)U(P \wedge \neg R))))$

The formulas were tested with the test cases from the *Absence* suite. No violation were reported.

4.2 SPS Errors

The testing of the SPS generated formulas revealed errors in 9 formulas due to ambiguity of the documentation. The formulas that had questionable behavior were those generated from the scopes *Between Q and R* and *After Q until R* for the *Existence* and *Precedence* patterns. In addition, the formulas for the *Response* pattern in all the scopes did not behave as expected.
In the *Between Q and R* and *After Q until R*, the documentation implies that the proposition must hold only in the interval built by the first occurrence of Q and R, and not in the subintervals within the outer interval. Examination of the Patterns Scopes figure for the *After Q* timelilne, given in Section 2.1, shows two Qs in the interval. More importantly, the SPS documentation states: "Before and after scopes for our patterns are interpreted relative to the first occurrence of the designated state/event." This implies that the second Q is not being considered. Because the timelines for the *Between Q and R* and the *After Q until R* scopes are similar to the diagram for the *After Q* scope, one would expect the same interpretation of these scopes to hold. There is not a detailed description of these scopes to indicate otherwise. Inspection of the formulas for the *Absence*, *Universality* , and *Response* patterns in the *After Q* scope given in Table 3, however, shows that the pattern is checked after each occurrence of Q and not just the first occurrence of Q. A list of violated test cases in Appendix C show that the *existence* and *precedence* patterns in the *Between L and R* and *After L until R* are also being checked in all subintervals that are formed. For example; in the *existence Between L and R* the test "- - - L - - - P - - - L - - - R - -" returns a violation, even though the documentation of SPS indicates that the occurrence of one P within the outer interval should be enough. However, a violation is returned since P does not hold in the second interval. The only way for a user to know that the pattern is checked within subintervals is to examine the complex formulas that are generated. We would argue that the intent of SPS and Prospec is to assist professionals, who may not be experts in a particular language, in creating formal specifications. Two approaches can be used to correct this: clarify the documentation to reflect the actual behavior of the formula, or revise the formula.

The ambiguity in the documentation concerning the behavior of the Response pattern can also lead to errors. The "Intent" section for the Response pattern as provided in section 2.1 states that the intent of the Response pattern is "To describe relationships between a pair of events/states where the occurrence of the first is a necessary pre-condition for an occurrence of the second. We say that an occurrence of the second is enabled by an occurrence of the first." In addition, the "Relationship" section of the Response pattern [16] states: "...Response says that some effect follows each cause." The English interpretation of the word "follow," according to Merriam Webster [8], is "to come or take place after in time, sequence, or order." The generated formula allows for S and P to occur in the same state in the case of S responds to P, as validated by test cases shown in Appendix C. In LTL this is a valid interpretation of "follow" since the current state is considered part of the future. This subtlety will be missed by someone who is not an expert in LTL and, as a result, should be clearly stated.

Prospec Failed Test	**SPS Failed Test**
Test 6 : - - - - - (RP) - - - - - - - - - - - - - - -	Test 26 : - - - - - L - - - P - - L - - - R - - - -
Pattern : Absence (p)	Pattern : Existence (p)
Scope : Before R	Scope : Between L and R
Formula: $\diamond R \rightarrow \neg(\neg(R) U P)$	Formula: $[](L \wedge(\neg R) \rightarrow (\neg(R) W (P \wedge \neg R)))$
P : (i == 5)	P : (i == 9)
R : (i == 5)	R : (i == 16)
	L : (i == 5) $\vee(i == 12)$
Interval: Yes	Interval: Yes
Expected Result: No violation	Expected Result: No violation
Actual Result: Violation	Actual Result: Violation

