Using Patterns and Composite Propositions to Automate the Generation of Complex LTL

Salamah Salamah

Ann Q. Gates
The University of Texas at El Paso, agates@utep.edu

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Steve Roach
The University of Texas at El Paso, sroach@utep.edu

Comments:
Technical Report: UTEP-CS-07-15b
Published in: K. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura (Eds.), *Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis ATVA'2007*, Tokyo, Japan, October 22-25, 2007, Springer Lecture Notes in Computer Science, 2007, Vol. 4762, pp. 533-542.

Using Patterns and Composite Propositions to Automate the Generation of LTL Specifications

Salamah Salamah, Ann Q. Gates, Vladik Kreinovich, and Steve Roach
Dept. of Computer Science, University of Texas at El Paso
El Paso, TX 79968, USA

Abstract

Property classifications and patterns, i.e., high-level abstractions that describe common behavior, have been used to assist practitioners in generating formal specifications that can be used in formal verification techniques. The Specification Pattern System (SPS) provides descriptions of a collection of patterns. Each pattern is associated with a scope that defines the extent of program execution over which a property pattern is considered. Based on a selected pattern, SPS provides a specification for each type of scope in multiple formal languages including Linear Temporal Logic (LTL). The (Prospec) tool extends SPS by introducing the notion of Composite Propositions (CP), which are classifications for defining sequential and concurrent behavior to represent pattern and scope parameters.

In this work, we provide definitions of patterns and scopes when defined using CP classes. In addition, we provide general (template) LTL formulas that can be used to generate LTL specifications for all combinations of pattern, scope, and CP classes.

1 Introduction

Although the use of formal verification techniques such as model checking [4] and runtime monitoring [8] improve the dependability of programs, they are not widely adapted in standard software development practices. One reason for the hesitance in using formal verification is the high level of mathematical sophistication required for reading and writing the formal specifications required for the use of these techniques [3].

Different approaches and tools such as the Specification Pattern System (SPS) [2] and the Property Specification Tool (Prospec) [5] have been designed to provide assistance to practitioners in generating formal specifications. Such tools and approaches support the generation of formal specifications in multiple formalizations. The notions of patterns, scopes, and composite propositions (CP) have been identified as ways to assist users in defining formal properties. Patterns capture the expertise of developers by describing solutions to recurrent problems. Scopes on the other hand, allow the user to define the portion of execution where a pattern is to hold.

The aforementioned tools take the user's specifications and provide formal specifications that matches the selected pattern and scope in multiple formalizations. SPS for example provides specifications in Linear Temporal Logic (LTL) and computational Tree Logic (CTL) among others. On the other hand, Prospec provides specifications in Future Interval Logic (FIL) and Meta-Event Definition Language (MEDL). These tools however, do not support the generation of specifications that use CP in LTL. The importance of LTL stems from its expressive power and the fact that it is widely used in multiple formal verification tools. This work provides a set of template LTL formulas that can be used to specify a wide range of properties in LTL.

The paper is organized as follows: Section 2 provides the background related information including description of LTL and the work that has been done to support the generation of formal specifications. Section 3 highlights the problems of generating formal specifications in LTL. Sections 4 and 5 provide the general formal definitions of patterns and scopes that use CP. Section 6 motivates the need for three new LTL operators to simplify the specifications of complex LTL formulas. Last, the general LTL template formulas for the different scopes are described followed by summary and future work.

2 Background

2.1 Linear Temporal Logic

Linear Temporal Logic (LTL) is a prominent formal specification language that is highly expressive and widely used in formal verification tools such as the model checkers SPIN [4] and NUSMV [1]. LTL is also used in the runtime verification of Java programs [8].

