Validated Templates for Specification of Complex LTL Formulas

Salamah Salamah
Department of Electrical, computer, Software, and Systems Engineering
Embry Riddle Aeronautical University

600 S. Clyde Morris Blvd., Daytona Beach, Florida, 32114

Ann Gates and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso

500 W. University Blvd., El Paso, Texas, 79925

Abstract

Formal verification approaches that check software correctness against formal specifications have been shown to improve program dependability. Tools such as Specification Pattern System (SPS) and Property Specification (Prospec) support the generation of formal specifications. SPS has defined a set of patterns (common recurring properties) and scopes (system states over which a pattern must hold) that allows a user to generate formal specifications by using direct substitution of propositions into parameters of selected patterns and scopes. Prospec extended SPS to support the definition of patterns and scopes that include the ability to specify parameters with multiple propositions (referred to as composite propositions or CPs), allowing the specification of sequential and concurrent behavior. Prospec generates formal specifications in Future Interval Logic (FIL) using direct substitution of CPs into pattern and scope parameters. While substitution works trivially for FIL, it does not work for Linear Temporal Logic (LTL), a highly expressive language that supports specification of software properties such as safety and liveness. LTL is important because of its use in the model checker Spin, the ACM 2001 system Software Award winning tool, and NuSMV. This paper introduces abstract LTL templates to support automated generation of LTL formulas for complex properties in Prospec. In addition, it presents formal proofs and testing to demonstrate that the templates indeed generate the intended LTL formulas.

Keywords: Formal Specifications, LTL, Pattern, Scope, Composite Propositions, Model Checking

1. Introduction

Today more than ever, society depends on complex software systems to fulfill personal needs and to conduct business. Software is an integral part of many mission and safety critical systems. In transit systems, for example, software is used in railway signaling, train control, fault detection, and notification systems among other things [26]. Implanted drug delivery pumps, pacemakers and defibrillators, and automated cancer cell and DNA-based diagnostics systems are examples of medical equipment built around embedded software systems [10]. The National Aeronautics and Space Administration (NASA) Space Shuttle program, a multi-billion dollar program built on computer software and hardware, is an example of a safety critical system that could lead to the loss of life and huge finances if it fails.

Because of society's dependence on computers, it is vital to assure that software systems behave as intended. The estimated cost due to software errors in the aerospace industry alone was $6 billion in 1999 [17]. The numbers are even more alarming when considering that software errors cost U.S. economy $59.5 billion annually [15]. It is imperative that the software industry continue to invest in software assurance approaches, techniques, and tools.

Although the use of formal verification methods such as model checking [7], theorem proving [16], and runtime monitoring [24] has been shown to improve the dependability of programs, software development professionals have yet to adopt them. The reasons for this hesitance include the high level of mathematical sophistication required for reading and/or writing formal specifications needed for the use of these approaches [6].

Linear Temporal Logic (LTL) [11] is a prominent formal specification language that is highly expressive and widely used in formal verification tools such as the model checkers SPIN [7] 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 provides the temporal operators *next* (X), *eventually* ($\diamond$), *always* ($\Box$), *until*, (U), *weak until* (W), and *release* (R). These formulas assume discrete time, i.e., states $s = 0, 1, 2, \ldots$ The meaning of the temporal operators is straightforward:[1]

- The formula Xp holds at state s if p holds at the next state $s + 1$,
- the formula $p\,U\,q$ holds at state s, if there is a state $s' \geq s$ at

[1] In this work we only consider the first four of these operators

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 $\diamond p$ holds at state s if p is true at some state $s' \geq s$, and
- the formula $\Box p$ holds at state s if p is true at all states $s' \geq s$.

One problem with LTL is that, when specifying software properties, the resulting LTL expressions can become difficult to write and understand. For example, consider the following LTL specification: $\Box(a \rightarrow \diamond(p \wedge \diamond((\neg a) \wedge \neg p)))$, where a denotes "Train approaches the station." and p denotes "Train passes the station." It is not immediately obvious that the specification describes the following: "If a train approaches the station, then the train will pass the station and, after it passes, the train does not approach or pass the station."

To assist users in the generation of LTL specifications, Dwyer et al. [3, 4] defined a set of patterns to represent the most commonly used formal properties. The work also defined a set of scopes of system execution where the pattern of interest must hold. Each pattern and scope combination can be mapped to specifications in multiple formal languages including LTL. Using the notions of patterns and scopes a user can define system properties in LTL without being an expert in the language. Section 2 provides more details on SPS' patterns and scopes.

In SPS, patterns and scopes parameters are defined using atomic propositions (i.e., each pattern and scope parameter is defined using a single proposition with a single truth value). To extend the expressiveness of SPS, Mondragon et al. [12, 13] developed the Property Specification (Prospec) tool. Prospec attempts to extended SPS through the definition of a set of composite propositions (CP) classes with the intent of using these to define pattern and scope parameters. A complete description of Mondragon's composite proposition classes can be found in Section 2.

Although SPS provides LTL formulas for basic patterns and scopes (ones that use single, "atomic", propositions to define parameters) and Mondragon et al. [13] provided LTL semantics for the CP classes as described in Table 1. below, 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. We delay the introduction of a formal example of this inadequacy after Section 2 where we describe the notions of pattern, scope, and CP in more details.

This work aims at creating high-level LTL templates that can be used to define LTL formulas for complex system properties. These LTL templates take as an input a combination of of pattern, scope, and CP classes that describe the desired property. The output of the templates is an LTL formula that can be used by formal verification tools such as model checkers. However, in order to be able to combine patterns, scopes, and CP classes to generate LTL formulas, we first need to provide a precise definition of the semantics of each pattern and scope when used in conjunction with CP classes and vice versa. Providing these formal definitions is a secondary goal of this paper.

The rest of the paper is organized as follows; Section 2 provides the background of the work including SPS' patterns and scopes, as well as a more detailed description of the CP classes introduced by Mondragon. Section 2 also includes an example to show the problems that can arise when using direct substitution within LTL. In Section 3 we provide a formal definition of the meaning of patterns and scopes when defined using CP classes. Section 4 introduced a new LTL operators that will be used to simplify the abstract LTL templates. Those LTL templates are described in Section 5 along with an example of their use. In Section 6 we show the methods we used to validate that the LTL templates generate LTL that meet the original meaning of the selected pattern, scope, and CP combination. The paper concludes with summary and future work followed by the references.

2. Background

This section provides the background information needed for the rest of the paper. We describe the notions of patterns, scopes as defined by Dwyer[3]. We also describe Mondragon's CP classes as well as provide a more formal description of these classes. These formal descriptions of CP classes are necessary for describing the semantics of patterns and scopes that use CP classes, which we introduce in Section 3.

2.1. Specification Pattern System (SPS)

Writing formal specification, particularly those involving time, is difficult. The Specification Pattern System (SPS) [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 pre-condition 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 patterns and scopes for formal specification languages such as 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". Using SPS, we obtain the following LTL formula for this property: $(\Box\neg R) \vee (\neg R\ U\ (P \wedge \neg R))$. A complete list of the LTL formulas for SPS' pattern/scope combinations can be found in [4].

2.2. 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 "every time data is sent at state s_i, data is read at state $s_j \geq s_i$, the data is processed at state $s_k \geq s_j$, and data is stored at state $s_l \geq s_k$." 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 multiple propositions p_1 and p_2 (data is read and data is processed, respectively).

To describe such patterns, Mondragon et al. [13] 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.

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 practical property that an event p must hold at the next state. The corresponding translations have been implemented in the Property Specification tool (Prospec) [12], 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. The rest of this paper concentrates on providing the LTL templates for all pattern, scope, and CP combinations. We also validate these templates using formal proofs for templates of all patterns within the *Global* scope and we used model checking based testing technique for validating the remaining templates.

