Verification of Automatically Generated Pattern-Based LTL Specifications

Salamah Salamah
Dept. Comp. and Software Engineering, Embry-Riddle Aeronautical Univ., salamahs@erau.edu

Ann Q. Gates, Vladik Kreinovich, Steve Roach
Computer Science Dept., University of Texas at El Paso, {agates,vladik,sroach}@utep.edu

Abstract

The use of property classifications and patterns, i.e., high-level abstractions that describe common behavior, have been shown 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. The extent of program execution over which a pattern must hold is described by the notion of scope. SPS provides a manual technique for obtaining formal specifications from a pattern and a scope. The Property Specification Tool (Prospec) extends SPS by introducing Composite Propositions (CPs), a classification for defining sequential and concurrent behavior to represent pattern and scope parameters, and provides a tool to support users.

This work provides general templates for generating formal specifications in Linear Temporal Logic (LTL) for all pattern, scope, and CP combinations. In addition, the work explains the methodology for the verification of the correctness of these templates.

1 Introduction

Although the use of formal verification techniques such as model checking [5], theorem proving [9], and runtime monitoring [13] 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 [4].

Different approaches and tools, such as the Specification Pattern System (SPS) [2], the Property Elucidation tool (Propel) [11], and the Property Specification Tool (Prospec) [7], have been designed to provide assistance to practitioners in generating formal specifications in a variety of formal languages.

The notions of patterns, scopes, and composite propositions (CPs) have been identified as ways to assist users in defining formal properties. Patterns are high-level abstractions used to describe program behavior. Scopes define the portion of execution over which a pattern is to hold. A formal specification is generated from a user's selection for pattern and scope. SPS, for example, generates specifications in Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) among others, while Propel provides formal specifications in the form of a finite automaton. Finally, Prospec provides specifications in Future Interval Logic (FIL) and Meta-Event Definition Language. None of these tools generate specifications using CPs in LTL. This work provides a set of LTL templates that can be used to specify a wide range of properties in LTL. The importance of LTL stems from its expressive power and the fact that it is widely used in formal verification tools. By adding this capability to a tool such as Prospec, one can specify a complex property without being an LTL expert.

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 challenges of generating formal specifications in LTL. Sections 4 and 5 provide the general formal definitions of patterns and scopes that use CPs. Section 6 motivates the need for three auxiliary LTL operators to simplify the specifications and provides the formal definition for one of these operators. Section 7 describes the general LTL templates for the different patterns and scopes defined using CPs. Section 8 provides a description of the verification techniques for the LTL templates, followed by summary and future work (in Section 9).

2 Background

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 checker SPIN [5] and NUSMV [1]. LTL is also used in the runtime verification of Java programs [13].

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 provides 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. [8].

Specification Pattern System (SPS). Writing formal specifications, 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 software behavior for recurrent situations. Each pattern describes the structure of specific behavior and defines the pattern's relationship with other patterns. Patterns are associated with *scopes*, which 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 Dwyer [2].

Composite Propositions (CPs). The notion of CPs was introduced by Mondragon et al. [7] 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., CPs (examples are given in the following sections).

The CPs introduced by Mondragon et al. [7] allow one to specify the sequential and concurrent behavior among multiple propositions given as pattern or scope parameters. Mondragon et al. defined the following eight CP classes: $AtLeastOne_C$, $AtLeastOne_E$, $Parallel_C$, $Parallel_E$, $Consecutive_C$, $Consecutive_E$, $Eventual_C$, and $Eventual_E$. The subscripts C and E describe whether the propositions in the CP class are asserted as *c*onditions or *e*vents 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. Table 1 provides the semantics of the CP classes in LTL. The last four entries in the table are the auxiliary CP classes of type $Hold$ that are used in the definitions of the LTL formulas given in Section 7.

Table 1. Description of CP Classes in LTL

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

3 Challenges in Generating LTL Formulas

SPS provides LTL formulas for basic patterns and scopes, i.e., ones that use single, atomic propositions in the parameters L, R, P, and Q of the patterns and scope. Mondragon et al. provided LTL semantics for the CP classes as described in Table 1; however, in at least some 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 $Existence(Eventual_C(p_1, p_2))$ pattern and the $Before(r)$ scope, 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."

