550, rue Sherbrooke Ouest, bureau 100
Montréal (Québec) H3A 1B9
Tél. : (514) 840-1234;Téléc. : (514) 840-1244
888, rue St-Jean, bureau 555
Québec (Québec) G1R 5H6
Tél. : (418) 648-8080; téléc. : (418) 648-8141
http://www.crim.ca

CRIM - Documentation/Communications

Rapport technique

Adding Propositional Scopes to Linear Temporal Logic

CRIM- 05/05-06

May Haydar
Sergiy Boroday
Alexandre Petrenko
Houari Sahraoui

2005-05-24

Collection scientifique et technique

ISBN 2-89522-061-1

(Note facultative sur le verso de la page titre)

Pour tout renseignement, communiquer avec:
CRIM Centre de documentation
CRIM
550, rue Sherbrooke Ouest, bureau 100
Montréal (Québec) H3A 1B9

Téléphone : (514) 840-1234
Télécopieur : (514) 840-1244

Bibliothèque nationale du Québec
Bibliothèque nationale du Canada
ISBN 2-89522-061-1

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

We deal with the problem of property specification in LTL over a subset of states of a system under test while ignoring the valuation of the properties in the rest of the states. We introduce specialized operators that facilitate specifying properties over propositional scopes, where each scope constitutes a subset of states that satisfy a propositional logic formula. We also present the syntax and semantics of these operators and prove their correctness. Using the proposed operators, the user can specify properties more concisely and intuitively.

1. INTRODUCTION

Software and hardware systems are becoming more and more complex yet crucial and indispensable. Therefore, errors in such critical systems are very costly, thus, not permissible. A promising technique that, in theory, could guarantee absence of errors, is model checking. It allows the user to automatically check whether a model of a given system satisfies a set of required properties. However, over the years it has been realized that the main hurdles in applying model checking for software systems include the following. First, the complexity of modern programming languages and thus software systems is high. Second, it is cumbersome even to the expert, and virtually impossible [1] for the novice, to specify meaningful (often complex) properties using the usual temporal logic formalisms of model checking. This difficulty of expressing properties of the system grows even bigger when it comes to specifying properties of interest on part of a given system while ignoring the rest of it.

In this paper, we address the problem of property specification in LTL assuming that the user is interested in verifying properties over a subset of states while ignoring the valuation of those properties in the remaining states. We argue that applying abstraction techniques on the system model, where "uninteresting states" are discarded from the system model, and specifying the properties on the resulting model, may not constitute an adequate solution to the problem.

This work proposes a solution to the stated problem that does not require any changes in the system model. The idea is to define specialized operators for the specification of properties in a subset of states of interest. The new operators do not change the expressiveness of LTL, but rather help specifying the properties of interest more intuitively and succinctly.

The rest of this paper is organized as follows. In Section 2, we present an overview of the LTL variant, used in the paper, and explain the problem addressed in this paper as well as motives, related to known difficulties in the specification of properties in LTL formalism. Section 3 describes our solution, namely the syntax and semantics of the new operators and the proof of their correctness. In Section 4, we illustrate the usefulness of our approach with an example. In Section 5, we show how our results can be used in the existing specification patterns. In Section 6, we review the related work and conclude in Section 7.

2. LINEAR TEMPORAL LOGIC

2.1 LTL Formalism

LTL (or sometimes called PLTL) extends traditional propositional logic with temporal operators. Thus, LTL allows assertions about the temporal behavior of a system [13,14,19]. An LTL formula φ has the following syntax:

$$\varphi ::= p \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \,\mathrm{U}\, \varphi) \mid (\mathrm{G}\ \varphi) \mid (\mathrm{F}\ \varphi) \mid (\mathrm{X}\ \varphi)$$

where p is an atomic proposition, U is the *until* operator, G (or $\Box$) is the *always* operator, F (or $\Diamond$) is the *eventually* operator, and X (or o) is the *next* operator.

LTL semantics is originally defined by Pnueli [19] over infinite sequences of states that correspond to infinite or non-terminating sequences of computations. However, over the years, there has been more and more interest in run-time LTL verification that overcomes several inherent problems of model checking of full-scale models. Finite traces could be tackled by looping the last state. Here, we follow a variant of LTL semantics that applies to both finite and infinite traces, which is recently developed [27] in the context of LTL monitoring. Our choice is partially motivated by efficient application of scopes in our web application analysis tool [11]. While this variant may differ from the classical in boundary cases, we believe that our ideas apply to classical LTL as well.