2.2.1. Composite Propositions: A Formal Description

Mondragon et al. [13] 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:

- $AtLeastOne_C(p_1, \ldots, p_n)$ holds at state s if at least one proposition p_i, where $1 \leq i \leq n$, is true at state s
- $Parallel_C(p_1, \ldots, p_n)$ holds at state s if all propositions p_i, where $1 \leq i \leq n$, are true at state s
- $Consecutive_C(p_1, \ldots, p_n)$ holds at state s if p_1 is true at state s, p_2 is true at state $s+1, \ldots,$ and p_n is true at state $s+(n-1)$
- $Eventual_C(p_1, \ldots, p_n)$ holds at state s_1 if p_1 is true at state s_1, p_2 is true at some state $s_2 > s_1, \ldots,$ and p_n is true at some state $s_n > s_{n-1}$

The four CP classes of type event are $AtLeastOne_E(p_1, \ldots, p_n)$, $Parallel_E(p_1, \ldots, p_n)$, $Consecutive_E(p_1, \ldots, p_n)$, and $Eventual_E(p_1, \ldots, p_n)$ can be defined in terms of a new class of auxiliary formulas $T_H(p_1, \ldots, p_n)$. These CP classes are 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 defined equivalently to $AtLeastOne_C$
- $Parallel_H(p_1, \ldots, p_n)$ is defined equivalently to $Parallel_C$
- $Consecutive_H(p_1, \ldots, p_n)$ 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:

Table 1: Description of CP Classes in LTL

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

- 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.

2.2.2. Problem with Direct Substitution

As mentioned above, SPS provides LTL formulas for every pattern/scope combination, and Mondragon et. al/ provides LTL formulas for every CP class. However, directly substituting a pattern or a scope parameter by the LTL formula for the desired CP class might generate an LTL formula that does not match the original meaning of the property. The following example shows the problem with direct substitution.

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 a goal of this paper.

3. Formal Description of Patterns and Scopes Using Composite Propositions

As shown in Section 2, SPS provided LTL formulas for all 25 combinations of patterns and scopes. However, these definitions are only adequate for patterns and scopes with a single proposition for each parameter. In order to define LTL templates for patterns and scopes defined using CP classes, it is important to provide a precise description of what these patterns and scopes mean when defined using CP classes and not only by a single proposition.

3.1. Patterns Involving Single and Composite Propositions: Motivations and Definitions

As we mentioned in Section 1.2, Dwyer et al.[3] 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:

- $Absence$ of P,
- $Existence$ 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. [13], 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.

3.1.1. 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 template formula corresponding to the *Absence* of P is:

$$\boxed{\Box\neg P^{LTL}} \qquad (1)$$

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 template formula corresponding to the *Existence* of P is:

$$\boxed{\Diamond P^{LTL}} \qquad (2)$$

3.1.2. 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 property p holds, property q must hold either in a previous state or at the same state;
- q strictly precedes p means that every time property p holds, property q must hold in a previous state;
- q responds to p means that every time property p holds, 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 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 $e_P \le b_Q$ and P occurs after Q if $b_P \ge e_Q$.

For each state s and for each CP $P = T(p_1 \ldots, p_n)$ that holds at 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, $b_P(s) = e_P(s) = s$.
- For the CP class $P = AtLeastOne_E(p_1, \ldots, p_n)$ that holds at state s, $b_P(s) = e_P(s)$, i.e., 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, $b_P(s) = e_P(s) = s$.
- For the CP class $P = Parallel_E(p_1, \ldots, p_n)$ that holds at state s, $b_P(s) = e_P(s)$, i.e., 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, $b_P(s) = s$ and $e_P(s) = s + (n - 1)$.
- For the CP class $P = Consecutive_E(p_1, \ldots, p_n)$ that holds at state s, $b_P(s)$ is the first state $s' > s$ at which the proposition p_1 becomes true, and $e_P(s) = s' + (n - 1)$.
- For the CP class $P = Eventual_C(p_1, \ldots, p_n)$ that holds at state s_1, $b_P(s_1) = s_1$, and $e_P(s_1)$ is the first state $s_n > s_1$ 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, $b_P(s)$ is the first state s_1 at which the first proposition p_1 becomes true, and $e_P(s)$ is 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 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 $e_Q(s') \le b_P(s)$. 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 $b_Q(s') \ge e_P(s)$. 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.*

3.1.3. 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 1.2. We also provide semantics for these patterns.

We start by providing formal definitions of scopes that use CP as their parameters. These definitions use the notions of beginning and ending states as defined in Section 3.2.

- 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 $b_R(s_f)$.

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

- 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*").

4. Need for New Operators

To describe LTL templates for patterns and scopes with CP classes, we need to define new "and" operators that will be used to simplify the specification of the LTL templates described in Section 5. While it is still possible to define the LTL templates without the use of these new operators, their use will result in a significantly shorter and more understandable templates.

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 an LTL formula *A* holds at state *s* if some "subformula" of *A* holds 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 define 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.

In the following text, we 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 "and-ing" two CP classes).

4.1. The New Operator "$\&_r$"

Generally, in logic, 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.

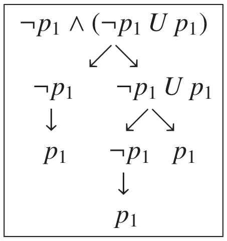
- *A formula P is its own subformula.*

- *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 formally define the meaning of $P \,\&_r\, Q$. 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 the 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

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

4.2. The New Operators “$\&_{-l}$” and “$\&_l$”

While we provided a general definition of the operator $\&_r$, we only define the two new operators “$\&_{-1}$” and “$\&_l$”in terms of CP classes. In other words, 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*

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

Table 3: Template LTL Formulas for Patterns Within *Global* Scope

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

5. General LTL Formulas for Patterns and Scopes With CP

The previous sections laid the foundation for defining high-level LTL templates that can be used to define LTL specifications for all pattern/scope/CP combinations. We provided a clear description of the meaning of each CP class, as well as the meaning of pattern and scopes that use CP classes to define their parameters. We also defined new LTL operators to simplify the LTL templates. This section provides the high-level LTL Templates for pattern/scope combinations defined using CP classes. 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.

It is important to note that in previous work [18] we modified some of the original LTL formulas for patterns and scopes provided by the SPS website. The new LTL formulas were shown to be more efficient for use with model checking. The modified LTL formulas (which were shown to be equivalent to the original SPS formulas) generate fewer states in the Buchi automaton used by model checkers such as Spin. This reduction in the number of states result in faster and more efficient model checking of systems.

5.1. Formulas for Patterns Within Global and Before R Scopes

Tables 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 indicate 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.

Finally, note that there are two LTL templates for $Q\ Precedes\ P_C$, and the $Q\ Precedes\ P_E$ pattern in Table 3. The first of these formulas (annotated with the superscript *) Q is of type $AtLeastOne_C$ or $Parallel_C$, while in the other template (annotated with the superscript +) Q is of types other than $AtLeastOne_C$ or $Parallel_C$.

5.2. 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 $\mathcal{P}_{<R}^{LTL}$ 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.