Formulas in LTL are constructed from elementary propositions and the usual Boolean operators for *not*, *and*, *or*, *imply* (*neg*, $\wedge$, $\vee$, $\rightarrow$, respectively). In addition, LTL allows for the use of the temporal operators *next* (X), *eventually* ($\diamond$), *always* ($\Box$), *until*, (U), *weak until* (W), and *release* (R). In this work, we only use the first four of these operators. These formulas assume discrete time, i.e., states $s = 0, 1, 2, \ldots$ The meaning of the temporal operators is straightforward. The formula XP holds at state s if P holds at the next state $s + 1$. $P\,U\,Q$ is true at state s, if there is a state $s' \geq s$ at which Q is true and, if s' is such a state, then P is true at all states s_i for which $s \leq s_i < s'$. The formula $\diamond P$ is true at state s if P is true at some state $s' \geq s$. Finally, the formula $\Box P$ holds at state s if P is true at all moments of time $s' \geq s$. Detailed description of LTL is provided by Manna et al. [6].

2.2 Specification Pattern System (SPS)

Writing formal specification, particularly those involving time, is difficult. The Specification Pattern System [2] provides patterns and scopes to assist the practitioner in formally specifying software properties. These patterns and scopes were defined after analyzing a wide range of properties from multiple industrial domains (i.e., security protocols, application software, and hardware systems). *Patterns* capture the expertise of developers by describing solutions to recurrent problems. Each pattern describes the structure of specific behavior and defines the pattern's relationship with other patterns. Patterns are associated with scopes that define the portion of program execution over which the property holds.

The main patterns defined by SPS are: $Universality$, *Absence*, *Existence*, *Precedence*, and *Response*. 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: *Global*, $Before\ R$, $After\ L$, $Between\ L\ And\ R$, and $After\ L\ Until\ R$. A detailed description of these patterns and scopes can be found in Dewyer [2].

2.3 Composite Propositions (CP)

The idea of CP was introduced by Mondragon et al. [5] to allow for patterns and scopes to be defined using multiple propositions. In practical applications, we often need to describe properties where one or more of the pattern or scope parameters are made of multiple propositions, i.e., composite propositions (CP). For example, the property that every time data is sent at state s_i the data is read at state $s_1 \geq s_i$, the data is processed at state s_2, and data is stored at state s_3, can be described using the $Existence(P)$ pattern within the *Between* L and R scope. In this example L stands for "data is sent", R stands for 'date is stored' and P is composed of p_1 and p_2 (data is read and data is processed, respectively).

To describe such patterns, Mondragon et al. [5] extended SPS by introducing a classification for defining sequential and concurrent behavior to describe pattern and scope parameters. Specifically, the work formally described several types of CP classes and provided formal descriptions of these CP classes in LTL. Mondragon et al defined eight CP classes and described their semantics using LTL. The eight CP classes defined by that work are $AtLeastOne_C$, $AtLeastOne_E$, $Parallel_C$, $Parallel_E$, $Consecutive_C$, $Consecutive_E$, $Eventual_C$, and $Eventual_E$. The subscripts C and E describe whether the propositions in the CP class are asserted as Conditions or Events respectively. A proposition defined as a condition holds in one or more consecutive states. A proposition defined as event means that there is an instant at which the proposition changes value in two consecutive states.

This work modified the LTL description of the CP classes $AtLeastOne_E$, $Eventual_C$, and $Eventual_E$. The work changed the the semantics of the $AtLeastOne_E$ class to one that is more consistent with the other CP classes

Table 1: Description of CP Classes in LTL

CP Class	**LTL Description** (P^{LTL})
$AtLeastOne_C$	$p_1 \vee \ldots \vee p_n$
$AtLeastOne_E$	$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \vee \ldots \vee p_n))$
$Parallel_C$	$p_1 \wedge \ldots \wedge p_n$
$Parallel_E$	$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \wedge \ldots \wedge p_n))$
$Consecutive_C$	$(p_1 \wedge X(p_2 \wedge (\ldots (\wedge X p_n)) \ldots))$
$Consecutive_E$	$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \wedge X(p_2 \wedge \neg p_3 \wedge \ldots \wedge \neg p_n \wedge X(\ldots \wedge X(p_{n-1} \wedge \neg p_n \wedge X p_n)) \ldots))$
$Eventual_C$	$(p_1 \wedge X(\neg p_2\, U\, (p_2 \wedge X(\ldots \wedge X(\neg p_{n-1}\, U\, (p_{n-1} \wedge X(\neg p_n\, U\, p_n)))) \ldots))))$
$Eventual_E$	$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \wedge ((\neg p_2 \wedge \ldots \wedge \neg p_n)\, U\, (p_2 \wedge \neg p_3 \wedge \ldots \wedge \neg p_n \wedge (\ldots \wedge (p_{n-1} \wedge \neg p_n \wedge (\neg p_n\, U\, p_n)) \ldots)))))$

of type E. The LTL description of the other two CP classes were modified to a semantically equivalent LTL formulas. Table 1. provides the semantics of the CP classes used in this paper in LTL.