Given a set of atomic propositions *AP*, let $M = (S, T, S_0, L)$ be a Kripke structure, where S is a set of states, $T \subseteq S \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, and L is a labeling function from S to the power set of *AP*. A state sequence $\pi = \langle s_0, s_1, \ldots\rangle$ is called a path of M if $s_0 \in S_0$, $(s_i, s_{i+1}) \in T$ for all i, $i \geq 0$. We denote by $|\pi|$ the length of a given state sequence π; if π is an infinite sequence of states, then $|\pi| = \infty$, assuming that ∞ is greater than any integer. An empty sequence of states is denoted ε; $|\varepsilon| = 0$. A path is called finite if it is a finite sequence of the form $\langle s_0, s_1, \ldots, s_k\rangle$, such that $s_0 \in S_0$, $(s_i, s_{i+1}) \in T$, and for all $s \in S$ $(s_k, s) \notin T$. $\pi^i = \langle s_i, s_{i+1}, \ldots\rangle$ denotes the suffix of a sequence $\pi = \langle s_0, s_1, \ldots\rangle$ starting at s_i. We assume that $\pi^i = \varepsilon$ for $|\pi| \leq i$. Also, note that $\pi^0 = \pi$.

The semantics of LTL formulae is defined as follows:

1. $\pi \models p \Leftrightarrow |\pi| > 0$, and $p \in L(s_0)$,

2. $\pi \models \neg\varphi \Leftrightarrow \pi \not\models \varphi$,

3. $\pi \models \varphi \wedge \psi \Leftrightarrow \pi \models \varphi$ and $\pi \models \psi$,

4. $\pi \models X\ \varphi \Leftrightarrow \pi^1 \models \varphi$,

5. $\pi \models G\ \varphi \Leftrightarrow$ for all i, $0 \le i < |\pi|$, $\pi^i \models \varphi$,

6. $\pi \models F\ \varphi \Leftrightarrow$ for some i, $0 \le i < |\pi|$, $\pi^i \models \varphi$,

7. $\pi \models \varphi\ U\ \psi \Leftrightarrow$ there exists an i, $0 \le i < |\pi|$ such that $\pi^i \models \psi$, and for all j, $0 \le j < i$, $\pi^j \models \varphi$.

Unlike the classical definition of LTL [19], our definition, inspired by [27], takes into account both infinite and finite cases. Note that for the case of $\pi = \varepsilon$, $\pi \models F\varphi$, and $\pi \models X\ \varphi$ do not hold. If $|\pi| = 1$, then $\pi \models X\ \varphi$ does not hold.

A Kripke structure satisfies a given formula φ ($M \models \varphi$) if and only if for every path π of M, $\pi \models \varphi$.

Let False be an abbreviation for $\varphi \wedge \neg\varphi$, and True an abbreviation for $\neg$False. The following are also abbreviations: $\varphi \vee \psi = \neg((\neg\varphi) \wedge (\neg\psi))$, $\varphi \rightarrow \psi = \neg\varphi \vee \psi$, $\varphi + \psi = (\varphi \vee \psi) \wedge \neg(\varphi \wedge \psi)$, $\varphi\ R\ \psi = \neg((\neg\varphi)\ U\ (\neg\psi))$, where R denotes the *release* operator which is the dual of until (U) operator, $\varphi\ W\ \psi = F\ \psi \rightarrow \varphi\ U\ \psi$, where W denotes the *weak* until operator. Two LTL formulae φ and ψ are said to be *semantically equivalent*, written $\varphi \equiv \psi$, if any state in one model, which satisfies one of them, also satisfies the other. Based on this definition, the following holds for any two LTL formulae φ and ψ: $\varphi \vee \psi \equiv \neg((\neg\varphi) \wedge (\neg\psi))$, $G\varphi \equiv \neg F\neg\varphi$, $F\varphi \equiv$ True $U\ \varphi$. These equivalences suggest that, in general, any LTL formula is expressible in terms of $\neg$, $\wedge$, U, and X.