Table 4: LTL Templates for Patterns Within $Before\ R$ Scope

Pattern	LTL Formula)
$Absence\ of\ P\ Before\ R_C$	$\neg((\neg R^{LTL})\ U\ ((P^{LTL} \&_r \neg R^{LTL}) \&_l \Diamond R^{LTL}))$
$Absence\ of\ P\ Before\ R_E$	$(\Diamond R^{LTL}) \rightarrow \neg((\neg((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X(R_H^{LTL})))\ U\ (P^{LTL} \&_r \neg R_H^{LTL}))$
$Existence\ of\ P\ Before\ R_C$	$\neg((\neg(P^{LTL} \&_r \neg R^{LTL}))\ U\ R^{LTL})$
$Existence\ of\ P\ Before\ R_E$	$(\Diamond R^{LTL}) \rightarrow ((\neg((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X(R_H^{LTL})))\ U\ (P^{LTL} \&_r \neg R_H^{LTL}))$
$Q\ Precedes\ P_C\ Before\ R_C$	$(\Diamond R^{LTL}) \rightarrow ((\neg(P^{LTL} \&_r \neg R^{LTL}))\ U\ ((Q^{LTL} \&_{-l} \neg P^{LTL}) \vee R^{LTL}))$
$Q\ Precedes\ P_E\ Before\ R_C$	$(\Diamond R^{LTL}) \rightarrow ((\neg((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg R^{LTL} \wedge X(P_H^{LTL} \&_r \neg R^{LTL})))\ U\ ((Q^{LTL} \&_{-l} \neg(P_H^{LTL})) \vee R^{LTL}))$
$Q\ Precedes\ P_C\ Before\ R_E$	$(\Diamond R^{LTL}) \rightarrow (((\neg(P^{LTL} \&_r \neg R_H^{LTL}))\ U\ ((Q^{LTL} \&_{-l} \neg P^{LTL}) \vee ((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X R_H^{LTL}))))$
$Q\ Precedes\ P_E\ Before\ R_E$	$(\Diamond R^{LTL}) \rightarrow ((\neg((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg R_H^{LTL} \wedge X(P_H^{LTL} \&_r \neg R_H^{LTL})))\ U\ ((Q^{LTL} \&_{-l} \neg P_H^{LTL}) \vee ((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X R_H^{LTL})))$
$Q\ Strictly\ Precedes\ P_C\ Before\ R_C$	$(\Diamond R^{LTL}) \rightarrow ((\neg(P^{LTL} \&_r \neg R^{LTL}))\ U\ ((Q^{LTL} \&_r \neg P^{LTL}) \vee R^{LTL}))$
$Q\ Strictly\ Precedes\ P_E\ Before\ R_C$	$(\Diamond R^{LTL}) \rightarrow ((\neg((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg R^{LTL} \wedge X(P_H^{LTL} \&_r \neg R^{LTL})))\ U\ ((Q^{LTL} \&_r \neg(P_H^{LTL})) \vee R^{LTL}))$
$Q\ Strictly\ Precedes\ P_C\ Before\ R_E$	$(\Diamond R^{LTL}) \rightarrow (((\neg(P^{LTL} \&_r \neg R_H^{LTL}))\ U\ ((Q^{LTL} \&_r \neg P^{LTL}) \vee ((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X R_H^{LTL}))))$
$Q\ Strictly\ Precedes\ P_E\ Before\ R_E$	$(\Diamond R^{LTL}) \rightarrow ((\neg((\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge \neg R_H^{LTL} \wedge X(P_H^{LTL} \&_r \neg R_H^{LTL})))\ U\ ((Q^{LTL} \&_r \neg P_H^{LTL}) \vee ((\neg r_1 \wedge \ldots \wedge \neg r_n) \wedge X R_H^{LTL})))$
$Q\ Responds\ to\ P\ Before\ R_C$	$\neg((\neg R^{LTL})\ U\ ((P^{LTL} \&_r \neg R^{LTL}) \&_l ((\neg(Q^{LTL} \&_r \neg R^{LTL}))\ U\ R^{LTL})))$
$Q\ Responds\ to\ P\ Before\ R_E$	$\neg((\neg((\neg r_1 \wedge \neg r_2 \wedge \ldots \wedge \neg r_n) \wedge X(R_H^{LTL})))\ U\ ((P^{LTL} \&_r \neg R_H^{LTL}) \&_l ((\neg(Q^{LTL} \&_r \neg R_H^{LTL}))\ U\ R_H^{LTL})))$

Table 5: LTL Templates for Patterns Within the Remaining Scopes

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

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 $\mathcal{P}^{LTL}_{<R}$). 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 ($\mathcal{P}^{LTL}_{<R}$) for *Response* (P,Q) *Before R* 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 $\mathcal{P}^{LTL}_{<R}$ 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:

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

or

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

6. Validation of LTL Templates

This section describes the validation of the defined LTL templates for all pattern, scope, and CP classes combinations. We use formal proofs to validate the correctness of the templates for patterns within the *Global* scope. We introduce a novel approach for testing the LTL templates for the remaining patterns using model checking. The smaller number of templates for the *Global* scope was the motivating factor in formally proving the correctness of these templates. On the other hand the larger number of templates for the remaining scopes meant there was a need for some form of automated validation technique through testing.

6.1. Formal Proofs of Correctness of Patterns Within Global Scope

As mentioned above, the formulas for patterns within the *Global* scope were verified using formal proofs. The proofs used the definitions of patterns within the *Global* scope provided in Section 3.

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 3. In this section, we only show the proof for *Q Responds* to *P*. The remaining proofs can be found in Appendix A. Also note that we consider the proofs of correctness of the LTL templates for the *Existence* and *Absence* patterns are straightforward and as a result are not shown in this work.

6.1.1. Theorem 1.1: 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 4, "*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:

$$\forall s\, (P(s) \rightarrow \exists s'\, (Q(s') \wedge b_Q(s') \geq e_P(s))) \tag{3}$$

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

$$\Box(P \rightarrow (P\ \&_l \diamond Q)) \tag{4}$$

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 (4) 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 (4) is equivalent to

$$\forall s\, (P(s) \rightarrow (P\ \&_l \diamond Q)(s)) \tag{5}$$

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 (4) can be equivalently reformulated as

$$\forall s\, (P(s) \rightarrow (P(s) \wedge (\diamond Q)(e_P(s)))). \tag{6}$$

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

$$\forall s\, (P(s) \rightarrow (\diamond Q)(e_P(s))). \tag{7}$$

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 (4) is equivalent to

$$\forall s\, (P(s) \rightarrow \exists s''\, (Q(s'') \wedge s'' \geq e_P(s))). \tag{8}$$

3°. Since the LTL formula (4) is equivalent to (8), to complete our proof we only need to prove the equivalence between (3) and (8).

3.1°. Let us first prove that (8) implies (3).

Indeed, let us assume that (8) holds, and that $P(s)$ holds for some moment of time s. Then, the formula (8) 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 (3).

3.2°. Let us now prove that (3) implies (8).

Indeed, assume that (3) holds, and $P(s)$ holds for some moment of time s. Then, according to (3), 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 (8).

The equivalence is proven and hence Theorem 1.1 is proven.

We use the same approach of using first order logic to prove the correctness of the remaining seven templates in Table 3. The remaining proofs are available in Appendix B

6.2. Model Checking-Based Testing of Templates

This section describes a novel approach to validating LTL formulas using model checking for the remaining scopes. While formal proofs were used to validate templates for the *Global* scope, this model checking approach was used to validate the remaining templates. We first used this approach in [19] to validate the correctness of the original LTL formulas for patterns and scope combinations (with no CP) as defined by SPS and the Prospec tool [12]. Using the approach we were able to discover some discrepancies in these original formulas[19]. This new approach has also been used in the teaching of formal specifications and LTL [22, 23].

Model checking is a formal technique for verifying finite-state concurrent systems by examining the consistency of the system against system specifications for all possible executions of the system. The process of model checking consists of three tasks: modeling, specification, and verification.

Modeling. The modeling phase consists of converting the design into a formalism accepted by the model checker. In some cases, modeling is simply compiling the source code representing the design. In most cases, however, the limits of time and memory mean that additional abstraction is required to come up with a model that ignores irrelevant details. In SPIN, the model is written in the Promela language [7].

Specification. As part of model checking a system, it is necessary to specify the system properties to be checked. Properties are usually expressed in a temporal logic. The use of temporal logic allows for reasoning about time, which becomes important in the case of reactive systems. In model checking, specifications are used to verify that the system satisfies the behavior expressed by the property.

Verification. Once the system model and properties are specified, the model checker verifies the consistency of the model and specification. The model checker relies on building a finite model of the system and then traversing the system model to verify that the specified properties hold in every execution of the model [2, 7]. If there is an inconsistency between the model and the property being verified, a counter example, in form of execution trace, is provided to assist in identifying the source of the error. Figure 1 shows the process of model checking.

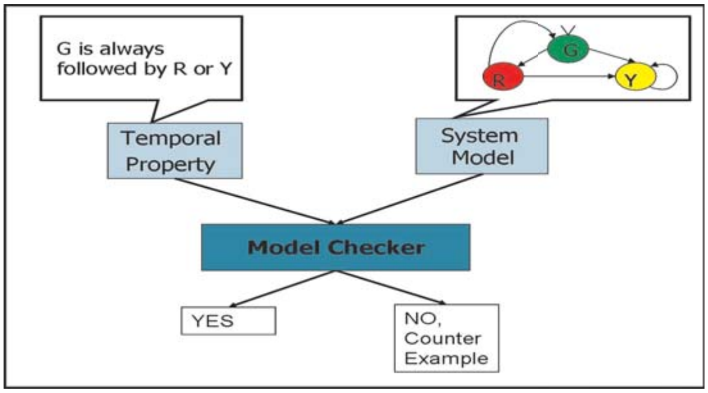


Figure 1: Model checking process.