According to Mondragon et al. [7], the LTL formula for the $Existence(P)$ pattern with $Before(R)$ 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 acceptable for R ("the delete button is enabled") to hold after p_1 ("the user logged in as an admin") holds and before p_2 ("the main window is invoked") holds. This should not be the case.

As seen by the above example, the temporal nature of LTL and its operators makes direct substitution problematic. To avoid the generation of formulas that do not meet the intent of the user as specified through patterns, scope, and CP, it is necessary to provide abstract LTL formulas that can be used as templates for the generation of LTL specifications for all combinations of patterns, scope, and CPs.

4. Patterns Defined with CPs

As we mentioned in Section 2, Dwyer's SPS system defined the notions of patterns and scopes to assist in the definition of formal specifications. 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. The strict precedence pattern was defined by Mondragon et al. [7], and it represents a modification of the precedence pattern as defined by Dwyer. 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 CPs, we need to explain what "after" and "before" mean for the case of CPs. While single propositions are evaluated in a single state, CPs, 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:

- 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 $P = AtLeastOne_E(p_1, \ldots, p_n)$ 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 $P = Parallel_C(p_1, \ldots, p_n)$ that holds at state s, we take $b_P(s) = e_P(s) = s$.
- For the CP class $P = Parallel_E(p_1, \ldots, p_n)$ 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 $P = Consecutive_C(p_1, \ldots, p_n)$ that holds at state s, we take $b_P(s) = s$ and $e_P(s) = s + (n-1)$.
- For the CP class $P = Consecutive_E(p_1, \ldots, p_n)$ 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 $P = Eventual_C(p_1, \ldots, p_n)$ 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 $p_2, \ldots, p_{n-1}$ were true at the corresponding states $s_2, \ldots, s_{n-1}$ for which $s < s_2 < \ldots < s_{n-1} < s_n$.
- For the CP class $P = Eventual_E(p_1, \ldots, p_n)$ 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.

We now 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 means that Q precedes P iff the ending state of Q is either identical to the beginning state of P or happens before this beginning state.

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 means 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 $b_Q(s') \geq e_P(s)$.*

5 Non-Global Scopes Defined with CPs

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.

We start by providing formal definitions of scopes that use CPs as their parameters.

- For $Before\ R$, 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 $After\ L$, 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 $Between\ L\ and\ R$, 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 $After\ L\ Until\ R$, 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 CPs, 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* holds in the scope *S (or S-*holds*, for short) if P^{LTL} holds at a state $s_P \in S$ for which ending state $e_P(s_p)$ belongs to the same scope S (i.e., $e_P(s_P) \in S$).*

Table 2 provides a formal description of what it means for a pattern to hold within a scope.

6 Auxiliary Operations

To describe LTL formulas for the patterns and scopes with CPs, it is useful to define a special "and" operation

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 $b_Q(s') \geq e_P(s)$.

that can be used as a shorthand to simplify the specification of the LTL formulas in Section 7. In propositional logic, the formula $A \wedge B$ means that both A and B are true. If we consider a propositional formula A as a particular case of LTL formulas, then A means simply that the statement A holds at the given state, and the formula $A \wedge B$ means that both A and B hold at this same state.

In general, it is necessary to state that an 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 $p_1 \wedge X p_2$ means that p_1 holds at the state s while p_2 holds at the state $s+1$, and 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 $s_2 > s$. For patterns involving CPs, 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,

$$(p_1 \wedge X p_2) \text{ and } B$$

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,

$$(p_1 \wedge X \diamond p_2) \text{ and } B$$

means that B holds both at state s and at state $s_2 > s$ when p_2 holds. At the original state s, we must have $p_1 \wedge B$, and 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 the symbol $\&_r$ to denote this new operation. In addition, we introduce two more operations:

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

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

This paper only includes the detailed description of "$\&_r$" operator. The formal descriptions of the other two LTL operators along with examples of their use is provided in Salamah [10].

Recursive definitions of a formula lead to a definition of a *subformula* as one of the auxiliary formulas in the construction of a given formula. Specifically, for our definition of LTL formulas, we have the following definition of an immediate subformula, which leads to the recursive definition of a subformula.

