Generating Linear Temporal Logic Formulas for Pattern-Based Specifications: Towards Making Software Model Checking User Friendlier

Salamah Salamah, Ann Q. Gates, Steve Roach, and Vladik Kreinovich
Dept. of Computer Science, University of Texas at El Paso
El Paso, TX 79968, USA
isalamah, agates, sroach, and vladik@utep.edu

Abstract

Property classifications and patterns, i.e., high-level abstractions that describe common behavior, have been used to assist practitioners in specifying properties. 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 formal definitions of patterns and scopes when defined using CP classes. In addition, we provide general (abstract) LTL formulas that can be used to generate LTL specifications for all combinations of pattern, scope, and CP classes.

1 Introduction and Motivation

Although the use of formal verification methods such as model checking [7], theorem proving [13], and runtime monitoring [14] improve the dependability of programs, they are not used in standard software development practices. One reason for this hesitance is the high level of mathematical sophistication required for reading and writing the formal specifications [6].

Linear Temporal Logic (LTL) is a prominent formal specification language that is highly expressive and is widely used in formal verification tools such as the model checker SPIN [7]. SPIN has been used in the verification of a variety of systems such as security protocols [1] and flight software [9]. In addition,

LTL is used by the model checkers NUSMV [2] and Java Path-Finder [8], and it is also used in the runtime verification of Java programs [14].

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 can be found in [12].

1.1 Specification Pattern System (SPS)

Writing formal specification, particularly those involving time, is difficult. The Specification Pattern System [3] provides patterns and scopes to assist the practitioner in formally specifying software properties. *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*. The descriptions given below are taken verbatim from the SPS website [4].

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

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 $After\ L\ Until\ R$. *Global* denotes the entire program execution; $Before\ R$ denotes the execution before the first time the condition R holds; $After\ L$ denotes execution after the first time L holds; $Between\ L\ And\ R$ denotes the execution between intervals defined by L and R; and $After\ L\ Until$ denotes the execution between intervals defined by L and R and, in the case when R does not occur, until the end of execution.

The SPS website provides a formal descriptions of patterns and scopes in several specification languages including Linear Temporal Logic (LTL), Computational Tree Logic (CTL), and Graphical Interval Logic (GIL). These formulas are provided for patterns and scopes involving *single (atomic) propositions*, i.e., patterns and scopes in which P, Q, L, and R each of which occur at a single state of execution. The website also provides examples of properties that can be defined using these patterns and scopes. For example, the property "When a connection is made to the SMTP server, all queued messages in the OutBox mail will be transferred to the server" can be defined using the $Existence\ P$ pattern within the $Before\ R$ scope, where R stands for 'all queued messages in the OutBox mail are transferred to the server' and P stands for 'connection is made to the SMTP server'. The LTL formula for this property is $(\diamond R \rightarrow (\neg RUP))$

SPS also provides formulas for the cases of multiple causes and single effect (when P is made of multiple propositions and Q is single) and of single cause and multiple effects (when P is single and Q is composed of multiple propositions).

1.2 Composite Propositions (CP)

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. This property can be described using the $Existence(P)$ pattern within the $Between\ L$ and R scope where 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. [11] 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 a formal descriptions of these CP classes in LTL.

Some of the corresponding patterns can be described in Future Interval Logic (FIL) language, a language that is similar to LTL, but less expressive than LTL. For example, FIL cannot describe a practically important property that an event p must hold at the next state. The corresponding translations have been implemented in the Property Specification tool (Prospec) [10], which uses patterns and scopes involving composite propositions to generate formal specifications in

FIL. In comparison to LTL, FIL has two limitations: first, due to the limited expressiveness of FIL, not all patterns and scopes involving composite propositions can be represented; second, FIL is not as widely used in formal verification tools, so the use of FIL restricts the software engineer's ability to use the resulting specifications. It is, therefore, important to provide a translation of all possible patterns and scopes involving composite propositions into the more expressive (and more widely used) language LTL. It is also important to show that these translations are correct for all patterns and scopes. This paper concentrates on providing the LTL translations for all patter/scope/CP combinations. The matter of proving the correctness of these translations is left as future work.