6.2.1. *Overview*

As opposed to using a model checker like SPIN[2] to test the correctness of the model, the technique described in this section uses a simple model to test whether an LTL specification holds for a given trace of computation. A *trace of computation* is a sequence of states that depicts the propositions that hold in each state. In this technique, the model produces a simple finite state automaton with exactly one possible execution and a small number of states.

The user models a trace of computation by assigning truth values to the propositions of the LTL formula for a particular state. For example, a user may examine one or more combinations of the following: a proposition holds in the first state, a proposition holds in the last state, a proposition holds in multiple states, a proposition holds in one state and not the next, an interval (scope) is built, an interval is not built, and nested intervals exist. This assignment of values is referred to as a test. The user runs SPIN using the Promela code, the test case, and the LTL specification. Each run assists the user in validating a formula by checking expected results against actual results. The simplicity of the model makes inspection of the result feasible.

6.2.2. *Steps for Model Checking-Based Testing*

The Promela code consists of a do-loop that begins with the initial value of i set to zero, and terminates when i reaches a predefined value called *limit*. Setting *limit* to the value 20 means that the model has a total of 20 states starting with the state i_0 and ending with the state i_{19}. The Promela code is given in Figure 2.

The steps for applying the technique are as follows:

1) Insert the simple Promela model into the model checker.

2) Specify the LTL formula to be tested.

3) Use conditions to assign the states in which propositions from the LTL formula are set to true.

[2] In this work we use SPIN as the model checker of choice. However the approach works well with any other model checker including SMV and NuSMV

```
#define limit 20
byte i = 0;
active proctype seq( ) {
    do
      :: (i < limit) → i = i+1;
      :: (i == limit) → break;
    od; }
```

Figure 2: Promela Code of the Simple Model

The Promela code remains the same in all test cases. In the third step above, the values of the propositions are set by assigning each proposition a truth value based on the variable *i* in the Promela model. For example, in Test 1 shown in Figure 3, *P* is true in the fifth state, and *R* is true in the eighth state. To assert *P* in the fifth state, we set *P* to be the truth value of the condition (i == 4) (note that the model starts with *i* set to zero). Similarly, we define *R* by the truth value of the condition (i == 7). Note that for this test case, *P* and *R* are only true in the fifth and eighth states respectively. However, it is possible to specify for a proposition to be true in more than one state using the "∨" "or" operator as is the case for *R* in Test 2 in Figure 3.

The complete list of test suites for the patterns and scope is available in [21]. The test suite consists of a set of test cases for each scope, where a test case is a set of assignments for the propositions in the LTL formula under test, such that a particular trace of computation is created. In the following trace of computation: "- - - Q - - P - - - - R - - - - - - - -"
Q is true in the fourth state, *P* is true in the seventh state, and *R* is true in the twelfth state. Note that each character in the string represents a state, and a dash (-) implies that none of the propositions is *true* at that state. A letter symbol, e.g., *Q*, *P*, and *R* in this example, denotes that the proposition is *true* in the designated state. Displaying more than one letter between parentheses implies that the propositions represented by the letters are valid at that state. For instance, (*PQ*) denotes that *P* and *Q* both hold in the same state. It is worth mentioning that the test cases were selected using the equivalence classes and boundary value analysis testing strategies based on the patterns and scopes.

Test 1 : - - - - - P - - R - - - - - - - - - - - - -	Test 2 : - - - - - - R - - P - - R - - - - - - - - -
Pattern: Existence of P	Pattern: Existence of P
Scope: Before R	Scope: Before R
Formula: $(\diamond R) \rightarrow ((\neg R) U (P \wedge \neg R))$	Formula: $(\diamond R) \rightarrow ((\neg R) U (P \wedge \neg R))$
P: (i == 5)	P: (i == 8)
R: (i == 8)	R: (i == 5 ∨ i == 11)
Expected Result: No violation	Expected Result: Violation

Figure 3: Sample Test Cases

6.2.3. Results of Testing

Vela [25] implemented a module in Prospec using the LTL templates to generate a complete set of LTL formulas for all pattern, scope, and CP combinations. The LTL templates can generate LTL formulas for over 34,000 different combinations. Munoz [14] used the model checking-based testing approach to check that the intended LTL formulas were generated. The work in [14] used the equivalence classes and boundary value analysis testing approaches[5] to generate represented test cases as traces of computations.

Equivalence class analysis partitions possible inputs into classes from which exemplars can be selected. This approach assumes that an input from a partition is equally likely to expose an error as any other input from that partition. For example, the inputs representing the following traces of computation, "L - - - P - - -" and "L - - - - - - P - -", belong to the same equivalence class for testing the generated LTL formula for $Existence of P After L$; as a result, only one will need to be tested.

Boundary analysis, on the other hand, works by selecting test cases to test input limits. For each limit, test cases are created to execute a value immediately to the left of that limit, a value immediately to the right of that limit, and a value that is exactly at that limit. For example, to test the $Existence of P Before R$, we ran the following traces "- - R P - - - -", "- - P R - - - -", and "- - (R P) - - - -".

The automated test generation algorithm in [14] generated over 3.8 million test cases. Of all the test cases ran to test the implementation of the LTL templates, 98% matched the expected results. All the test cases that failed, did so because of mismatched open/close parenthesis pairs in the implementation of the templates [14].

7. Summary and Future Work

The use of formal methods in software development has shown potential in increasing the dependability of the developed systems. With the increased use of formal verification techniques in the software development, it is important that software engineers are able to automatically generate complex formal specifications because of the complexity of defining formal specifications manually. Furthermore, it is imperative that they have confidence that the formal specifications accurately reflect the intended meaning of their properties. SPS and Prospec are example tools that automatically generate LTL formulas for use in formal verification tools such as the Spin model checker.

This paper provides formal descriptions of complex formulas that use composite propositions. In addition, it presents formal descriptions of the patterns and scopes defined by Dwyer et al. [3] when using CP classes. Specifically, the paper provides the following:

- LTL templates that can be used to generate LTL specifications of properties defined by patterns, scopes, and CP classes. These templates have been used in the generation of LTL formulas as part of the new Prospec 2.0 tool [9, 25].

- General technique for using first order logic to prove correctness of LTL specifications.

- Novel approach for using model checking for the validation of LTL formulas against user's (specifier) initial intent. This new approach was used to validate the correctness of the LTL templates and the corresponding LTL formulas [14].

As part of the future work, we intend to provide some form of graphical presentations of the test cases used to validated the defined templates. We believe that this will allow the users of Prospec to validate that the LTL formulas generated by Prospec using the above mentioned templates do indeed match the original property. Finally, we plan on using the same approach to define and validate abstract templates for Computation Tree Logic (CTL), which is another commonly used language for specifying software properties.

References

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

[2] Clarke, E., Grumberg, O., and D. Peled. "Model Checking". MIT Publishers, 1999

[3] Dwyer, M. B., G. S. Avrunin, and J. C. Corbett, "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 Mandrioli, D., "Foundamentals of Software Engineering". Prentice Hall, 2003

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

[9] Gallegos, I., Ochoa, O., Gates, A., Roach, S., Salamah, S., and Vela, C., "A Property Specification Tool for Generating Formal Specifications: Prospec 2.0" *In the Proceedings of the Software Engineering and Knowledge Engineering Conference*, San Francisco, California, July, 2008, 273-278

[10] MacKenzie, D. M., Designing "Safe Software for Medical Devices," *Proceedings of the 21st international conference on Software engineering*, 1999, 618.