Definition 5 *A formula P is an* immediate subformula *of the formulas $\neg P$, $P \vee Q$, $Q \vee P$, $P \wedge Q$, $Q \wedge P$, $P \rightarrow Q$, $Q \rightarrow P$, XP, $\diamond P$, $\Box P$, $P\,U\,Q$, and $Q\,U\,P$.*

Definition 6

- *A formula P is its own subformula.*
- *If a formula P is an immediate subformula of the formula Q, then P is a subformula of Q.*
- *If P is a subformula of Q and Q is a subformula of R, then P is a subformula of R.*
- *Nothing else is a subformula.*

Definition 7

- *An LTL formula that does not contain any LTL temporal operations X, $\diamond$, $\Box$, and U, is called a* propositional formula.
- *A propositional formula P that is a subformula of an LTL formula Q is called a* propositional subformula *of Q.*
- *A formula P is called a* maximal propositional subformula *of the LTL formula Q if it is a propositional subformula of Q and it is not a subformula of any other propositional subformula of Q.*

For example, a formula $\neg p_1$ is a propositional subformula of the LTL formula

$$(\neg p_1 \wedge \neg p_2) \wedge ((\neg p_1 \wedge \neg p_2)\, U\, (p_1 \vee p_2))$$

(another particular case of $AtLeastOne_E^{LTL}$) but it is not its maximal propositional subformula – because it is a subformula of another propositional subformula $\neg p_1 \wedge \neg p_2$. On the other hand, $\neg p_1 \wedge \neg p_2$ is a maximal propositional subformula.

Now, we are ready to define the construction $P \&_r Q$. Informally, we replace each maximal propositional subformula P' of the formula P with $P' \wedge Q$.

- If P is a propositional formula, then $P \&_r Q$ is defined as $P \wedge Q$.
- If P is not a propositional formula, P is of the type $\neg R$, and $R \&_r Q$ is already defined, then $P \&_r Q$ is defined as $\neg(R \&_r Q)$.
- If P is not a propositional formula, P is of the type $R \vee R'$, and formulas $R \&_r Q$ and $R' \&_r Q$ are already defined, then $P \&_r Q$ is defined as

$$(R \&_r Q) \vee (R' \&_r Q).$$

- If P is not a propositional formula, P is of the type $R \wedge R'$, and formulas $R \&_r Q$ and $R' \&_r Q$ are already defined, then $P \&_r Q$ is defined as

$$(R \&_r Q) \wedge (R' \&_r Q).$$

- If P is not a propositional formula, P is of the type $R \rightarrow R'$, and formulas $R \&_r Q$ and $R' \&_r Q$ are already defined, then $P \&_r Q$ is defined as $(R \&_r Q) \rightarrow (R' \&_r Q)$.
- If P is of the type XR, and $R \&_r Q$ is already defined, then $P \&_r Q$ is defined as $X(R \&_r Q)$.
- If P is of the type $\diamond R$, and $R \&_r Q$ is already defined, then $P \&_r Q$ is defined as $\diamond(R \&_r Q)$.
- If P is of the type $\Box R$, and $R \&_r Q$ is already defined, then $P \&_r Q$ is defined as $\Box(R \&_r Q)$.
- If P is of the type $R\,U\,R'$, and formulas $R \&_r Q$ and $R' \&_r Q$ are already defined, then $P \&_r Q$ is defined as $(R \&_r Q)\, U\, (R' \&_r Q)$.

For example, when P is a formula:

$$(\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n) \wedge$$

$$((\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \vee p_2 \vee \ldots \vee p_n))$$

then $P \&_r Q$ is equivalent to the formula

$$(\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \wedge Q) \wedge$$

$$((\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \wedge Q)\, U\, ((p_1 \vee p_2 \vee \ldots \vee p_n) \wedge Q))$$

7 Specification of Patterns and Scopes with CPs in LTL

This section provides the LTL templates that can be used to define LTL specifications for all pattern/scope/CP combinations. We start by defining the formulas within the Global and Before R scopes. These formulas will be used to define the formulas for patterns within the remaining scopes as explained in Section 7.