3 Problem With Direct Substitution

Although SPS provides LTL formulas for basic patterns and scopes (ones that use single, "atomic", propositions to define L, R, P, and Q) and Mondragon et al. provided LTL semantics for the CP classes as described in Table 1., in most cases it is not adequate to simply substitute the LTL description of the CP class into the basic LTL formula for the pattern and scope combination. Consider the following property: "The delete button is enabled in the main window only if the user is logged in as administrator and the main window is invoked by selecting it from the Admin menu.". This property can be described using the $Existence(Eventual_C(p_1, p_2))\ Before(r)$ where p_1 is "the user logged in as an admin", p_2 is "the main window is invoked", and r is "the delete button is enabled". As mentioned above, the LTL formula for the $Existence(P)\, Before(R)$ is "$(\Box \neg R) \vee (\neg R\, U\, (P \wedge \neg R))$", and the LTL formula for the CP class $Eventual_C$, as described in Table 1, is $(p_1 \wedge X(\neg p_2\, U\, p_2))$. By replacing P by $(p_1 \wedge X(\neg p_2\, U\, p_2))$ in the formula for the pattern and scope, we get the formula: "$(\Box \neg R) \vee (\neg R\, U\, ((p_1 \wedge X(\neg p_2\, U\, p_2)) \wedge \neg R))$" This formula however, asserts that either R never holds or R holds after the formula $(p_1 \wedge X(\neg p_2\, U\, p_2))$ becomes true. In other words, the formula asserts that it is an acceptable behavior if R ("the delete button is enabled") holds after p_1 ("the user logged in as an admin") holds and before p_2 ("the main window is invoked") holds, which should not be an acceptable behavior.

As seen by the above example, the temporal nature of LTL and its operators means that direct substitution could lead to the description of behaviors that do not match the actual intent of the specifier. For this reason, it is necessary to provide abstract LTL formulas that can be used as templates for the generation of LTL specifications for all patterns, scopes, and CP classes combinations, which is the goal of this paper.

4 Patterns Defined With Composite Propositions

As we mentioned in Section 2.2, Dwyer et al. defined the notions of patterns and scopes to assist in the definition of formal specifications. Patterns provide common solutions to recurring problems, and scopes define the extent of program execution where the pattern is evaluated. In this work we are concerned with the following patterns: the absence of P, the existence of P, Q precedes P, Q strictly precedes P, and Q responds to P.

Note that the strict precedence pattern was defined by Mondragon et al. [5], and it represents a modification of the precedence pattern as defined by Dwyer et al. The following subsections describe these patterns when defined using single and composite propositions.

The absence of P means that the (single or composite) property P never holds, i.e., for every state s, P does not hold at s. In the case of CP classes, this simply means that P^{LTL} (as defined in Table 1 for each CP class) is never true. The LTL formula corresponding to the absence of P is:

$$\Box \neg P^{LTL}$$

The existence of P means that the (single or composite) property P holds at some state s in the computation. In the case of CP classes, this simply means that P^{LTL} is true at some state of the computation. The LTL formula corresponding to the existence of P is:

$$\Diamond P^{LTL}$$

For single proposition, the meaning of "precedes", "strictly precedes", and "responds" is straightforward. To extend the meanings of these patterns to ones defined using CP, we need to explain what "after" and "before" mean for the case of CP. While single propositions are evaluated in a single state, CP, in general, deal with a sequence of states or a time interval (this time interval may be degenerate, i.e., it may consist of a single state). Specifically, for every CP $P = T(p_1, \ldots, p_n)$, there is a beginning state b_P – the first state in which one of the propositions p_i becomes true, and an ending state e_P – the first state in which the condition T is fulfilled. For example, for $Consecutive_C$, the ending state is the state $s+(n-1)$ when the last statement p_n holds; for $AtLeastOne_C$, the ending state is the same as the beginning state – it is the first state when one of the propositions p_i holds for the first time.