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

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

[13] Mondragon, O. and A. Q. Gates, "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," *Intl. Journal Software Engineering and Knowledge Engineering, 14(1), Feb. 2004.*

[14] Munoz, C., and Roach, S., "Automated Testing of LTL Formula Generation by Prospec", *In the proceedings of the IEEE 12th International Symposium on High Assurance Systems Engineering, Nov, 2010.*

[15] National Institute of Standards and Technology (NIST), June 02, "http://www.nist.gov/public-affairs/releases/n02-10.htm"

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

[17] The Economic Impacts of Inadequate Infrastructure for Software Testing," Healt, Social, and Economic Research, Project No. 7007,001, Resaerch Triangle Park, NC 27709, 2002

[18] Salamah, S., Gates, A., and Roach, S., "Towards Support for Software Model Checking: Improving the Efficiency of Formal Specifications", *Journal of Advances of Software Engineering, Volume 2011.*

[19] Salamah, S., Gates, A., Roach, S., and Mondragon, O., "Verifying Pattern-Generated LTL Formulas: A Case Study." *In the 12th International SPIN Workshop, Aug. 2005.*

[20] 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.

[21] Salamah, S., "Supporting Documentation for Dissertation Work,", the University of Texas at El Paso, May 2007.

[22] Salamah, S., and Gates, A.,"A Technique for Using Model Checkers to Teach Formal Specifications" *In Proceedings of the 21st IEEE-CS International Conference on Software Engineering Education and Training (CSEET), Charleston, SC, April 2008, 181-188*

[23] Salamah, S., Ochoa, O., Gates, A., "A Comparative Study of a Tool-Based Approach to Teaching Formal Specifications", *In the Proceedings of the 39th Annual Frontiers in Education (FIE) Conference, Oct, 2010*

[24] Stolz, V. and E. Bodden, "Temporal Assertions using AspectJ", *In the Proceedings of the Fifth Workshop on Runtime Verification, July 2005.*

[25] Vela, C. Y. An Implementation for Automatic Generation of Linear Temporal Logic Formulas. Master's Project, Department of Computer Science, The University of Texas at El Paso, 2009.

[26] Zhi-Wei Lin, "Network OAM Requirements for the New York City Transit Network," IEEE Communications Magazine, October, 2004, 112-116

Appendix A. Remaining Proofs

In this Appendix we provide the proofs of correctness of the LTL Templates in Table 3.

Theorem 1.2:
The LTL formula

$$\text{“}\neg((\neg(Q^{LTL}\ \&_r\ \neg P^{LTL}))\ U\ P^{LTL})\text{”}$$

is equivalent to the formal definition of the pattern "*Q Strictly Precedes P*" in *Global* scope for *P* of type Condition.

Proof:

1°. According to Definition 10, "*Q Strictly Precedes P*" means that if *P* holds at some moment *t*, then *Q* also holds at some moment t' for which $e_Q(t') < b_P(t)$. Formally, we can describe this property as follows:

$$\forall s\,(P(s) \rightarrow \exists s'\,(Q(s') \wedge e_Q(s') < b_P(s))) \tag{A.1}$$

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

$$\neg((\neg(Q\ \&_r\ \neg P))\ U\ P) \tag{A.2}$$

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

2.1°. The LTL formula (A.2) is a negation of the expression

$$(\neg(Q\ \&_r\ \neg P))\ U\ P. \tag{A.3}$$

By the definition of the "until" operator U, the formula $A\ U\ B$ holds at moment 0 if there exists a moment of time s such that $B(s)$ holds at this moment of time, and A is true for all previous moments of time.

So, the auxiliary expression (A.3) means that there exists a moment s such that $P(s)$ is true and $\neg(Q\ \&_r\ \neg P)$ holds for all the previous moments of time $s'' < s$, i.e., that

$$\exists s\,(P(s) \wedge \forall s'' < s\ \neg(Q\ \&_r\ \neg P)(s'')). \tag{A.4}$$

2.2°. We have shown that the auxiliary expression (A.3) is equivalent to the formula (A.4). The LTL formula (A.2) is equivalent to the negation of the auxiliary expression (A.3), hence it is equivalent to the negation of the formula (A.4).

If we use de Morgan rules to move negation inside the formula, we conclude that the LTL formula (A.2) is equivalent to the formula

$$\forall s\,(P(s) \to \exists s'' < s\,(Q\,\&_r\,\neg P)(s'')). \tag{A.5}$$

2.3°. Since the LTL formula (A.2) is equivalent to (A.5), to complete our proof we only need to prove the equivalence between (A.1) and (A.5).

3°. Let us first prove that both in the formula (A.1) and in the formula (A.5), instead of a universal quantifier over s, it is sufficient to only consider the first moment of time s_P at which P becomes true. In other words, we will prove that the formula (A.1) is equivalent to

$$\exists s'\,(Q(s') \wedge e_Q(s') < b_P(s_P)), \tag{A.6}$$

and the formula (A.5) is equivalent to

$$\exists s'' < s_P\,(Q\,\&_r\,\neg P)(s''). \tag{A.7}$$

3.1°. Let us first prove that the formula (A.1) is equivalent to (A.6).

The fact that (A.1) implies (A.6) is straightforward. Indeed, since the implication (A.1) holds for all moment of time s at which P holds, it should also hold for the first moment of time s_P when P is true. Thus, (A.1) indeed implies (A.6).

Vice versa, let us assume (A.6) holds, i.e., that there exists a moment s' for which $Q(s')$ holds and $e_Q(s') < b_P(s_P)$. Let s be any moment of time at which P holds. Let us show that for this new moment of time s, we can take the same moment s' and we will have $Q(s')$ holds and $e_Q(s') < b_P(s)$.

By our choice of s', we have $Q(s')$, so the only thing that remains to prove is that $e_Q(s') < b_P(s)$. By definition of s_P as the first moment of time at which P holds, we conclude that $s_P \leq s$. Thus, we have $b_P(s_P) \leq b_P(s)$, so from $e_Q(s') < b_P(s_P)$ we can conclude that $e_Q(s') < b_P(s_P) \leq b_P(s)$ and $e_Q(s') < b_P(s)$. The statement is proven.

3.2°. Similarly, we can prove that the formula (A.5) is equivalent to (A.7).

4°. So, to prove our result, it is sufficient to prove that the formula (A.6) is equivalent to (A.7).

4.1°. Let us first prove that (A.6) implies (A.7).

Indeed, let us assume that the formula (A.6) is true, i.e., that there exists a s' for which $Q(s')$ and $e_Q(s') < b_P(s_P)$. Since the composite proposition P is of type condition, it is true at the moment $b_P(s_P)$, so $b_P(s_P) = s_P$. Since $e_Q(s') < b_P(s_P) = s_P$, all Q-relevant moments of time, i.e., all moments of time between s' and $e_Q(s')$, occur before s_P. We know that s_P is the first moment of time at which P holds, so in all previous moments of time, P is false.

Thus, we can conclude that P is false at all Q-relevant moments of time. Let us now show that (A.7) holds for $s'' = s'$. Indeed, by definition of the new connective $\&_r$, the expression $Q\,\&_r\,\neg P$ hold at a moment s' if Q holds at this moment s' (which is true), and $\neg P$ holds at all Q-relevant moments of time, i.e., at all moments of time between s' and $e_Q(s')$. So, (A.7) is indeed true.

4.2°. Let us now prove that (A.7) implies (A.6).

Let us assume that the formula (A.7) is true, i.e., that there exists a $s'' < s_P$ for which $(Q\,\&_r\,\neg P)(s'')$, i.e., for which $Q(s'')$ is true, and $\neg P$ holds for all Q-relevant moments of time, i.e., for all moments of time between s'' and $e_Q(s'')$.

By definition, s_P is the first moment of time at which P holds, so P is false at all moments of time $s < s_P$. In particular, since $s'' < s_P$, P is false at all moments of time $s \leq s''$. We have also shown that P is false at all moments of time between s'' and $e_Q(s'')$. Thus, P is false at all moments of time $s \leq e_Q(s'')$. Since P is true at the moment s_P, this means that this moment s_P cannot precede or be equal to $e_Q(s'')$; thus, $e_Q(s'') < s_P$.

We have already shown that $s_P = b_P(s_P)$, hence $e_Q(s'') < b_P(s_P)$. So, we have a moment s'' at which $Q(s'')$ and $e_Q(s'') < b_P(s_P)$. In other words, for this moment s'' as s', the formula (A.6) holds.