Formulas for Patterns within $Global$ and $Before\ R$ Scopes. Tables 3 and 4 provide the LTL templates for patterns within the Global and Before R scopes respectively. Note that 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

Table 3. LTL Templates for Patterns within *Global* Scope

Pattern	LTL Formula
Absence of P	$\Box\neg P^{LTL}$
Existence of P	$\diamond P^{LTL}$
Q Precedes P_C	$\neg((\neg(Q^{LTL}\&_{-l}\neg P^{LTL}))\,U\,P^{LTL})$
Q Precedes P_E	$\neg((\neg(Q^{LTL}\&_{-l}\neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})))\,U\,(\neg p_1\wedge\ldots\wedge\neg p_n\wedge XP_H^{LTL}))$
Q Strictly Precedes P_C	$\neg((\neg(Q^{LTL}\,\&_r\,\neg P^{LTL}))\,U\,P^{LTL})$
Q Strictly Precedes P_E	$\neg((\neg(Q^{LTL}\&_r\neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})))\,U\,(\neg p_1\wedge\ldots\wedge\neg p_n\wedge XP_H^{LTL}))$
Q Responds to P	$\Box(P^{LTL}\rightarrow(P^{LTL}\&_l\diamond Q^{LTL}))$

this indicates that the type of the CP class is irrelevant and that the formula works for both types of CP classes. Also, in Tables 3-5, the terms P^{LTL}, L^{LTL}, R^{LTL}, Q^{LTL} refer to the LTL formula representing the CP class as described in Table 1.

Formulas for Patterns within the Remaining Scopes. Pattern formulas for the scopes $After\ L$, $Between\ L\ and\ R$, and $After\ L\ Until\ R$ can be generated using the formulas for the *Global* and $Before\ R$ scopes described in Tables 3 and 4. In this section, we use the symbol $\mathcal{P}_G^{LTL}$ to refer to formulas for the specific pattern within the *Global* scope, and we use the symbol $\mathcal{P}_{<\mathcal{R}}^{\mathcal{LTL}}$ to refer to formulas for the specific pattern within the $Before\ R$ scope. Table 5 provides the LTL templates for patterns within the $After\ L$, $Between\ L\ and\ R$, and $After\ L\ Until\ R$ scopes.

The following is an example of how these general LTL formulas can be used. Let us assume that the desired property can be described by the $Response(P,Q)$ pattern within the $Between\ L\ and\ R$ scope. In addition, let us assume that L is of type $Parallel_C(l_1,l_2)$, P is of type $Consecutive_C p_1,p_2)$, Q is of type $Parallel_C(q_1,q_2)$, and R is of type $AtLeastOne_C(r_1,r_2)$. To get the desired LTL formula for the $Response(P,Q)$ pattern within the $Between\ L\ and\ R$ scope, we first need to get the formula for this pattern within the $Before\ R$ scope (i.e. we need to find $\mathcal{P}_{<\mathcal{R}}^{\mathcal{LTL}}$). The general LTL formula corresponding to this pattern, scope, and CP classes combination is the one next to last in Table 4. The resulting LTL formula ($\mathcal{P}_{<\mathcal{R}}^{\mathcal{LTL}}$) for $Response(P,Q)$ pattern with $Before\ R$ scope is:

$$\neg((\neg(r_1\vee r_2))\,U\,(((p_1\wedge(\neg(r_1\vee r_2))\wedge X((p_2\wedge\neg(r_1\vee r_2))$$
$$\wedge(((\neg((q_1\wedge q_2\wedge\neg(r_1\vee r_2))))\,U\,(r_1\vee r_2)))))))).$$

We can then use this formula $\mathcal{P}_{<\mathcal{R}}^{\mathcal{LTL}}$ to generate the LTL formula for the $Response(P,Q)\ Between\ L\ and\ R$. Using the second general LTL formula in Table 5, the resulting