For each state s and for each CP $P = T(p_1 \ldots, p_n)$ that holds at this state s, we will define the beginning state $b_P(s)$ and the ending state $e_P(s)$. The following is a description of b_P and e_P for the CP classes of types condition and event defined in Table 1 (to simplify notations, wherever it does not cause confusion, we will skip the state s and simply write b_P and e_P):

- For the CP class $P = AtLeastOne_C(p_1, \ldots, p_n)$ that holds at state s, we take $b_P(s) = e_P(s) = s$.

- For the CP class $P = AtLeastOne_E(p_1, \dots, p_n)$ that holds at state s, we take, as $e_P(s)$, the first state $s' > s$ at which one of the propositions p_i becomes true and we take $b_P(s) = (e_P(s) - 1)$.
- For the CP class $P = Parallel_C(p_1, \dots, p_n)$ that holds at state s, we take $b_P(s) = e_P(s) = s$.
- For the CP class $P = Parallel_E(p_1, \dots, p_n)$ that holds at state s, we take, as $e_P(s)$, the first state $s' > s$ at which all the propositions p_i become true and we take $b_P(s) = (e_P(s) - 1)$.
- For the CP class $P = Consecutive_C(p_1, \dots, p_n)$ that holds at state s, we take $b_P(s) = s$ and $e_P(s) = s + (n - 1)$.
- For the CP class $P = Consecutive_E(p_1, \dots, p_n)$ that holds at state s, we take, as $b_P(s)$, the last state $s' > s$ at which all the propositions were false and in the next state the proposition p_1 becomes true, and we take $e_P(s) = s' + (n)$.
- For the CP class $P = Eventual_C(p_1, \dots, p_n)$ that holds at state s, we take $b_P(s) = s$, and as $e_P(s)$, we take the first state $s_n > s$ in which the last proposition p_n is true and the previous propositions $p_2, \dots, p_{n-1}$ were true at the corresponding states $s_2, \dots, s_{n-1}$ for which $s < s_2 < \dots < s_{n-1} < s_n$.
- For the CP class $P = Eventual_E(p_1, \dots, p_n)$ that holds at state s, we take as $b_P(s)$, the last state state s_1 at which all the propositions were false and in the next state the first proposition p_1 becomes true, and as $e_P(s)$, the first state s_n in which the last proposition p_n becomes true.

Using the notions of beginning and ending states, we can give a precise definitions of the Precedence, Strict Precedence, and Response patterns with Global scope:

Definition 1 *Let P and Q be CP classes. We say that Q* precedes *P if once P holds at some state s, then Q also holds at some state s' for which $e_Q(s') \leq b_P(s)$. This simply indicates that Q precedes P iff the ending state of Q is the same as the beginning state of P or it is a state that happens before the beginning state of P.*

Definition 2 *Let P and Q be CP classes. We say that Q* strictly precedes *P if once P holds at some state s, then Q also holds at some state s' for which $e_Q(s') < b_P(s)$. This simply indicates that Q strictly precedes P iff the ending state of Q is a state that happens before the beginning state of P.*

Definition 3 *Let P and Q be CP classes. We say that Q* responds to *P if once P holds at some state s, then Q also holds at some state s' for which $b_Q(s') \geq e_P(s)$. This simply indicates that Q responds to P iff the beginning state of Q is the same as the ending state of P or it is a state that follows the ending state of P.*

5 Non-Global Scopes Defined With Composite Propositions

So far we have discussed patterns within the "Global" scope. In this Section, we provide a formal definition of the other scopes described in Section 2.2.