2 Composite Propositions: A Formal Description

Mondragon et al. [11] defined eight CP classes to describe sequential and concurrent behavior. CP classes are categorized to be either of condition type (denoted with a subscript C) or event type (denoted with a subscript E). A condition is a proposition that holds over multiple consecutive states, where an event represents a change in the truth value of a proposition in two consecutive states. The four CP classes of condition type are defined as follows:

- $AtLeastOne_C(p_1, \ldots, p_n)$ means that at least one of the propositions p_i holds at the given state s
- $Parallel_C(p_1, \ldots, p_n)$ means that all propositions p_i hold at state s
- $Consecutive_C(p_1, \ldots, p_n)$ means that p_1 holds at state $s_1 = s$, p_2 holds at state $s_2 = s + 1$, ..., and p_n holds at state $s_n = s + (n - 1)$
- $Eventual_C(p_1, \ldots, p_n)$ means that p_1 holds at state $s_1 = s$, p_2 holds at some state $s_2 > s_1$, ..., and p_n holds at some state $s_n > s_{n-1}$

The four CP classes of event type are $AtLeastOne_E(p_1, \ldots, p_n)$, $Parallel_E(p_1, \ldots, p_n)$, $Consecutive_E(p_1, \ldots, p_n)$, and $Eventual_E(p_1, \ldots, p_n)$. These can be defined in terms of a new class of auxiliary formulas $T_H(p_1, \ldots, p_n)$ (CP classes of type hold). The main motivation for T_H is that in CP of type condition, we only require each p_i to hold at a certain state s_i, and we do not make any assumptions about other propositions p_j $(j \neq i)$ at state s_i. In some practical applications, it is important to require that p_i become true in the prescribed order, i.e., that not only p_i becomes true in state s_i, but that it is false until then. In addition to using CP classes of type hold to define CP classes of type event, CP classes of type hold make it easier to define the general LTL formulas in Section 6, which is the main goal of this work. The four CP classes of hold type are defined as follows:

- $AtLeastOne_H(p_1, \ldots, p_n)$ is equivalent to $AtLeastOne_C$ and means that at least one of the propositions p_i is true at the given state s

- $Parallel_H(p_1, \ldots, p_n)$ is equivalent to $Parallel_C$ and means that all of the propositions p_i are true at the given state s
- $Consecutive_H(p_1, \ldots, p_n)$ means that:
  - p_1 is true at state $s_1 = s$, and $p_2 \ldots p_n$ are false at s_1,
  - p_2 is true at state $s_2 = s + 1$, and $p_3 \ldots p_n$ are false at s_2,
  - $\ldots$,
  - p_{n-1} is true at state $s_{n-1} = s + (n-1)$ and p_n is false at s_{n-1}
  - and p_n is true at state $s_n = s + (n)$
- $Eventual_H(p_1, \ldots, p_n)$ means that:
  - at state $s_1 = s$, the proposition p_1 is true and the following propositions $p_2, \ldots, p_n$ are false; these propositions $p_2, \ldots, p_n$ remain false until some future state $s_2 > s_1$ where only p_2 becomes true;
  - $\ldots$
  - at state $s_i > s_{i-1}$ $(1 < i < n)$, the proposition p_i is true and the following propositions $p_{i+1}, \ldots, p_n$ are false; these propositions $p_{i+1}, \ldots, p_n$ remain false until some future state $s_{i+1} > s_i$ in which p_{i+1} is true and the remaining propositions $p_{i+2} \ldots p_n$ are false;
  - $\ldots$
  - At state $s_{n-1} > s_{n-2}$, the proposition p_{n-1} is true and proposition p_n is false;
  - finally, at state $s_n > s_{n-1}$, the proposition p_n is true.

CP classes of type event are defined using the above definition of CP classes of type hold as follows:

Definition 1 *We say that a CP of type event (i.e, $T_E(p_1, \ldots, p_n)$)* holds *at state s if at this state, all propositions p_i are false, and they remain false until some future state s' when the composite proposition $T_H(p_1, \ldots, p_n)$ becomes true.*

For example, a composite proposition $AtLeastOne_E(p_1, \ldots, p_n)$ holds at state s if all the propositions $p_1, \ldots, p_n$ are false at s, and at least one of these propositions $p_1, \ldots, p_n$ is true at some future state $s' > s$. Table 1 provides a formal description of the above mentioned CP classes in LTL. We use the notation P^{LTL} to refer to the LTL formula describing a CP class.

3 Patterns Involving Single and Composite Propositions: Motivations and Definitions

As we mentioned in Section 1.1, Dwyer et al. defined the notions of patterns and scopes to assist in the definition of formal specifications. Patterns provide

Table 1: Description of CP Classes in LTL

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

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, and
- 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. [11], 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.