Table 4. LTL Templates for Patterns within $Before\ R$ Scope

Pattern	LTL Formula)
Absence of P Before R_C	$\neg((\neg R^{LTL})\,U\,((P^{LTL}\&_r\neg R^{LTL})\&_l\diamond R^{LTL}))$
Absence of P Before R_E	$((\diamond R^{LTL})\rightarrow\neg((\neg((\neg r_1\wedge\ldots\wedge\neg r_n)\wedge X(R_H^{LTL})))\,U\,(P^{LTL}\&_r(\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL})))))$
Existence of P Before R_C	$\neg((\neg(P^{LTL}\,\&_r\,\neg R^{LTL}))\,U\,R^{LTL})$
Existence of P Before R_E	$\neg((\neg(P^{LTL}\&_r\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL})))\,U\,(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL}))$
Q Precedes P_C *Before* R_C	$(\diamond R^{LTL}\rightarrow((\neg(P^{LTL}\&_r\neg R^{LTL}))\,U\,((Q^{LTL}\&_{-l}\neg P^{LTL})\vee R^{LTL}))$
Q Precedes P_E *Before* R_C	$(\diamond R^{LTL})\rightarrow((\neg((\neg p_1\wedge\ldots\wedge\neg p_n)\wedge\neg R^{LTL}\wedge X(P_H^{LTL}\&_r\neg R^{LTL})))\,U\,((Q^{LTL}\&_{-l}\neg(\neg p_1\wedge\ldots\neg p_n\wedge XP_H^{LTL}))\vee R^{LTL}))$
Q Precedes P_C *Before* R_E	$(\diamond R^{LTL})\rightarrow(((\neg(P^{LTL}\&_r\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL})))\,U\,((Q^{LTL}\&_{-l}\neg P^{LTL})\vee((\neg r_1\wedge\ldots\wedge\neg r_n)\wedge XR_H^{LTL}))))$
Q Precedes P_E *Before* R_E	$(\diamond R^{LTL})\rightarrow((\neg((\neg p_1\wedge\ldots\wedge\neg p_n)\wedge\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR^{LTL})_H\wedge X(P_H^{LTL}\&_r\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL}))))\,U\,((Q^{LTL}\&_{-l}\neg(\neg p_1\wedge\ldots\wedge\neg p_n\wedge XP_H^{LTL}))\vee((\neg r_1\wedge\ldots\wedge\neg r_n)\wedge XR_H^{LTL})))$
Q Str.Precedes P_C *Before* R_C	$(\diamond R^{LTL})\rightarrow((\neg(P^{LTL}\&_r\neg R^{LTL}))\,U\,((Q^{LTL}\&_r\neg P^{LTL})\vee R^{LTL}))$
Q Str.Precedes P_E *Before* R_C	$(\diamond R^{LTL})\rightarrow((\neg((\neg p_1\wedge\ldots\wedge\neg p_n)\wedge\neg R^{LTL}\wedge X(P_H^{LTL}\&_r\neg R^{LTL})))\,U\,((Q^{LTL}\&_r\neg(\neg p_1\wedge\ldots\neg p_n\wedge XP_H^{LTL}))\vee R^{LTL}))$
Q Str.Precedes P_C *Before* R_E	$(\diamond R^{LTL})\rightarrow(((\neg(P^{LTL}\&_r\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL})))\,U\,((Q^{LTL}\&_r\neg P^{LTL})\vee((\neg r_1\wedge\ldots\wedge\neg r_n)\wedge XR_H^{LTL}))))$
Q Str.Precedes P_E *Before* R_E	$(\diamond R^{LTL})\rightarrow((\neg((\neg p_1\wedge\ldots\wedge\neg p_n)\wedge\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR^{LTL})_H\wedge X(P_H^{LTL}\&_r\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL}))))\,U\,((Q^{LTL}\&_r\neg(\neg p_1\wedge\ldots\wedge\neg p_n\wedge XP_H^{LTL}))\vee((\neg r_1\wedge\ldots\wedge\neg r_n)\wedge XR_H^{LTL})))$
Q Responds to P Before R_C	$\neg((\neg R^{LTL})\,U\,((P^{LTL}\&_r\neg R^{LTL})\&_l((\neg(Q^{LTL}\&_r\neg R^{LTL}))\,U\,R^{LTL})))$
Q Responds to P Before R_E	$\neg((\neg((\neg r_1\wedge\ldots\wedge\neg r_n)\wedge X(R_H^{LTL})))\,U\,((P^{LTL}\&_r\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL}))\&_l((\neg(Q^{LTL}\&_r\neg(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL})))U(\neg r_1\wedge\ldots\wedge\neg r_n\wedge XR_H^{LTL}))))$

Table 5. LTL Templates for Patterns within the Remaining Scopes

Scope	LTL Formula)
$After\ L$	$\neg((\neg L^{LTL})\,U\,(L^{LTL}\&_l\neg\mathcal{P}_G^{LTL}))$
$Between\ L\ and\ R_C$	$\Box((L^{LTL}\&_l\neg R^{LTL})\rightarrow(L^{LTL}\&_l\mathcal{P}_{<R}^{LTL}))$
$Between\ L\ and\ R_E$	$\Box(L^{LTL}\rightarrow(L^{LTL}\&_l\mathcal{P}_{<R}^{LTL}))$
$After\ L\ Until\ R_C$	$\Box((L^{LTL}\&_l\neg R^{LTL})\ \rightarrow\ (L^{LTL}\&_l((\mathcal{P}_{<R}^{LTL}\ \wedge\ ((\neg\diamond R^{LTL})\rightarrow\mathcal{P}_G^{LTL})))))$
$After\ L\ Until\ R_E$	$\Box((L^{LTL})\rightarrow(L^{LTL}\&_l((\mathcal{P}_{<R}^{LTL}\wedge((\neg\diamond R^{LTL})\ \rightarrow\mathcal{P}_G^{LTL})))))$