2.2 LTL Limitations

Although LTL has been widely considered a natural choice for automata based verification of reactive systems, it suffers from few limitations when it comes to the expressiveness of the language [14]. Over the last two decades, there have been discussions [9,16,23] on comparing LTL to other formalisms such as CTL, μ-calculus, automata, etc. Each formalism has its own pros and cons in terms of expressiveness and complexity (when it comes to executing the verification algorithm). These advantages and disadvantages could vary to different research and industrial communities depending on the specific needs.

Expressive power of LTL is sufficient for many practical specification tasks, however, specifying non-trivial properties in LTL, as well as in other temporal logics, is often considered difficult even for experts and virtually impossible for novices [1]. One of particularly difficult problems is the expression of non-trivial properties which are related

only to a subset of the states of a system under test. While the problem was partially resolved with templates [7,8], syntax sugar [1], visual tools for property specification such as the Timeline editor [20], and even designated graphical logics for intervals of states [6], it is not yet resolved for more arbitrary state subsets, e.g., defined by a propositional formula. A suitable approach might be to add so-called syntax sugar operators, which allow succinct property specification, while known LTL model checking tools and algorithms still apply. The challenge is to specify properties over certain states that are of interest while ignoring the validity of the property in the remaining states. In other words, one needs to define a scope as an arbitrary set of states over a single path and verify the property over that scope. Distinction between the states of a system may, for example, depend on the type of actions enabled at each state. This has practical significance since the resulting partition of states could be used to express various levels of granularity at which the behavior of the system is described. Example of state partitioning is *stable* (*quiescent*) vs. *unstable* states (sometimes called *transient*), or *final* vs. *intermediate* [18,22,24,11]. As an example, in [11] a framework for modeling and verification of web applications uses communicating automata to model various entities of a given web application, namely, frames and multiple windows. Each entity is modeled by an automaton where states represent the pages displayed in the entity and transitions represent HTTP requests of these pages. In case of frames, we consider a display of pages stable, when the pages displayed in frames are all loaded and shown to the user. Thus, we have stable and transient global states in the communicating automata composition. Then, the user may wish to verify a property in stable (unstable) states only. A simple example is the property F*p* which is valid on a path, but is invalid in designated stable states of the same path.

The property "eventually *p* on stable states of path" could be reformulated as $F(p \wedge stable)$, where *stable* is a predicate that identifies stable states. However, for more complicated properties, even for other operators, the solution is not as simple as a conjunction of a predicate with the formula, as we show in Section 3. This problem was mentioned in [8], where the authors stated that Boolean variables could be used in the property specification to distinguish between states and concluded that simple conjunction and disjunction of Booleans with the original property do not serve the purpose.

A straightforward solution to this problem is to remove the "uninteresting" states from the model leaving only the subset of states in which the properties need to be verified. This solution would rely on a projection of a given Kripke structure onto a subset of states that are of interest.

Definition 1. *Let* $M = (S, T, S_0, L)$ *and* $M' = (S', T', S_0', L')$ *be two Kripke structures such that* $S' \subseteq S$. *We say that* M' *is a projection of* M *onto* S', *iff*

$T' = \{(s_i, s_k) \mid s_i, s_k \in S'$, *and either* $(s_i, s_k) \in T$ *or there exists a path suffix* $\pi^i = \langle s_i, s_{i+1}, \ldots, s_{k-1}, s_k, \ldots\rangle$, *such that* $s_{i+1}, \ldots, s_{k-1} \notin S'\}$,

$S_0' = \{s_i \mid s_i \in S_0 \cap S'$ *or there exists a path* $\pi = \langle s_0, s_1, \ldots, s_{i-1}, s_i, \ldots\rangle$ *of* M, *such that* $s_0 \in S_0 \setminus S'$ *and* $s_0, \ldots, s_{i-1} \notin S'\}$, *and*

$L'(s) = L(s)$ *for all* $s \in S'$.

Note that if $S' = S$, then $M' = M$.

Then, a standard model checking algorithm [5] could be used to verify the properties on the projection of the model. However, with such a solution, one faces two main problems:

1. If there exists a number of properties each of which concerns a different subset of states, then for each property one has to project the model separately. So the number of models may reach the number of properties.

2. The proposed solution may be not applicable for model checkers, like Spin [13], which use Kripke representation internally, and where the user specifies a modular system in a high level language, such as Promela.