3.1 Absence and Existence: Precise Descriptions.

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:

$$\boxed{\Box \neg P^{LTL}} \tag{1}$$

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:

$$\boxed{\diamond P^{LTL}} \tag{2}$$

3.2 Precedence, Strict Precedence, and Response: Precise Definition

For single proposition, the meaning of "precedes", "strictly precedes", and "responds" is straightforward:

- q precedes p means that every time the property p holds, the property q must hold either in a previous state or at the same state;
- q strictly precedes p means that every time the property p holds, the property q must hold in a previous state;
- q responds to p means that every time the property p holds, the property q must hold either at the same state or at a later state.

To extend the above meanings to CP, we need to explain what "after" and "before" mean for in 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. In these terms, P occurs before Q if $e_P \leq b_Q$ and P occurs after Q if $b_P \geq e_Q$.

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 $e_P(s)$ 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 $P = AtLeastOne_E(p_1, \ldots, p_n)$ that holds at state s, we take, as $b_P(s) = e_P(s)$, the first state $s' > s$ at which one of the propositions p_i becomes true.
- 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 $b_P(s) = e_P(s)$, the first state $s' > s$ at which all the propositions p_i become true.

- 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 first state $s' > s$ at which the proposition p_1 becomes true, and we take $e_P(s) = s' + (n-1)$.
- 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 first state s_1 at which 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.

Now that we have defined the meaning of before and after in the case of CP, we can give a precise definitions of Precedence, Strict Precedence, and Response with Global scope:

Definition 2 *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 3 *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 4 *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.*

4 Non-Global Scopes Involving Composite Propositions: Motivations and Definitions

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 1.1. We also provide semantics for these patterns.

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

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

- For the "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 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 5 *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 2 provides a formal description of what it means for a pattern to hold within a scope.

Now that we have defined what it means for a pattern to hold within the different types of scopes, we are ready to provide the LTL description of the five patterns within the scopes ("Before R", "After L", "Between L And R", and "After L Until R").

[1]These definitions use the notion of beginning state and ending state as defined in Section 3.2

5 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 6.

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 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 $p_1 \wedge X p_2$ 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, $(p_1 \wedge X p_2)$ 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 original "and" $A \wedge B$ which means that B holds at the original moment of time t and to the new "repeated and" $A \,\&_r, B$ meaning that B holds at all moments of time which are relevant for the LTL formula A, we will also need two more operation.

- 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.

In the following text, we will give formal definitions of these operations. Specifically, the definition of $\&_r$ is given for general LTL formulas; for $\&_{-l}$ and $\&_l$, we will only give the definition for the particular cases needed in our patterns (i.e, in the cases of "anding" two CP classes).

5.1 The New Operator "$\&_r$"

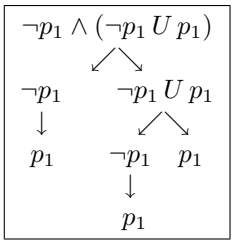
In logic in general, recursive definitions of a formula lead to a definition of a *subformula* – as one of the auxiliary formulas in the construction of a given formula. Specifically, for our definition of LTL formulas, we have the following definition of an immediate subformula which leads to the recursive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

Definition 6 *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 7

- *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.*

Subformulas of a given formula P form a (parse) tree, in which the formula P is a root, immediate subformulas are children, and single propositions are leaves. For example, the LTL formula $\neg p_1 \wedge (\neg p_1\,U\,p_1)$ (the simplest case of $AtLeastOne_E^{LTL}$) has the following subformula tree:



Definition 8

- *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 $X\,R$, 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

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

(general case of $AtLeastOne_E^{LTL}$), then $P \,\&_r\, Q$ is the formula

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

5.2 The New Operators "$\&_{-l}$" and "$\&_l$"

Unlike the operator $\&_r$, we do not provide a general definition of the two new operators the new operators "$\&_{-1}$" and "$\&_l$". We provide a definition for these operators in term of CP class. In other word, we provide definitions for (A $\&_{-1}$ B) and (A $\&_l B$) in the cases where A and B are both CP classes. This definition does not extend to any A and B such that A and B are LTL formulas but not CP classes.