formula is:

$$\Box((l_1\wedge l_2\wedge\neg(r_1\vee r_2))\rightarrow((l_1\wedge l_2\wedge(\mathcal{P}_{<\mathcal{R}}^{\mathcal{LTL}})))),$$

or

$$\Box((l_1\wedge l_2\wedge\neg(r_1\vee r_2))\rightarrow$$

$$((l_1\wedge l_2\wedge(\neg((\neg(r_1\vee r_2))\,U\,(((p_1\wedge(\neg(r_1\vee r_2))\wedge X$$

$$((p_2\wedge\neg(r_1\vee r_2))\wedge(((\neg((q_1\wedge q_2\wedge\neg(r_1\vee r_2))))\,U\,(r_1\vee r_2)\ldots).$$

8 Verification of LTL Templates

An advantage of formal verification techniques is that they can uncover subtle errors that are missed by traditional techniques. This motivates the use of these techniques in safety critical systems, where a failure could result in the loss of human life or expensive equipments. For this reason, it is important to verify that the formal specifications used in these techniques match the original intent of the specifier.

The LTL templates introduced in the previous section were verified using the following three techniques:

- formal proofs were used to verify the formulas for patterns within the *Global* scope (i.e., formulas in Table 4),
- testing using a model checker with equivalence classes and boundary analysis techniques was used to verify the formulas for patterns within the $Before\ R$ scope (i.e., formulas in Table 5), and
- reviews were used to verify the correctness of the formulas for patterns within the remaining three scopes (i.e., formulas in Table 6).

Because of the lack of space, this paper only discusses the proofs used for verifying the LTL templates for the global scope and provides one such proof. The remaining proofs along with the detailed descriptions of the techniques used in the verification of the LTL templates for the other scopes are provided in Salamah [10].

As mentioned above, the formulas for patterns within the global scope were verified using formal proofs. The proofs used definitions 1-3 of patterns within the *Global* scope provided in Section 4. This section highlights the proof of correctness for the LTL template corresponding to the *Response* pattern.

The main theorem of this paper follows:

Theorem 1 *For every pattern within the Global scope, the corresponding LTL formula is equivalent to the formal definition of the pattern in first order logic*

In order to prove this theorem, it is necessary to prove the correctness of each of the formulas in Table 4. Let us provide a proof for one case.

Proposition 1 *The LTL formula*

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

is equivalent to the formal definition of the pattern "Q Responds to P" in Global scope.

Proof.

1°. According to Definition 3, "Q responds to P" means that if P holds at some moment s, then Q holds at some moment s' for which $b_Q(s')\geq e_P(s)$. Formally, we can describe this property as follows:

$$\forall s\,(P(s)\rightarrow\exists s'\,(Q(s')\wedge b_Q(s')\geq e_P(s)) \qquad (1)$$

We want to prove that this formula is equivalent to the corresponding LTL formula

$$\Box(P\rightarrow(P\,\&_l\,\diamond\,Q)) \qquad (2)$$

Comment. To make the proof more readable, we describe the LTL formula P^{LTL} corresponding to P simply as P. We already know that the formulas P and P^{LTL} are equivalent, so from the logical viewpoint these simplified notations are well justified.

Similarly, we describe the LTL formula Q^{LTL} corresponding to Q simply as Q.

2°. To prove the desired equivalence, let us first reformulate the LTL formula (2) in terms of quantifiers.