So, (A.7) implies (A.6), and thus, these formulas are indeed equivalent. The equivalence is proven and hence Theorem 1.2 is proven.

Theorem 1.3:
The LTL formula

$$\text{“}\neg((\neg(Q^{LTL}\,\&_r\,\neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge X\,P_H^{LTL})))\,U\,(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge X\,P_H^{LTL}))\text{”}$$

is equivalent to the formal definition of the pattern “*Q Strictly Precedes P*” in *Global* scope for P of type event.

Proof:

1°. Let us prove that for the case when P is of type event, the formula “*Q Strictly Precedes P*” (expressed by the formula (A.1)) is equivalent to the corresponding LTL formula

$$\neg((\neg(Q\,\&_r\,\neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})))\,U$$
$$(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})) \tag{A.8}$$

2°. To prove the desired equivalence, let us first reformulate the LTL formula (A.8) in terms of quantifiers.

2.1°. The LTL formula (A.8) is a negation of the expression

$$(\neg(Q\,\&_r\,\neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))\,U$$
$$(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})). \tag{A.9}$$

By the definition of the “until” operator U, the formula $A\,U\,B$ holds at moment 0 if there exists a moment of time s such that $B(s)$ holds at this moment of time, and A is true for all previous moments of time.

So, the auxiliary expression (A.9) means that there exists a moment s such that

$$\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s) \tag{A.10}$$

is true and

$$\neg(Q \,\&_r\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})) \quad (A.11)$$

holds for all the previous moments of time $s'' < s$, i.e., that

$$\exists s\,(\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s) \wedge$$

$$\forall s'' < s\, \neg(Q \,\&_r\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))(s'')). \quad (A.12)$$

2.2°. We have shown that the auxiliary expression (A.9) is equivalent to the formula (A.12). The LTL formula (A.8) is equivalent to the negation of the auxiliary expression (A.9), hence it is equivalent to the negation of the formula (A.12).

If we use de Morgan rules to move negation inside the formula, we conclude that the LTL formula (A.8) is equivalent to the formula

$$\forall s\,((\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s)) \rightarrow$$

$$\exists s'' < s\,(Q \,\&_r\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))(s'')). \quad (A.13)$$

2.3°. Since the LTL formula (A.8) is equivalent to (A.13), to complete our proof we only need to prove the equivalence between (A.1) and (A.13).

3°. Let us first prove that both in the formula (A.1) and in the formula (A.13), instead of a universal quantifier over s, it is sufficient to only consider the first moment of time s_P at which the formula (A.10) becomes true. In other words, we will prove that the formula (A.1) is equivalent to

$$\exists s'\,(Q(s') \wedge e_Q(s') < b_P(s_P)), \quad (A.14)$$

and the formula (A.13) is equivalent to

$$\exists s'' < s_P\,(Q \,\&_r\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))(s''). \quad (A.15)$$

3.1°. Let us first prove that the formula (A.1) is equivalent to (A.14).

Similarly to the previous proof (of the case when P is of type condition), we can prove that in the formula (A.1) it is sufficient to consider the first moment of time f_P at which P becomes true. In other words, we can prove that the formula (A.1) is equivalent to the formula

$$\exists s'\,(Q(s') \wedge e_Q(s') < b_P(f_P)), \quad (A.16)$$

According to the definition of an event P and of the beginning moment of time b_P, the formula P holds at a moment s if at some moment $b_P(s) \geq s$, we have

$$(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})(b_P(s)) \quad (A.17)$$

and all single propositions p_i are false for all the moments of time between s and $b_P(s)$. If P holds at s, then this property P is true for all moments s' between s and $b_P(s)$; for all these moments s', the starting point $b_P(s')$ is the same – the moment $b_P(s)$.

So, for $s = f_P$, the moment $b_P(f_P)$ coincides with the first moment of time s_P when the expression (A.10) becomes true. For this moment, $b_P(s_P) = s_P$. Thus, the formula (A.1) indeed implies (A.14), and, moreover, $b_P(s_P) = s_P$.

Vice versa, let us assume that (A.14) holds, i.e., there exists a moment s' for which $Q(s')$ and $e_Q(s') < b_P(s_P)$. We have already proven that $b_P(s_P) = b_P(f_P)$, so we have $e_Q(s') < b_P(f_P)$.

Let s be any moment of time at which P holds. Let us show that for this new moment of time s, we can take the same moment s' and we will have $Q(s')$ holds and $e_Q(s') < b_P(s)$.

By our choice of s', we have $Q(s')$, so the only thing that remains to prove is that $e_Q(s') < b_P(s)$. By definition of f_P as the first moment of time at which P holds, we conclude that $f_P \leq s$. Thus, we have $b_P(f_P) \leq b_P(s)$, so from $e_Q(s') < b_P(f_P)$ we can conclude that $e_Q(s') < b_P(f_P) \leq b_P(s)$ and $e_Q(s') < b_P(s)$. The statement is proven.

3.2°. Similarly, we can prove that the formula (A.13) is equivalent to (A.15).

4°. So, to prove our result, it is sufficient to prove that the formula (A.14) is equivalent to (A.15).

4.1°. Let us first prove that (A.14) implies (A.15).

Indeed, let us assume that the formula (A.14) is true, i.e., that there exists a s' for which $Q(s')$ and $e_Q(s') < b_P(s_P)$. We have already mentioned that we have $b_P(s_P) = s_P$. Since $e_Q(s') < b_P(s_P) = s_P$, all Q-relevant moments of time, i.e., all moments of time between s' and $e_Q(s')$, occur before s_P. We know that s_P is the first moment of time at which (A.10) holds, so in all previous moments of time, (A.10) is false.

Thus, we can conclude that (A.10) is false at all Q-relevant moments of time. Let us now show that (A.15) holds for $s'' = s'$. Indeed, by definition of the connective $\&_r$, the expression

$$Q \,\&_r\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}) \quad (A.18)$$

holds at a moment s' if Q holds at this moment s' (which is true), and the negation of (A.10) holds at all Q-relevant moments of time, i.e., at all moments of time between s' and $e_Q(s')$. So, (A.15) is indeed true.

4.2°. Let us now prove that (A.15) implies (A.14).

Let us assume that the formula (A.15) is true, i.e., that there exists a $s'' < s_P$ for which $Q(s'')$ is true, and the negation of (A.10) holds for all Q-relevant moments of time, i.e., for all moments of time between s'' and $e_Q(s'')$.

By definition, s_P is the first moment of time at which (A.10) holds, so (A.10) is false at all moments of time $s < s_P$. In particular, since $s'' < s_P$, (A.10) is false at all moments of time $s \leq s''$. We have also shown that (A.10) is false at all moments of time between s'' and $e_Q(s'')$. Thus, (A.10) is false at all moments of time $s \leq e_Q(s'')$. Since (A.10) is true at the moment s_P, this means that this moment s_P cannot precede or be equal to $e_Q(s'')$; thus, $e_Q(s'') < s_P$.

Since $b_P(s_P) = s_P$, we thus have $Q(s'')$ and $e_Q(s'') < b_P(s_P)$. In other words, for this moment s'' as s', the formula (A.14) holds.

So, (A.15) implies (A.14), and thus, these formulas are indeed equivalent. The equivalence is proven and hence Theorem 1.3 is proven.

Theorem 1.4:
The LTL formula

$$\text{“}\neg((\neg Q^{LTL})\,U\,(P^{LTL} \wedge \neg Q^{LTL}))\text{”}$$

is equivalent to the formal definition of the pattern "*Q Precedes P*" in *Global* scope for P of type condition and Q is of type $AtLeastOne_C$ or $Parallel_C$.

Proof:

1°. According to Definition 11, "*Q Precedes P*" means that if P holds at some moment t, then Q also holds at some moment t' for which $e_Q(t') \leq b_P(t)$. Formally, we can describe this property as follows:

$$\forall s\, (P(s) \to \exists s'\, (Q(s') \wedge e_Q(s') \leq b_P(s))) \tag{A.19}$$

When Q is of type $AtLeastOne_C$ or $Parallel_C$, then $Q(s')$ means that $b_Q(s') = e_Q(s') = s'$. So, the property (A.19) is equivalent to the following simplified formula

$$\forall s\, (P(s) \to \exists s'\, (Q(s') \wedge s' \leq b_P(s))) \tag{A.20}$$

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

$$\neg((\neg Q)\, U\, (P \wedge \neg Q)) \tag{A.21}$$

2°. To prove the desired equivalence, let us first reformulate the LTL formula (A.21) in terms of quantifiers.