Our solution is to introduce new LTL operators so that properties are verified over an arbitrary subset of states, and at the same time, standard LTL model checking algorithms and tools can still be used. Such a solution does not require any change in the model of the system; it rather helps in expressing properties in question more succinctly and intuitively. As we prove in the next section, the semantics of the new operators follow from the semantics of existing LTL operators.

3. EXTENDING LTL WITH PROPOSITIONAL SCOPES

In this section, we discuss how to specify LTL properties that should be verified over arbitrary subsets of the state space of the system under test. However, we first give a definition of a scope. In first order logic [25], a scope is defined as follows. Given the formulae $\forall xB$, and $\exists xB$, where B is a formula, B is called the *scope* of the respective quantifier, and any occurrence of variable x in the scope of a quantifier is bound by the closest $\forall x$ or $\exists x$. Similarly, we define a scope of a linear temporal logic formula over a given path as the subset of states on the path where the formula is checked.

Based on this definition, we consider the partition of the state space into in-scope and out-of-scope states. In-scope states are the states of interest, where a given property has to be checked, while ignoring the valuation of the property in the remaining states, which we designate out-of-scope states. For this purpose, we introduce new LTL operators that can be used to formulate properties in the in-scope states. These operators do not extend the LTL formalism; they rather help formulating properties more succinctly. We denote $\mathfrak{I}$ a propositional logic expression that valuates to True in every state where a given property should be verified. The set of states in which $\mathfrak{I}$ holds constitutes what we call $\mathfrak{I}$-*scope*.

Since any LTL property is expressible with the $\neg$, $\wedge$, U, and X operators, we generalize them as the operators $\neg_{\mathfrak{I}}$ (*not in scope*), $\wedge_{\mathfrak{I}}$ (*and in scope*), $U_{\mathfrak{I}}$ (*until in scope*), and $X_{\mathfrak{I}}$ (*next in scope*), and use the obtained operators to derive $F_{\mathfrak{I}}$ (*eventually in scope*) and $G_{\mathfrak{I}}$ (*always in scope*). If $\mathfrak{I}$-scope is the full set of states of the system, those operators coincide with their corresponding LTL counterparts, namely, $\neg$, $\wedge$, U, X, F, and G.

In the following, we formally define the $\mathfrak{I}$-scope operators. However, we first explain their intended semantics informally to clarify the intuition behind each of them. For example, $\neg_{\mathfrak{I}} \varphi$ implies that φ does not hold in the first in-scope state encountered. $\varphi U_{\mathfrak{I}} \psi$ means that φ holds in all the in-scope states preceding the one in which ψ holds. $X_{\mathfrak{I}}\varphi$ means that φ holds in the next in-scope state after the first such state encountered, and if no in-scope states exist along the path, the property will not hold. $F_{\mathfrak{I}}\varphi$ means that φ eventually holds in an in-scope state irrespective of its validity in out-of-scope states. Similarly, $G_{\mathfrak{I}}\varphi$ means that φ holds in all the in-scope states on the path. Figure 1 shows examples of properties using $\mathfrak{I}$-scope operators.

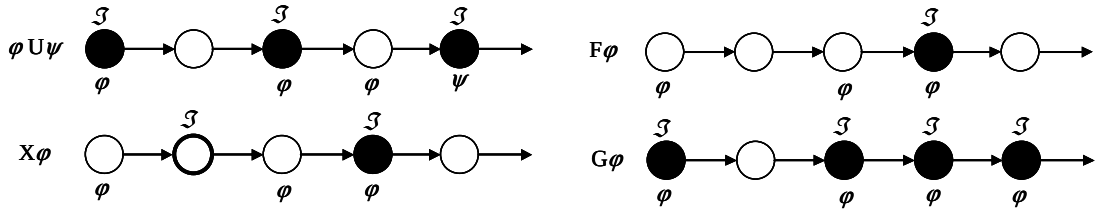

Figure 1. Examples of properties using $\mathfrak{I}$-scope operators.