We start by providing formal definitions of scopes that use CP as their parameters.[1]

- For the "Before R" scope, there is exactly one scope – the interval $[0, b_R(s_f))$, where s_f is the first state when R becomes true. Note that the scope contains the state where the computation starts, but it does not contain the state associated with $b_R(s_f)$.
- For the scope "After L", there is exactly one scope – the interval $[e_L(s_f), \infty)$, where s_f is the first state in which L becomes true. This scope, includes the state associated with $e_L(s_f)$.
- For the scope "Between L and R", a scope is an interval $[e_L(s_L), b_R(s_R))$, where s_L is the state in which L holds and s_R is the first state $> e_L(s_L)$ when R becomes true. The interval contains the state associated with $e_L(s_L)$ but not the state associated with $b_R(s_R)$.
- For the scope "After L Until R", in addition to scopes corresponding to "Between L and R", we also allow a scope $[e_L(s_L), \infty)$, where s_L is the state in which L holds and for which R does not hold at state $s > e_L(s_L)$.

Using the above definitions of scopes made up of CP, we can now define what it means for a CP class to hold within a scope.

Definition 4 *Let P be a CP class, and let S be a scope. We say that P s-holds (meaning, P holds in the scope S) in S if P^{LTL} holds at state $s_p \in S$ and $e_P(s_P) \in S$ (i.e. ending state $e_P(s_p)$ belongs to the same scope S).*

Table 3 provides a formal description of what it means for a pattern to hold within a scope.

6 Need for New Operations

To describe LTL formulas for the patterns and scopes with CP, we need to define new "and" operations. These operations will be used to simplify the specification of the LTL formulas in Section 7.

In non-temporal logic, the formula $A \wedge B$ simply means that both A and B are true. In particular, if we consider a non-temporal formula A as a particular case of LTL formulas, then A means simply that the statement A holds at the

[1] These definitions use the notion of beginning state and ending state as defined in Section 4.

Table 2: Description of Patterns Within Scopes

Pattern	**Description**
Existence	We say that there is an *existence of P within a scope S* if P s-holds at some state within this scope.
Absence	We say that there is an *absence of P within a scope S* if P never s-holds at any state within this scope.
Precedence	We say that Q *precedes P within the scope s* if once P s-holds at some state s, then Q also s-holds at some state s' for which $e_Q(s') \leq b_P(s)$.
Strict Precedence	We say that Q *strictly precedes P within the scope s* if once P s-holds at some state s, then Q also s-holds at some state s' for which $e_Q(s') < b_P(s)$.
Response	We say that Q *responds to P within the scope s* if once P s-holds at some state s, then Q also s-holds at some state s' for which $b_Q(s') \geq e_P(s)$.

given state, and the formula $A \wedge B$ means that both A and B hold at this same state.

In general a LTL formula A holds at state s if some "subformulas" of A hold in s and other subformulas hold in other states. For example, the formula $p_1 \wedge Xp_2$ means that p_1 holds at the state s while p_2 holds at the state $s+1$; the formula $p_1 \wedge X \diamond p_2$ means that p_1 holds at state s and p_2 holds at some future state $s_2 > s$, etc. The statement $A \wedge B$ means that different subformulas of A hold at the corresponding different states but B only holds at the original state s. For patterns involving CP, we define an "and" operation that ensures that B holds at all states in which different subformulas of A hold. For example, for this new "and" operation, $(p_1 \wedge Xp_2)$ and B would mean that B holds both at the state s and at the state $s+1$ (i.e. the correct formula is $(p_1 \wedge B \wedge X(p_2 \wedge B)))$. Similarly, $(p_1 \wedge X \diamond p_2)$ and B should mean that B holds both at state s and at state $s_2 > s$ when p_2 holds. In other words, we want to state that at the original state s, we must have $p_1 \wedge B$, and that at some future state $s_2 > s$, we must have $p_2 \wedge B$. This can be described as $(p_1 \wedge B) \wedge X \diamond (p_2 \wedge B)$.

To distinguish this new "and" operation from the original LTL operation $\wedge$, we will use a different "and" symbol & to describe this new operation. However, this symbol by itself is not sufficient since people use & in LTL as well; so, to emphasize that our "and" operation means "and" applied at several different moments of time, we will use a combination $\&_r$ of several & symbols.

In addition to the $A \,\&_r, B$ operator, we will also need two more operations:

- The new operation $A \,\&_l\, B$ will indicate that B holds at the *last* of A-relevant moments of time.

- The new operation $A \,\&_{-l}\, B$ will indicate that B holds at the all A-relevant moments of time except for the last one.