Fig. 3. Examples of Prospec and SPS Failed Tests

4.3 Discussion

It is critical to provide assurance concerning the correctness of formulas that are generated automatically from tools. Verifying complex formulas is difficult even when experts are involved, and spin is effective at solving this problem. By defining a systematic approach for creating test cases and using a software model checker, it is possible to provide the user with an added layer of confidence that the formulas generated from SPS and Prospec are correct. While the problems found in Prospec were related to an error in the formula, those in SPS are related to the ambiguity of the language in the documentation. An unexpected result from using spin came from analysis of the difference in formulas generated by SPS and Prospec for $After\ Q$ scopes for the *universality*, *absence* and *response* patterns. Although the tests are

able to show that the formulas generated expected answers, the formulas add additional overhead by unnecessarily checking beyond the first occurrence of Q. This can create a performance issue as unneccesary check for additional Q's would result in delays in the model checker's executions.

References

1. Dillon, L., G. Kutty, L. E. Moser, P. M. Melliar – Smith, and Y.S. Ramakrishna, *A Graphical Interval Logic for Specifying Concurrent Systems*, ACM Transactions on Software Eng. and Methodology, **3** (1994), 131–165.
2. Dwyer, M. B., G. S. Avrunin, and J. C. Corbett, "Property Specification Patterns for Finite–State Verification," 2nd Workshop on Formal Methods in Software Practice. Clearwater Beach, Florida, 1998, 7–15.
3. Dwyer, M. B., G. S. Avrunin, and J. C. Corbett, "Patterns in Property Specification for Finite–State Verification," 21st Intl. Conference on Software Engineering, Los Angeles, CA, USA, 1999, 411–420.
4. Gamma, E. and R. Helm, "Design Patterns, Elements of Reusable Object–Oriented Software," Addison–Wesley, 1995, 416.
5. Kutty, G., "A Graphical Environment for Temporal Reasoning," Dissertation, Electrical and Computer Engineering Department, University of California at Santa Barbara, 1994.
6. Laroussinie, F. and P. Schnoebelen, "Specification in CTL+Past for Verification in CTL," *Information and Computation,* 2000, 236–263.
7. Manna, Z. and A. Pnueli, "Completing the Temporal Picture," *Theoretical Computer Science,* 83(1), 1991, 97-130.
8. Merriam Webster Online , http://www.m-w.com/cgi-bin/dictionary?book=Dictionaryva=follow, April 2005.
9. Mondragon, O., A. Gates, and S. Roach, "Composite Propositions: Toward Support for Formal Specification of System Properties," *Proceedings of the 27th Annual IEEE/NASA Goddard Software Engineering Workshop,* Greenbelt, MD, USA, December 2002.
10. Mondragon, O., A. Q. Gates, and S. Roach, "Prospec: Support for Elicitation and Formal Specification of Software Properties," in O. Sokolsky and M. Viswanathan (Eds.): *Proc. Runtime Verification Workshop, ENTCS*, 89(2), 2004.
11. Mondragon, O. and A. Q. Gates, "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," *Intl. Journal Software Engineering and Knowledge Engineering,*XS 14(1), Feb. 2004.
12. Mondragon, O., "Elucidation and Specification of Software Properties through Patterns and Composite Propositions to Support Formal Verification Techniques," Dissertation, The University of Texas at El Paso, May 2004.
13. Salamah, S., "Supporting Documentation for the SPS-Prospec Case Study," UTEP-CS-05-14, the University of Texas at El Paso, April 2005.
14. Spec Patterns, http://patterns.projects.cis.ksu.edu/, April 2005.
15. Spec Patterns, http://patterns.projects.cis.ksu.edu/documentation/patterns/scopes.shtml, April 2005.
16. Spec Patterns, http://patterns.projects.cis.ksu.edu/documentation/patterns/response.shtml, April 2005.

5 Appendix A. EQUIVALENCE CLASSES

A Complete list of all the equivalence classes for all patterns and scopes used in the case study can be found in Salamah [13].