Definition 2. *Let $\mathfrak{I}$ be a propositional logic expression, the operators $\neg_{\mathfrak{I}}$, $\wedge_{\mathfrak{I}}$, $U_{\mathfrak{I}}$, $X_{\mathfrak{I}}$, $F_{\mathfrak{I}}$, and $G_{\mathfrak{I}}$ are as follows:*

1. $\neg_{\mathfrak{I}} \varphi \stackrel{def}{=} \neg\, \mathfrak{I}\, U\, (\neg\varphi \wedge \mathfrak{I})$
2. $\varphi \wedge_{\mathfrak{I}} \psi \stackrel{def}{=} \neg\mathfrak{I}\, U\, ((\varphi \wedge \psi) \wedge \mathfrak{I})$
3. $\varphi U_{\mathfrak{I}} \psi \stackrel{def}{=} (\mathfrak{I} \rightarrow \varphi)\, U\, (\psi \wedge \mathfrak{I})$
4. $X_{\mathfrak{I}} \varphi \stackrel{def}{=} \neg\mathfrak{I}\, U\, [\mathfrak{I} \wedge X\, (\neg\mathfrak{I}\, U\, (\mathfrak{I} \wedge \varphi))]$
5. $F_{\mathfrak{I}} \varphi \stackrel{def}{=} F\, (\varphi \wedge \mathfrak{I})$
6. $G_{\mathfrak{I}} \varphi \stackrel{def}{=} G\, (\mathfrak{I} \rightarrow \varphi)$

Based on these auxiliary definitions, we introduce the $\mathfrak{I}$-scope operator denoted **In**, with the following recursive definition.

Definition 3. Let $\mathfrak{I}$ be a propositional logic expression, the $\mathfrak{I}$-scope operator **In** is defined as follows:

1. p **In** $\mathfrak{I} = \neg\mathfrak{I}\ \mathrm{U}\ (p \wedge \mathfrak{I})$
2. $(\neg\varphi)$ **In** $\mathfrak{I} = \neg_{\mathfrak{I}}(\varphi$ **In** $\mathfrak{I})$
3. $(\varphi \wedge \psi)$ **In** $\mathfrak{I} = (\varphi$ **In** $\mathfrak{I}) \wedge_{\mathfrak{I}} (\psi$ **In** $\mathfrak{I})$
4. $(\varphi\ \mathrm{U}\ \psi)$ **In** $\mathfrak{I} = (\varphi$ **In** $\mathfrak{I})\ \mathrm{U}_{\mathfrak{I}}\ (\psi$ **In** $\mathfrak{I})$
5. $(\mathrm{G}\ \varphi)$ **In** $\mathfrak{I} = \mathrm{G}_{\mathfrak{I}}(\varphi$ **In** $\mathfrak{I})$
6. $(\mathrm{F}\ \varphi)$ **In** $\mathfrak{I} = \mathrm{F}_{\mathfrak{I}}(\varphi$ **In** $\mathfrak{I})$
7. $(\mathrm{X}\ \varphi)$ **In** $\mathfrak{I} = \mathrm{X}_{\mathfrak{I}}(\varphi$ **In** $\mathfrak{I})$

The following lemmas and theorems describe the semantics of the introduced operators.

Lemma 1. $\pi \models p$ **In** $\mathfrak{I} \Leftrightarrow$ *there exists an* i, $0 \le i < |\pi|$, *such that* $\pi^i \models p$ *and* $\pi^i \models \mathfrak{I}$, *and for all* j, $0 \le j < i$, $\pi^j \not\models \mathfrak{I}$.

Proof. $\pi \models p$ **In** $\mathfrak{I} \Leftrightarrow$ (According to the definition of p **In** $\mathfrak{I}$), $\pi \models \neg\mathfrak{I}\ \mathrm{U}\ (p \wedge \mathfrak{I})$.

$\Leftrightarrow$ (By semantics of U), there exists an i, $0 \le i < |\pi|$ such that $\pi^i \models (p \wedge \mathfrak{I})$ and for all j, $0 \le j < i$, $\pi^j \models (\neg\mathfrak{I})$, and (by semantics of $\wedge$) $\pi^i \models p$ and $\pi^i \models \mathfrak{I}$, (by semantics of $\neg$) $\pi^j \not\models \mathfrak{I}$.

$\Leftrightarrow$ There exists an i, $1 \le i < |\pi|$ such that $\pi^i \models p$ and $\pi^i \models \mathfrak{I}$, and for all j, $0 \le j < i$, $\pi^j \not\models \mathfrak{I}$. QED

Lemma 2. $\pi \models \neg_{\mathfrak{I}}\varphi \Leftrightarrow$ *there exists an* i, $0 \le i < |\pi|$, *such that* $\pi^i \not\models \varphi$ *and* $\pi^i \models \mathfrak{I}$, *and for all* j, $0 \le j < i$, $\pi^j \not\models \mathfrak{I}$.