2.1°. The LTL formula (A.21) is a negation of the expression

$$(\neg Q)\, U\, (P \wedge \neg Q) \tag{A.22}$$

By the definition of the "until" operator U, the formula $A\, U\, B$ holds at moment 0 if there exists a moment of time s such that $B(s)$ holds at this moment of time, and A is true for all previous moments of time.

So, the auxiliary expression (A.22) means that there exists a moment s such that $P(s)$ is true, $\neg Q(s)$ is true, and $\neg Q$ holds for all the previous moments of time $s'' < s$, i.e., that

$$\exists s\, (P(s) \wedge \neg Q(s) \wedge \forall s'' < s\, \neg Q(s'')). \tag{A.23}$$

2.2°. We have shown that the auxiliary expression (A.22) is equivalent to the formula (A.23). The LTL formula (A.21) is equivalent to the negation of the auxiliary expression (A.22), hence it is equivalent to the negation of the formula (A.23).

If we use de Morgan rules to move negation inside the formula, we conclude that the LTL formula (A.21) is equivalent to the formula

$$\forall s\, (P(s) \to (Q(s) \vee \exists s'' < s\, Q(s''))). \tag{A.24}$$

The fact that Q should be true either for the moment s or for some moment $s'' < s$ can be described by saying that Q should be true for some moment $s'' \leq s$:

$$\forall s\, (P(s) \to \exists s'' \leq s\, Q(s'')). \tag{A.25}$$

2.3°. Since the LTL formula (A.21) is equivalent to (A.25), to complete our proof we only need to prove the equivalence between (A.20) and (A.25).

3°. Similarly to the case of "*Strictly Precedes*", one can prove that both in the formula (A.20) and in the formula (A.25), instead of a universal quantifier over s, it is sufficient to only consider the first moment of time s_P at which P becomes true. In other words, the formula (A.20) is equivalent to

$$\exists s'\, (Q(s') \wedge s' \leq b_P(s_P)), \tag{A.26}$$

and the formula (A.25) is equivalent to

$$\exists s'' \leq s_P\, Q(s''). \tag{A.27}$$

Indeed, since P is of type C, we have $b_P(s_P) = s_P$ and therefore, the formulas (A.26) and (A.27) are identical – and hence, equivalent. The equivalence is proven and hence Theorem 1.4 is proven.

Theorem 1.5:
The LTL formula

$$\text{“}\neg((\neg(Q^{LTL} \wedge \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge X\, P_H^{LTL})))\, U\, (\neg p_1 \wedge \ldots \wedge \neg p_n \wedge X\, P_H^{LTL} \wedge \neg Q^{LTL}))\text{”}$$

is equivalent to the formal definition of the pattern "*Q Precedes P*" in *Global* scope for P of type event and Q is of type $AtLeastOne_C$ or $Parallel_C$.

Proof:

1°. Let us prove that for the case when P is of type event, the formula "*Q Precedes P*" (expressed by the formula (A.20)) is equivalent to the corresponding LTL formula

$$\neg((\neg(Q \wedge \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})))\, U\, (\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL} \wedge \neg Q)) \tag{A.28}$$

2°. To prove the desired equivalence, let us first reformulate the LTL formula (A.28) in terms of quantifiers.

2.1°. The LTL formula (A.28) is a negation of the expression

$$((\neg(Q \wedge \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})))\, U\, (\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL} \wedge \neg Q)). \tag{A.29}$$

By the definition of the "until" operator U, the formula $A\, U\, B$ holds at moment 0 if there exists a moment of time s such that $B(s)$ holds at this moment of time, and A is true for all previous moments of time.

So, the auxiliary expression (A.29) means that there exists a moment s such that

$$\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s) \wedge \neg Q(s) \tag{A.30}$$

is true and

$$\neg(Q \wedge \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}) \wedge \neg Q(s'')) \tag{A.31}$$

holds for all the previous moments of time $s'' < s$, i.e., that

$$\exists s\, (\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s) \wedge \neg Q(s) \wedge$$
$$\forall s'' < s\, \neg(Q \wedge \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL} \wedge \neg Q))(s'')). \tag{A.32}$$

2.2°. We have shown that the auxiliary expression (A.29) is equivalent to the formula (A.32). The LTL formula (A.28) is equivalent to the negation of the auxiliary expression (A.29), hence it is equivalent to the negation of the formula (A.32).

If we use de Morgan rules to move negation inside the formula, we conclude that the LTL formula (A.28) is equivalent to the formula

$$\forall s\,((\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s)) \rightarrow$$

$$Q(s) \vee \exists s'' < s\,(Q(s'') \wedge \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))(s'')).\text{(A.33)}$$

If $Q(s)$ is true, then, since $(\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s))$ holds, we have

$$Q(s) \wedge \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})(s). \qquad \text{(A.34)}$$

Thus, in the formula (A.33), we can combine the cases $Q(s)$ and $s'' < s$ into a single case $s'' \leq s$:

$$\forall s\,((\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s)) \rightarrow$$

$$\exists s'' \leq s\,(Q(s'') \wedge \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))(s'')). \text{ (A.35)}$$

2.3°. Since the LTL formula (A.28) is equivalent to (A.35), to complete our proof we only need to prove the equivalence between (A.29) and (A.35).

For the case of "*Strictly Precedes*", we have proven a similar equivalence, with the only difference that there we had < instead of ≤. One can similarly show that the same equivalence holds for ≤ as well. The equivalence is proven and hence Theorem 1.5 is proven.

Theorem 1.6:
The LTL formula

The LTL formula "$\neg((\neg(Q^{LTL} \,\&_{-l}\, \neg P^{LTL}))\; U\; P^{LTL})$"

is equivalent to the formal definition of the pattern "*Q Precedes P*" in *Global* scope for P of type condition and Q is of types other than $AtLeastOne_C$ or $Parallel_C$.

Proof:

1°. We need to prove that the first order formula (A.19) is equivalent to the corresponding LTL formula

$$\neg((\neg(Q \,\&_{-l}\, \neg P))\; U\; P) \qquad \text{(A.36)}$$

2°. To prove the desired equivalence, let us first reformulate the LTL formula (A.36) in terms of quantifiers.

2.1°. The LTL formula (A.36) is a negation of the expression

$$(\neg(Q \,\&_{-l}\, \neg P))\; U\; P. \qquad \text{(A.37)}$$

By the definition of the "until" operator U, the formula $A\;U\;B$ holds at moment 0 if there exists a moment of time s such that $B(s)$ holds at this moment of time, and A is true for all previous moments of time.

So, the auxiliary expression (A.37) means that there exists a moment s such that $P(s)$ is true and $\neg(Q \,\&_{-l}\, \neg P)$ holds for all the previous moments of time $s'' < s$, i.e., that

$$\exists s\,(P(s) \wedge \forall s'' < s\, \neg(Q \,\&_{-l}\, \neg P)(s'')). \qquad \text{(A.38)}$$

2.2°. We have shown that the auxiliary expression (A.37) is equivalent to the formula (A.38). The LTL formula (A.36) is equivalent to the negation of the auxiliary expression (A.37), hence it is equivalent to the negation of the formula (A.38).

If we use de Morgan rules to move negation inside the formula, we conclude that the LTL formula (A.36) is equivalent to the formula

$$\forall s\,(P(s) \rightarrow \exists s'' < s\,(Q \,\&_{-l}\, \neg P)(s'')). \qquad \text{(A.39)}$$

2.3°. Since the LTL formula (A.36) is equivalent to (A.39), to complete our proof we only need to prove the equivalence between (A.19) and (A.39).