Table 4. EQUIVALENCE CLASSES FOR ABSENCE-BEFORE R

EQUIVALENCE CLASSES ON P	TEST CASES WITH BOUNDARY ANALYSIS AND EQUIVALENCE CLASSES ON R	EXPECTED RESULTS
P Does not hold in any state in the interval	1. R holds in first state:	
	a) R -	1a. Valid
	b) (RP) -	1b. Valid
	c) R P - - - - - - - - - - - - - - - - - - -	1c. Valid
	2. R holds in last state:	
	a) - R	2a. Valid
	b) - (RP)	2b. Valid
	3. R holds in other states:	
	a) - - - - - - - - R - - - - - - - - - - - -	3a. Valid
	b) - - - - - - - - - (RP) - - - - - - - - - - -	3b. Valid
	c) - - - R - - - - - - - P R - - - - - - - -	3c. Valid
	4. Interval is not built	
	a) -	4a. Valid
	b) - - - - - - - P - - - - - - P - - - - - -	4b. Valid
P holds in some state of the interval	5. R holds in last state:	
	- - - - - - - - - - - - - - - - - - - P R	5. Not valid
	6. R holds in other states:	
	a) - - - - - - - - - P R - - - - - - - - - -	6a. Not valid
	b) - - - - - - - - - - R P - - - - - - - - -	6b. Valid

Table 5. EQUIVALENCE CLASSES for PRESEDENCE BETWEEN L AND R

EQUIVALENCE CLASSES ON P	TEST CASES WITH BOUNDARY ANALYSIS AND EQUIVALENCE CLASSES ON R	EXPECTED RESULTS
T Does not precede P in any state in the interval	1. The interval is not made:	
	a) - - - - - - - - - P - - - - - - - - - - R	1a. Valid
	b) - - - - (LRP) - - - - - - - - - - - - - - - -	1b. Valid
	c) - - - - L - - - - - - - P - - - - - - - -	1c. Valid
	d) - - - - - - - - - - - - P - - - - - - - -	1d. Valid
	e) R - - - - - - - P - - - - L - - - - - - -	1e. Valid
	2. A single interval is made:	
	a) L holds in first state:	
	i. L - - - P - - - - R - - - - - - - - - - -	2ai. Not valid
	ii. L - - - T - - - - R - - - - - - - - - - -	2aii. Valid
	iii. L - - P - - T - - R - - - - - - - - - - -	2aiii. Not valid
	iv. L - - - - (RP) - - - - - - - - - - - - - - -	2aiv. Valid
	v. L - - - - - R P - - - - - - - - - - - - -	2av. Valid
	vi. L - - - (PT) - - - - R - - - - - - - - - - -	2avi. Not valid
	b) R holds in last state:	
	i. - - - - - - - - - - - P L - - - - - - - R	2bi. Valid
	ii. - - - - - - - - - - - - - - - - L - - - (RP)	2bii. Valid
	iii. - - - - - - - - - - - L - - P - - - - - R	2biii. Not valid
	iv. - - - - - - - - - - - L - - P - - T - - R	2biv. Not valid
	v. - - - - - - - - - - - L - - T - - - - - R	2bv. Valid
	3. Multiple intervals are made:	
	a) L - - - - R - - - - - L - - - P - - R - -	3a. Not valid
	b) L - - - - R - - P - - - L - - - - R - - -	3b. Valid
	4. Nested intervals are made:	
	a) - - - - L - - - - - - L - - P - - R - - -	4a. Not valid
	b) - - - - L - - P - - - L - - - - - R - - -	4b. Not valid
	c) - - - L - - - L - - - R - - P - - R - - -	4c. Valid
	d) - - L - - T - - L - - - P - - - - R - - -	4d. Valid (in SPS) Not valid (in Prospec)
T precedes P in the interval	5. A single interval is made:	
	a) L holds in first state:	
	i. L T - P - - - - - R - - - - - - - - - - -	5ai. Valid
	ii. (LT) P - - - - R - - - - - - - - - - - - - -	5aii. Valid
	iii. L - - - T - P R - - - - - - - - - - - - -	5aiii. Valid
	b) R holds in last state:	
	i. - - - - - - - - - - - - - - L T P - - - R	5bi. Valid
	ii. - - - - - - - - - - - - - - (TL) P - - - - R	5bii. Valid
	iii. - - - - - - - - - - - - - - L - - T - P R	5biii.Valid
	c) L-R hold in other states:	
	i. - - - - - - - - L - - T - - P - - R - - -	5ci. Valid
	ii. - - - - - - - - - L - - T - - P - R - - -	5cii. Valid
	6. Multiple intervals are made:	
	a. L - T - P - - R - - - - L - - R - - L P R	6a. Not valid
	b. L - T P - - R - L - - T - P - R - L - P -	6b. Valid
	7. Nested intervals are made:	
	- - - L - - T - - L - - - P - R - - - - -	7. Valid in SPS Not valid in Prospec