Proof. Lemma 2 directly follows from Lemma 1. QED

Lemma 3. $\pi \models \varphi\ \mathrm{U}_{\mathfrak{I}}\ \psi \Leftrightarrow$ *there exists an* i, $0 \le i < |\pi|$, *such that* $\pi^i \models \psi$ *and* $\pi^i \models \mathfrak{I}$, *and for all* j, $0 \le j < i$ $\pi^j \not\models \mathfrak{I}$ *or* $\pi^j \models \varphi$.

Proof. $\pi \models \varphi \, U_{\mathfrak{I}} \, \psi \Leftrightarrow$ (According to definition of $U_{\mathfrak{I}}$), $\pi \models (\mathfrak{I} \rightarrow \varphi)$ U $(\psi \wedge \mathfrak{I})$.

$\Leftrightarrow$ (By semantics of U), there exists an i, $0 \leq i < |\pi|$ such that $\pi^i \models (\psi \wedge \mathfrak{I})$ and for all j, $0 \leq j < i$, $\pi^j \models (\mathfrak{I} \rightarrow \varphi)$; and (by semantics of $\wedge$), $\pi^i \models \psi$ and $\pi^i \models \mathfrak{I}$, and (by definition of $\rightarrow$), $\mathfrak{I} \rightarrow \varphi = \neg \mathfrak{I} \vee \varphi$, and (by semantics of $\neg$ and $\vee$) $\pi^j \not\models \mathfrak{I}$ or $\pi^j \models \varphi$.

$\Leftrightarrow$ There exists an i, $0 \leq i < |\pi|$ such that $\pi^i \models \psi$ and $\pi^i \models \mathfrak{I}$, and for all j, $0 \leq j < i$, $\pi^j \not\models \mathfrak{I}$ or $\pi^j \models \varphi$. QED

Lemma 4. $\pi \models X_{\mathfrak{I}} \varphi \Leftrightarrow$ *there exist* i, k, $0 \leq i < k \leq |\pi|$, *such that* $\pi^i \models \mathfrak{I}$, $\pi^k \models \mathfrak{I}$ *and* $\pi^k \models \varphi$, *and for all* j, l, $0 \leq j < i < l < k$, $\pi^j \not\models \mathfrak{I}$ *and* $\pi^l \not\models \mathfrak{I}$.

Proof. $\pi \models X_{\mathfrak{I}} \varphi \Leftrightarrow \pi \models \neg\mathfrak{I}$ U $[\mathfrak{I} \wedge X (\neg\mathfrak{I}$ U $(\mathfrak{I} \wedge \varphi))]$.

$\Leftrightarrow$ (According to semantics of U), there exists an i, $0 \leq i < |\pi|$ such that $\pi^i \models [\mathfrak{I} \wedge X (\neg\mathfrak{I}$ U $(\mathfrak{I} \wedge \varphi))]$ and for all j, $0 \leq j < i$, $\pi^j \models \neg \mathfrak{I}$; and (by semantics of $\wedge$), $\pi^i \models \mathfrak{I}$ and $\pi^i \models X (\neg\mathfrak{I}$ U $(\mathfrak{I} \wedge \varphi))$, and (by semantics of X), $\pi^{i+1} \models (\neg\mathfrak{I}$ U $(\mathfrak{I} \wedge \varphi))$; and (by semantics of U), there exists k, $i + 1 \leq k < |\pi|$ such that $\pi^k \models (\mathfrak{I} \wedge \varphi)$ and for all l, $i < l < k$, $\pi^l \models \neg\mathfrak{I}$.

$\Leftrightarrow$ There exist i, k, $0 \leq i < k < |\pi|$, such that $\pi^i \models \mathfrak{I}$, $\pi^k \models \mathfrak{I}$ and $\pi^k \models \varphi$, and for all j, l, $0 \leq j < i < l < k$, $\pi^j \not\models \mathfrak{I}$, and $\pi^l \not\models \mathfrak{I}$. QED

Lemma 5. $\pi \models F_{\mathfrak{I}} \varphi \Leftrightarrow$ *for some* i, $0 \leq i < |\pi|$, $\pi^i \models \varphi$ *and* $\pi^i \models \mathfrak{I}$.