Definition 9 *the operator "$\&_{-l}$" is defined as follows:*

- *When P is of the type $T_C(p_1, \ldots, p_n)$ or $T_H(p_1, \ldots, p_n)$, with $T = Parallel$ or $T = AtLeastOne$, then $P \,\&_{-l}\, A$ is defined as $P \wedge A$.*
- *When P is of the type $T_C(p_1, \ldots, p_n)$, with $T = Consecutive$ or $T = Eventual$, then $P \,\&_l\, A$ is defined as $T_C(p_1 \wedge A, \ldots, p_{n-1} \wedge A, p_n)$.*
- *When P is of the type $T_H(p_1, \ldots, p_n)$, with $T = Consecutive$ or $T = Eventual$, then $P \,\&_{-l}\, A$ is defined as*

$$T_C(p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \wedge A, \ldots, p_{n-1} \wedge \neg p_n \wedge A, p_n).$$

- *When P is of the type $T_E(p_1, \ldots, p_n$, then $P \,\&_{-l}\, A$ is defined as*

$$(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge A) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n \wedge A)\, U\, (T_H(p_1, \ldots, p_n)\, \&_{-l} A)).$$

Definition 10 *the operator "$\&_l$" is defined as follows:*

- *When P is of the type $T_C(p_1, \ldots, p_n)$ or $T_H(p_1, \ldots, p_n)$, with $T = Parallel$ or $T = AtLeastOne$, then $P \,\&_l\, A$ is defined as $P \wedge A$.*
- *When P is of the type $T_C(p_1, \ldots, p_n)$, with $T = Consecutive$ or $T = Eventual$, then $P \,\&_l\, A$ is defined as $T_C(p_1, \ldots, p_{n-1}, p_n \wedge A)$.*
- *When P is of the type $T_H(p_1, \ldots, p_n)$, with $T = Consecutive$ or $T = Eventual$, then $P \,\&_l\, A$ is defined as*

$$T_C(p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n, \ldots, p_{n-1} \wedge \neg p_n, p_n \wedge A).$$

- *When P is of the type $T_E(p_1, \ldots, p_n$, then $P \,\&_l\, A$ is defined as*

$$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (T_H(p_1, \ldots, p_n)\, \&_l A)).$$

6 General LTL Formulas for Patterns and Scopes With CP

This Section provides the abstract LTL formulas 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 6.2.

6.1 Formulas for Patterns Within Global and Before R Scopes

Table 3 and 4 provide the abstract LTL formulas 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 available, then this indicates that the type of the CP class is not relevant and that the formula works for both types of CP classes. Also, in Table 3, the terms P^{LTL}, L^{LTL}, R^{LTL}, Q^{LTL} refer to the LTL formula representing the CP class as described in Table 1

Table 3: Abstract LTL Formulas 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_C *Precedes* P_E	$\neg((\neg Q^{LTL})\, U\, ((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg Q^{LTL} \wedge X(P_H^{LTL})))$
Q_E *Precedes* P_E	$\neg((\neg(Q^{LTL} \&_{-l} \neg P_H^{LTL}))\, U\, (((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge X\,(P_H^{LTL}))))$
Q Strictly Precedes P_C	$\neg((\neg(Q^{LTL} \&_r \neg P^{LTL}))\ U\ P^{LTL})$
Q_C *Strictly Precedes* P_E	$\neg((\neg Q^{LTL})\, U\, ((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg Q^{LTL} \wedge X(P_H^{LTL} \wedge \neg Q^{LTL})))$
Q_E *Strictly Precedes* P_E	$\neg((\neg(Q^{LTL} \&_r \neg P_H^{LTL}))\, U\, (((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge X(P_H^{LTL}))))$
Q Responds to P	$\Box(P^{LTL} \rightarrow (P^{LTL} \&_l \diamond Q^{LTL}))$

6.2 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 description of the abstract LTL formulas 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 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) Before R is:

Table 4: Abstract LTL Formulas 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	$(\diamond R^{LTL}) \rightarrow \neg((\neg((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X(R_H^{LTL})))\, U\, (P^{LTL} \&_r \neg R_H^{LTL}))$
Existence of P Before R_C	$\neg((\neg(P^{LTL} \&_r \neg R^{LTL}))\, U\, R^{LTL})$
Existence of P Before R_E	$(\diamond R^{LTL}) \rightarrow ((\neg((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X(R_H^{LTL})))\, U\, (P^{LTL} \&_r \neg R_H^{LTL}))$
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	$(\diamond R^{LTL}) \rightarrow ((\neg((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg R^{LTL} \wedge X(P_H^{LTL} \&_r \neg R^{LTL})))$ $U\, ((Q^{LTL} \&_{-l} \neg(P_H^{LTL})) \vee R^{LTL}))$
Q Precedes P_C Before R_E	$(\diamond R^{LTL}) \rightarrow (((\neg(P^{LTL} \&_r \neg R_H^{LTL}))\, U\, ((Q^{LTL} \&_{-l} \neg P^{LTL}) \vee ((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X R_H^{LTL}))))$
Q Precedes P_E Before R_E	$(\diamond R^{LTL}) \rightarrow ((\neg((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg R_H^{LTL} \wedge X(P_H^{LTL} \&_r \neg R_H^{LTL})))$ $U\, ((Q^{LTL} \&_{-l} \neg P_H^{LTL}) \vee ((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X R_H^{LTL})))$
Q Strictly 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 Strictly Precedes P_E Before R_C	$(\diamond R^{LTL}) \rightarrow ((\neg((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg R^{LTL} \wedge X(P_H^{LTL} \&_r \neg R^{LTL})))$ $U\, ((Q^{LTL} \&_r \neg(P_H^{LTL})) \vee R^{LTL}))$
Q Strictly Precedes P_C Before R_E	$(\diamond R^{LTL}) \rightarrow (((\neg(P^{LTL} \&_r \neg R_H^{LTL}))\, U\, ((Q^{LTL} \&_r \neg P^{LTL}) \vee ((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X R_H^{LTL}))))$
Q Strictly Precedes P_E Before R_E	$(\diamond R^{LTL}) \rightarrow ((\neg((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg R_H^{LTL} \wedge X(P_H^{LTL} \&_r \neg R_H^{LTL})))$ $U\, ((Q^{LTL} \&_r \neg P_H^{LTL}) \vee ((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X R_H^{LTL})))$
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	$\neg((\neg((\neg r_1 \wedge \neg r_2 \wedge \ldots \wedge \neg r_n) \wedge X(R_H^{LTL})))$ $U\, ((P^{LTL} \&_r \neg R_H^{LTL}) \&_l ((\neg(Q^{LTL} \&_r \neg R_H^{LTL}))U R_H^{LTL})))$

Table 5: Abstract LTL Formulas 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	$\Box((L^{LTL}) \rightarrow (L^{LTL} \&_l ((\mathcal{P}_{<R}^{LTL} \wedge ((\neg \diamond R^{LTL}) \rightarrow \mathcal{P}_G^{LTL})))))$

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

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 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

$$\Box((l_1 \wedge l_2 \wedge \neg(r_1 \vee r_2)) \rightarrow$$
$$((l_1 \wedge l_2 \wedge (\neg((\neg(r_1 \vee r_2))\, U\, (((p_1 \wedge (\neg(r_1 \vee r_2)) \wedge X((p_2 \wedge \neg(r_1 \vee r_2))$$
$$\wedge(((\neg((q_1 \wedge q_2 \wedge \neg(r_1 \vee r_2))))\, U\, (r_1 \vee r_2))))))))))))$$

7 Summery and Future Work

The work in this paper provided formal descriptions of the different composite propositions (CP) classes defined by Mondragon et al. [11]. In addition, we formally described the patterns and scopes defined by Dweyer et al. [3] 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 LTL translations have been systematically tested using equivalence classes and boundary analysis testing techniques [5]. The next step in this work consists of providing formal proofs that these translations to LTL do indeed meet the characteristics of patterns and scopes defined using CP classes. 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] Audun, J.,"Security Protocol Verification using SPIN", *Proceedings of the First SPIN Workshop*, October 1995.

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

[3] 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.

[4] http://patterns.projects.cis.ksu.edu/

[5] Ghezzi, C., Jazayeri, M., and Mandirioli, D., *Fundamentals of Software Engineering,* Prentice Hall, 2002.

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

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

[8] Havelund, K., and Pressburger, T., "Model Checking Java Programs using Java PathFinder", *International Journal on Software Tools for Technology Transfer*, 2(4), April 2000.

[9] Glück, P, R., and Holzmann, G. J., "Using SPIN Model Checking for Flight Software Verification," *Aerospace Conference, IEEE*, March 2002

[10] Mondragon, O., Gates, A. Q., and Roach, S., "Prospec: Support for Elicitation and Formal Specification of Software Properties," in O. Sokolsky and M. Viswanathan (Eds.), *Proceedings of Runtime Verification Workshop, ENTCS*, 89(2), 2004.

[11] 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.

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

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

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