6 Appendix B. Prospec's Screen-Shots

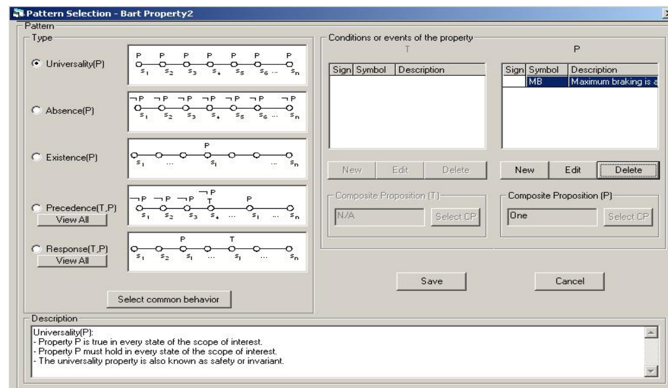

Fig. 4. Prospec's Pattern Screen.

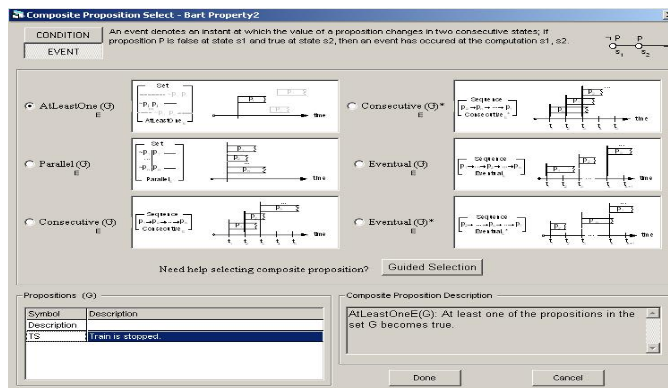

Fig. 5. Prospec's Composite Proposition Screen

7 Appendix C. SPS and Prospec's Sample Failed Tests

A Complete list of all the test cases used in the case study can be found in Salamah [13].

Prospec Failed Test
Test 6 : - - - - - (RP) - - - - - - - - - - - - - - - - Pattern : Absence (p) Scope : Before R Formula: $\diamond R \rightarrow (\neg(\neg(R)UP))$ P : (i == 5) R : (i == 5) Interval: Yes Expected Result: No violation Actual Result: Violation
Test 26 : - - - - - L - - - - - - - - - - - (RP) - - - - Pattern : Existence (p) Scope : Between L and R Formula: []((L $\wedge(\neg R)$ $\wedge$ $(\diamond R))$ $\rightarrow$ $(\neg(\neg(R)UP))$ P : (i == 16) R : (i == 16) L : (i == 5) Interval: Yes Expected Result: No violation Actual Result: Violation
Test 46 : - - - - - L - - - - - - - - - - - (RP) - - - - Pattern : Existence (p) Scope : After L Until R Formula: [](L $\wedge(\neg R) \rightarrow (\neg(\neg(R)UP))$ P : (i == 16) R : (i == 16) L : (i == 5) $\vee(i == 12)$ Interval: Yes Expected Result: No violation Actual Result: Violation