Proof. Directly follows from the semantics of F. QED

Lemma 6. $\pi \models G_{\mathfrak{I}} \varphi \Leftrightarrow$ *for all* i, $0 \leq i < |\pi|$, *where* $\pi^i \models \mathfrak{I}$, $\pi^i \models \varphi$.

Proof. Directly follows from the semantics of G. QED

To demonstrate the correctness of the definitions and semantics of $\mathfrak{I}$-*scope* operators and formulae, we state two theorems in which we claim that if a formula (φ **In** $\mathfrak{I}$) holds in a given path (model) including in-scope and out-of-scope states the corresponding LTL formula φ must hold in the projection of the path (model) including only the in-scope states. To this end, we first define a projection relation based on Definition 1 that removes the out-of-scope states from a path and keeps only the in-scope states.

Definition 4. *Let $\mathfrak{I}$ be a propositional logic formula, $M = (S, T, S_0, L)$ be a Kripke structure, $S_{\mathfrak{I}} = \{s \in S \mid \mathfrak{I}$ is true in $s\}$, and let $\pi = \langle s_0, s_1, \ldots\rangle$ be a path of M. The projection of π onto $S_{\mathfrak{I}}$, denoted $\pi_{\downarrow\mathfrak{I}}$, is the* (*possibly finite*) *subsequence of π derived by discarding all states s_i from π such that $s_i \notin S_{\mathfrak{I}}$.*

Proposition 1. *Let $\mathfrak{I}$ be a propositional logic formula, and $M = (S, T, S_0, L)$ and $M_{\mathfrak{I}} = (S_{\mathfrak{I}}, T_{\mathfrak{I}}, S_{0\mathfrak{I}}, L_{\mathfrak{I}})$ be two Kripke structures, where $M_{\mathfrak{I}}$ is the projection of M onto $S_{\mathfrak{I}}$, and π be a path of M, then $\pi_{\downarrow\mathfrak{I}} \neq \varepsilon$ is a path of $M_{\mathfrak{I}}$.*

Theorem 1. *For any LTL formula φ and its corresponding formula φ In $\mathfrak{I}$, $\pi \vDash \varphi$* **In** *$\mathfrak{I}$ if and only if $\pi_{\downarrow\mathfrak{I}} \vDash \varphi$.*

Proof. The proof is by induction on the number of operators, n, of a formula φ. For simplicity, we assume that $\mathfrak{I}$ is an atomic proposition. The proof could easily be extended onto any propositional $\mathfrak{I}$. To make the proof more intuitive, we index the formulae with the number of operators that constitute each formula.

Base case. For $n = 0$, where the formula is simply an atomic predicate and φ_0 **In** $\mathfrak{I} = p$ **In** $\mathfrak{I}$, we prove that $\pi \vDash p$ **In** $\mathfrak{I}$ if and only if $\pi_{\downarrow\mathfrak{I}} \vDash p$.

$\pi \vDash p$ **In** $\mathfrak{I} \Leftrightarrow \pi \vDash \neg\, \mathfrak{I}\, \mathrm{U}\, (p \wedge \mathfrak{I})$.

$\Leftrightarrow$ (According to Lemma 1) there exists an i, $0 \le i < |\pi|$, such that $\pi^i \vDash p$ and $\pi^i \vDash \mathfrak{I}$, and for all j, $0 \le j < i$, $\pi^j \nvDash \mathfrak{I}$.

$\Leftrightarrow$ There exists an i, $0 \le i < |\pi|$, such that $p \in L(s_i)$ and $\mathfrak{I} \in L(s_i)$, and for all j, $0 \le j < i$, $\mathfrak{I} \notin L(s_j)$; and (by Definition 4) s_i is the first state in $\pi^i{}_{\downarrow\mathfrak{I}}$ and for all j, $0 \le j < i$, s_j is not in $\pi^j{}_{\downarrow\mathfrak{I}}$.

$\Leftrightarrow$ There exists an i, $0 \le i < |\pi|$, such that $\pi = \langle \ldots, s_i, \ldots\rangle$, $\pi^i{}_{\downarrow\mathfrak{I}} \vDash p$ and $\pi_{\downarrow\mathfrak{I}} = \pi^i{}_{\downarrow\mathfrak{I}} = \langle s_i, \ldots\rangle$.