3°. Similarly to the case of "*Strictly Precedes*", we can prove that both in the formula (A.19) and in the formula (A.39), instead of a universal quantifier over s, it is sufficient to only consider the first moment of time s_P at which P becomes true. In other words, the formula (A.19) is equivalent to

$$\exists s'\,(Q(s') \wedge e_Q(s') \leq b_P(s_P)), \qquad \text{(A.40)}$$

and the formula (A.39) is equivalent to

$$\exists s'' < s_P\,(Q \,\&_{-l}\, \neg P)(s''). \qquad \text{(A.41)}$$

4°. So, to prove our result, it is sufficient to prove that the formula (A.40) is equivalent to (A.41).

4.1°. Let us first prove that (A.40) implies (A.41).

By definition of the new connective $\&_{-l}$, the property $A \,\&_{-l}\, B$ holds at the moment s if A holds at this moment s and B holds at all A-relevant moments of time with the possible exception of the last moment $e_A(s)$. In other words, B should hold at all moments of time s'' from s (included) to $e_A(s)$ (excluded): $s \leq s'' < e_A(s)$.

Let us assume that the formula (A.40) is true, i.e., that there exists a s' for which $Q(s')$ and $e_Q(s') \leq b_P(s_P)$. Since the composite proposition P is of type condition, it is true at the moment $b_P(s_P)$, so $b_P(s_P) = s_P$. Since $e_Q(s') \leq b_P(s_P) = s_P$, all moments of time s'' for which $s' \leq s'' < e_P(s')$, occur before s_P. We know that s_P is the first moment of time at which P holds, so in all previous moments of time, P is false.

Thus, we can conclude that Q is true at s' and that P is false at all moments of time t for which $s' \leq t < e_Q(s')$. By definition of $\&_{-l}$, this means that $Q \,\&_{-l}\, \neg P$ holds at the moment s'. So, (A.41) is indeed true.

4.2°. Let us now prove that (A.41) implies (A.40).

Let us assume that the formula (A.41) is true, i.e., that there exists a $s'' < s_P$ for which $(Q \,\&_{-l}\, \neg P)(s'')$, i.e., for which $Q(s'')$ is true, and $\neg P$ holds for all moments of time t for which $s'' \leq t < e_Q(s'')$.

By definition, s_P is the first moment of time at which P holds, so P is false at all moments of time $s < s_P$. In particular, since $s'' < s_P$, P is false at all moments of time $s \leq s''$. We have also shown that P is false at all moments of time t for which $s'' \leq t < e_Q(s'')$. Thus, P is false at all moments of time $s < e_Q(s'')$. Since P is true at the moment s_P, this means that this moment s_P cannot precede $e_Q(s'')$; thus, $e_Q(s'') \leq s_P$.

We have already shown that $s_P = b_P(s_P)$, hence $e_Q(s'') \le b_P(s_P)$. So, we have a moment s'' at which $Q(s'')$ and $e_Q(s'') \le b_P(s_P)$. In other words, for this moment s'' as s', the formula (A.40) holds.

So, (A.41) implies (A.40), and thus, these formulas are indeed equivalent. The equivalence is proven and hence Theorem 1.6 is proven.

Theorem 1.7:
The LTL formula

$$\text{“}\neg((\neg(Q^{LTL} \,\&_{-l}\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge X\, P_H^{LTL})))\; U\; (\neg p_1 \wedge \ldots \wedge \neg p_n \wedge X\, P_H^{LTL}))\text{”}$$

is equivalent to the formal definition of the pattern “*Q Precedes P*” in *Global* scope for P of type event and Q is of types other than $AtLeastOne_C$ or $Parallel_C$.

Proof:

1°. Let us prove that for the case when P is of type event, the formula “*Q Precedes P*” (expressed by the formula (A.19)) is equivalent to the corresponding LTL formula

$$\neg((\neg(Q \,\&_{-l}\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})))\; U$$
$$(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})) \tag{A.42}$$

2°. To prove the desired equivalence, let us first reformulate the LTL formula (A.42) in terms of quantifiers.

2.1°. The LTL formula (A.42) is a negation of the expression

$$(\neg(Q \,\&_{-l}\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})))\; U$$
$$(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}). \tag{A.43}$$

By the definition of the “until” operator U, the formula $A\,U\,B$ holds at moment 0 if there exists a moment of time s such that $B(s)$ holds at this moment of time, and A is true for all previous moments of time.

So, the auxiliary expression (A.43) means that there exists a moment s such that

$$\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s) \tag{A.44}$$

is true and

$$\neg(Q \,\&_{-l}\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL})) \tag{A.45}$$

holds for all the previous moments of time $s'' < s$, i.e., that

$$\exists s\, ((\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s)) \wedge$$
$$\forall s'' < s\, \neg(Q \,\&_{-l}\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))(s'')). \tag{A.46}$$

2.2°. We have shown that the auxiliary expression (A.43) is equivalent to the formula (A.46). The LTL formula (A.42) is equivalent to the negation of the auxiliary expression (A.43), hence it is equivalent to the negation of the formula (A.46).

If we use de Morgan rules to move negation inside the formula, we conclude that the LTL formula (A.42) is equivalent to the formula

$$\forall s\, ((\neg p_1(s) \wedge \ldots \wedge \neg p_n(s) \wedge (XP_H^{LTL})(s)) \rightarrow$$
$$\exists s'' < s\, (Q \,\&_{-l}\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))(s'')). \tag{A.47}$$

2.3°. Since the LTL formula (A.42) is equivalent to (A.47), to complete our proof we only need to prove the equivalence between (A.19) and (A.47).

3°. Similarly to the “*Strictly Precedes*” case, we can prove that both in the formula (A.19) and in the formula (A.47), instead of a universal quantifier over s, it is sufficient to only consider the first moment of time s_P at which the formula (A.44) becomes true. In other words, we will prove that the formula (A.19) is equivalent to

$$\exists s'\, (Q(s') \wedge e_Q(s') \le b_P(s_P)), \tag{A.48}$$

and the formula (A.47) is equivalent to

$$\exists s'' < s_P\, (Q \,\&_{-l}\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}))(s''). \tag{A.49}$$

4°. So, to prove our result, it is sufficient to prove that the formula (A.48) is equivalent to (A.49).

4.1°. Let us first prove that (A.48) implies (A.49).

Indeed, let us assume that the formula (A.48) is true, i.e., that there exists a s' for which $Q(s')$ and $e_Q(s') \le b_P(s_P)$. We have already mentioned that we have $b_P(s_P) = s_P$. Since $e_Q(s') \le b_P(s_P) = s_P$, all moments of time t for which $s' \le t < e_Q(s')$ occur before s_P. We know that s_P is the first moment of time at which (A.44) holds, so in all previous moments of time, (A.44) is false.

Let us now show that (A.49) holds for $s'' = s'$. Indeed, by definition of the connective $\&_{-l}$, the expression

$$Q \,\&_{-l}\, \neg(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge XP_H^{LTL}) \tag{A.50}$$

holds at a moment s' if Q holds at this moment s' (which is true), and the negation of (A.44) holds at all moments of time t for which $s' \le t < e_Q(s')$. So, (A.49) is indeed true.

4.2°. Let us now prove that (A.49) implies (A.48).

Let us assume that the formula (A.49) is true, i.e., that there exists a $s'' \le s_P$ for which $Q(s'')$ is true, and the negation of (A.44) holds for all moments of time t for which $s'' \le t < e_Q(s'')$.

By definition, s_P is the first moment of time at which (A.44) holds, so (A.44) is false at all moments of time $s < s_P$. In particular, since $s'' \le s_P$, (A.44) is false at all moments of time $s < s''$. We have also shown that (A.44) is false at all moments of time t for which $s'' \le t < e_Q(s'')$. Thus, (A.44) is false at all moments of time $s < e_Q(s'')$. Since (A.44) is true at the moment s_P, this means that this moment s_P cannot precede $e_Q(s'')$; thus, $e_Q(s'') \le s_P$.

Since $b_P(s_P) = s_P$, we thus have $Q(s'')$ and $e_Q(s'') \le b_P(s_P)$. In other words, for this moment s'' as s', the formula (A.48) holds.

So, (A.49) implies (A.48), and thus, these formulas are indeed equivalent. The equivalence is proven and hence Theorem 1.7 is proven.

By proving theorems 1.1 to 1.7 we have proven Theorem 1, which is the main Theorem in this paper.