Fig. 6. Examples of Prospec Failed Tests

Test 30: - - L - - - P - - - L - - - - - - R - - - Pattern : Existence (p) Scope : Between L and R Formula: $[]((L \wedge (\neg R)) \rightarrow (\neg R W (P \wedge \neg R)))$ P : (i == 6) L : (i == 2) $\vee (i == 10)$ R : (i == 17) Interval: Yes Expected Result: No violation Actual Result: Violation	Test 42 : - - L - - - P - - - L - - - - - - R - - - Pattern : Existence (p) Scope : After L Until R Formula: $[]((L \wedge (\neg R)) \rightarrow (\neg R U (P \wedge \neg R)))$ P : (i == 6) L : (i == 2) $\vee (i == 10)$ R : (i == 17) Interval: Yes Expected Result: No violation Actual Result: Violation
Test 38 : - - L - - - T - - - L - - P - - - R - - - Pattern : (T) Precedes (p) Scope : Between L and R Formula: $[]((L \wedge (\neg R) \wedge \diamond R) \rightarrow (\neg P U (T \vee R)))$ P : (i == 13) L : (i == 2) $\vee (i == 10)$ R : (i == 17) T : (i == 6) Interval: Yes Expected Result: No violation Actual Result: Violation	Test 41 : - - L - - - T - - - L - - P - - - R - - - Pattern : (T) Precedes (p) Scope : After L until R Formula: $[]((L \wedge (\neg R)) \rightarrow (\neg P W (T \vee R)))$ P : (i == 13) L : (i == 2) $\vee (i == 10)$ R : (i == 5) T : (i == 6) Interval: Yes Expected Result: No violation Actual Result: Violation
Test 3 : - - - - - (TP) - - - - - - - - - - - - - - - Pattern : T Responds to P Scope : Global Formula: $[](P \rightarrow \diamond T)$ P : (i == 5) T : (i == 5) Interval: Yes Expected Result: Violation Actual Result: No violation	Test 9 : - - - - - (TP) - - R - - - - - - - - - - - - Pattern : T Responds to P Scope : Before R Formula: $\diamond R \rightarrow (P \rightarrow (\neg R U (T \wedge \neg R))) U R$ P : (i == 5) T : (i == 5) R : (i == 8) Interval: Yes Expected Result: Violation Actual Result: No violation
Test 20 : - - L - - (RP) - - - - - - - - - - - - - - - Pattern : T Responds to P Scope : After L Formula: $[](L \rightarrow [](P \rightarrow \diamond T))$ P : (i == 5) T : (i == 5) L : (i == 2) Interval: Yes Expected Result: Violation Actual Result: No violation	Test 36 : - - L - - (RP) - - - R - - - - - - - - - - - Pattern : T Responds to P Scope : Between L and R Formula: $[]((L \wedge (\neg R) \wedge \diamond R) \rightarrow (P \rightarrow (\neg R U (T \wedge \neg R))) U R)$ P : (i == 5) L : (i == 2) T : (i == 5) R : (i == 9) Interval: Yes Expected Result: Violation Actual Result: No violation
Test 43 : - - L - - (RP) - - - R - - - - - - - - - - - Pattern : T Responds to P Scope : After L Until R Formula: $[]((L \wedge (\neg R)) \rightarrow (P \rightarrow (\neg R U (T \wedge \neg R))) W R)$ P : (i == 5) L : (i == 2) T : (i == 5) R : (i == 9) Interval: Yes Expected Result: Violation Actual Result: No violation	Test 44 : - - L - - (RP) - - - - - - - - - - - - - R Pattern : T Responds to P Scope : After L Until R Formula: $[]((L \wedge (\neg R)) \rightarrow (P \rightarrow (\neg R U (T \wedge \neg R))) W R)$ P : (i == 5) L : (i == 2) T : (i == 5) R : (i == limit) Interval: Yes Expected Result: Violation Actual Result: No violation

Fig. 7. Examples of SPS Failed Tests