2.1°. By the definition of the "always" operator $\Box$, the formula $\Box A$ means that A holds at all moments of time s, i.e., that $\forall s\,A(s)$. So, the above formula (2) is equivalent to

$$\forall s\,(P(s)\rightarrow(P\,\&_l\,\diamond\,Q)(s))) \qquad (3)$$

2.2°. The connective $(A\,\&_l\,B)(s)$ was defined as meaning that A holds at the moments s and B holds at the last of A-relevant moments of time, i.e., at the moment $e_A(s)$. Thus, the formula (2) can be equivalently reformulated as

$$\forall s\,(P(s)\rightarrow(P(s)\wedge(\diamond Q)(e_P(s)))). \qquad (4)$$

In this implication, if $P(s)$ holds, then of course $P(s)$ automatically holds, so we can delete this term from the right-hand side of the implication and simplify the above formula to

$$\forall s\,(P(s) \rightarrow (\diamond Q)(e_P(s))). \qquad (5)$$

2.3°. By the definition of the "eventually" operator $\diamond$, the formula $\diamond A$ means that A holds either at the current moment of time s, or at some later moment of time $s'' > s$, i.e., that $\exists s''\,(A(s'') \wedge s'' \geq s)$.

Thus, the formula (2) is equivalent to

$$\forall s\,(P(s) \rightarrow \exists s''\,(Q(s'') \wedge s'' \geq e_P(s))). \qquad (6)$$

3°. Since the LTL formula (2) is equivalent to (6), to complete our proof we only need to prove the equivalence between (1) and (6).

3.1°. Let us first prove that (6) implies (1).

Indeed, let us assume that (6) holds, and that $P(s)$ holds for some moment of time s. Then, the formula (6) implies that for some $s'' \geq s$, we have $Q(s'')$ and $s'' \geq e_P(s)$.

We have defined $b_A(s)$ as the first moment of time $\geq s$ for which a certain condition holds. Thus, we always have $b_A(s) \geq s$.

In particular, we have $b_Q(s'') \geq s''$. From $s'' \geq e_P(s)$, we can now conclude that $b_Q(s'') \geq e_P(s)$. Thus, for $s' = s''$, we have $b_Q(s') \geq e_P(s)$ and $Q(s')$. So, we have proven the formula (1).

3.2°. Let us now prove that (1) implies (6).

Indeed, assume that (1) holds, and $P(s)$ holds for some moment of time s. Then, according to (1), there exists a moment s' for which $b_Q(s') \geq e_P(s)$ and $Q(s')$.

By definition of $b_A(s)$, we can easily conclude that the formula A always holds at the moment $b_A(s)$: $A(b_A(s))$. Thus, for $s'' = b_Q(s)$, we have $Q(s'')$ and $s'' \geq e_P(s)$. So, we have proven the formula (6).

The equivalence is proven, hence Proposition 1 is true.

9 Summary and Future Work

This paper provides formal descriptions of the different CP classes defined by Mondragon et al. [7]. In addition, it presents formal descriptions of the patterns and scopes defined by Dwyer et al. [2] when using CP classes. The main contributions of the paper are the definitions of 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 (formulas in Table 4) have been verified using formal proofs [10]. Formulas for the remaining scopes (formulas in Tables 5 and 6) were verified using testing and formal reviews [3, 10].

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 was partly funded by NSF grants EAR-0225670 and HRD-0734825.

References

[1] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M., "NUSMV: a new Symbolic Model Verifier", *Int'l Conf. 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," *Proc. 21st Int'l Conf. on Software Engineering*, Los Angeles, CA, 1999, 411–420.

[3] Garcia, L.A., *Automatic Generation and Verification of Complex Pattern-Based Specifications*, University of Texas at El Paso, Department of Computer Science, Master's thesis, July 2007.

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

[5] Holzmann, G.J., *The SPIN Model Checker,* Addison-Wesley, 2004.

[6] Havelund, K., and Pressburger, T., "Model Checking Java Programs using Java PathFinder", *Int'l J. on Software Tools for Technology Transfer*, 2(4), April 2000.

[7] Mondragon, O., and Gates, A.Q., "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," *Int'l J. Software Engineering and Knowledge Engineering,* 14(1), Feb. 2004.

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

[9] Rushby, J., "Theorem Proving for Verification," *Modelling and Verification of Parallel Processes,* June 2000.

[10] 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.

[11] Smith, R., Avrunin, G., Clarke, L., and Osterweil, L., "PROPEL: an approach supporting property elucidation" *Proc. 22rd Int'l Conf. on Software Engineering*, ICSE, May 2002, Orlando, Florida

[12] Spec Patterns, http://patterns.projects.cis.ksu.edu/, August 2007.

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