For the lack of space, this paper does not include the detailed description of these new LTL operators. The formal descriptions of these LTL operators along with examples of their use is provided by Salamah [7].

7 General LTL Formulas for Patterns and Scopes With CP

Using the above mentioned new LTL operators, this work defined template LTL formulas that can be used to define LTL specifications for all pattern/scope/CP combinations. The work defined three groups of templates; templates to generates formulas for the *Global* scope, templates to generate formulas for the $BeforeR$ scope, and templates to generate formulas for the remaining scopes. The templates for these remaining scopes use the templates for the *Global* and $BeforeR$ scopes. For the lack of space, we show an example template LTL formula from each of these three groups. The remaining templates are available in Salamah [7].

An example of a template LTL formula within the *Global* scope, is the template LTL formula for Q *Responds* to P:

- $\Box(P^{LTL} \rightarrow (P^{LTL} \,\&_l\, \Diamond Q^{LTL}))$

An example of a template LTL formula within the $Before\ R$ scope, is the template LTL formula for $Q\,Precedes\ P_C\,Before\,R_C$:

- $(\Diamond R^{LTL}) \rightarrow ((\neg(P^{LTL}\&_r\neg R^{LTL}))\,U\,((Q^{LTL}\&_{-l}\neg P^{LTL}) \vee R^{LTL}))$

Finally, template formulas for the three remaining scopes can be constructed based on the templates for the *Global* and $BeforeR$ scopes. The formulas for the $AfterL$ scope can be built using the formulas for the *Global* scope as follows:

- $\neg((\neg L^{LTL})\,U\,(L^{LTL}\&_l\neg\mathcal{P}_G^{LTL}))$

This means that for any pattern, the formula for this pattern within the $ASfterL$ scope can be generated using the above formula and simply substituting the term $\mathcal{P}_G^{LTL}$ by the formula for that pattern within the *Global* scope.

In these examples and the remaining templates, the subscripts C and E attached to each CP indicates whether the CP class is of type condition or event, respectively. In the case where no subscript is provided, then this indicates that the type of the CP class is irrelevant and that the formula works for both types of CP classes.

8 Summary and Future Work

The work in this paper provided formal descriptions of the different composite propositions (CP) classes defined by Mondragon et al. [5]. In addition, we formally described the patterns and scopes defined by Dweyer et al. [2] when using CP classes. The main contribution of the paper is defining general LTL formulas that can be used to generate LTL specifications of properties defined by patterns, scopes, and CP classes. The general LTL formulas for the Global scope have been verified using formal proofs [7]. On the other hand, formulas for the remaining scopes were verified using testing and formal reviews [7].

The next step in this work consists of providing formal proofs for formulas of the remaining scopes. In addition, we aim at enhancing the Prospec tool by including the generation of LTL formulas that use the translations provided by this paper.

Acknowledgments. This work is funded in part by the National Science Foundation (NSF) through grant NSF EAR-0225670, by Texas Department of Transportation grant No. 0-5453, and by the Japan Advanced Institute of Science and Technology (JAIST) International Joint Research Grant 2006-08.

References

[1] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri,M., "NUSMV: a new Symbolic Model Verifier", *International Conference on Computer Aided Verification CAV*, July 1999.

[2] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C.,"Patterns in Property Specification for Finite State Verification," *Proceedings of the 21st international conference on Software engineering*, Los Angeles, CA, 1999, 411–420.

[3] Hall, A., "Seven Myths of Formal Methods," IEEE Software, September 1990, 11(8)

[4] Holzmann G. J., *The SPIN Model Checker: Primer and Reference Manual,* Addison-Wesley Professional, 2004.

[5] 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.

[6] Manna, Z. and Pnueli, A., "Completing the Temporal Picture," *Theoretical Computer Science,* 83(1), 1991, 97–130.

[7] Salamah, I. S., *Defining LTL formulas for complex pattern-based software properties*, University of Texas at El Paso, Department of Computer Science, PhD Dissertation, July 2007.

[8] Stolz, V. and Bodden, E., "Temporal Assertions using AspectJ", *Fifth Workshop on Runtime Verification,* July 2005.