$\Leftrightarrow$ (According to the semantics of p) $\pi_{\downarrow\mathfrak{I}} \vDash p$.

Inductive step. Assume $\pi \vDash \varphi_m$ **In** $\mathfrak{I}$ if and only if $\pi_{\downarrow\mathfrak{I}} \vDash \varphi_m$ for all formulae φ_m that consist of m operators, m, $0 \le m \le n$, holds. We must show that the equivalence holds as well for $n + 1$. Due to lack of space, we present here the proofs only for the most complicated cases where the formula φ_{n+1} is of the form $\chi_u \,\mathrm{U}\, \psi_v$, where χ_u and ψ_v are formulae with u and v operators respectively, such that, $0 \le u \le n$, $0 \le v \le n$, and $u + v = n$,

and of the form X ψ_n. The proofs for the remaining LTL operators could be performed in a similar manner.

(1) Here we prove that for all formulae χ_u, ψ_v which together contain n or less operators, $\pi \vDash (\chi_u \text{ U } \psi_v)$ **In** $\mathfrak{I}$ if and only if $\pi_{\downarrow\mathfrak{I}} \vDash \chi_u \text{ U } \psi_v$.

$\pi \vDash (\chi_u \text{ U } \psi_v)$ **In** $\mathfrak{I} \Leftrightarrow \pi \vDash (\chi_u$ **In** $\mathfrak{I}) \text{ U}_{\mathfrak{I}} (\psi_v$ **In** $\mathfrak{I})$.

$\Leftrightarrow$ (According to Lemma 3) there exists an i, $0 \le i < |\pi|$, such that $\pi^i \vDash (\psi_v$ **In** $\mathfrak{I})$ and $\pi^i \vDash \mathfrak{I}$, and for all j, $0 \le j < i$ $\pi^j \nvDash \mathfrak{I}$ or $\pi^j \vDash (\chi_u$ **In** $\mathfrak{I})$.

$\Leftrightarrow$ There exists an i, $0 \le i < |\pi|$, such that (by Definition 4) s_i is the first state in $\pi^i{}_{\downarrow\mathfrak{I}}$, and (according to the induction hypothesis) for all i, $0 \le i < |\pi|$, $\pi^i{}_{\downarrow\mathfrak{I}} \vDash \psi_v$; and for all j, $0 \le j < i$, either (by Definition 4) s_j is not in $\pi^j{}_{\downarrow\mathfrak{I}}$, or s_j is a state in $\pi^j{}_{\downarrow\mathfrak{I}}$ and (by induction hypothesis) $\pi^j{}_{\downarrow\mathfrak{I}} \vDash \chi_u$.

$\Leftrightarrow$ There exists an i, $0 \le i < |\pi|$, such that $\pi = \langle \ldots, s_i, \ldots \rangle$, ($\pi^i{}_{\downarrow\mathfrak{I}} \vDash \psi_m$, and $\pi_{\downarrow\mathfrak{I}} = \pi^i{}_{\downarrow\mathfrak{I}} = \langle s_i, \ldots \rangle$) or ($\pi^i{}_{\downarrow\mathfrak{I}} \vDash \psi_v$, and for all j, $0 \le j < i$, $\pi^j{}_{\downarrow\mathfrak{I}} \vDash \chi_u$).

$\Leftrightarrow$ (By semantics of U) $\pi_{\downarrow\mathfrak{I}} \vDash \chi_u \text{ U } \psi_v$.

(2) Here we prove that for each formula ψ_n that contain n or less operators, $\pi \vDash (\text{X } \psi_n)$ **In** $\mathfrak{I}$ if and only if $\pi_{\downarrow\mathfrak{I}} \vDash \text{X } \psi_n$.

$\pi \vDash (\text{X } \psi_n)$ **In** $\mathfrak{I} \Leftrightarrow \text{X}_{\mathfrak{I}} (\psi_n$ **In** $\mathfrak{I})$.

$\Leftrightarrow$ (According to Lemma 4) there exist i, k, $0 \le i < k < |\pi|$, such that $\pi^i \vDash \mathfrak{I}$, $\pi^k \vDash \mathfrak{I}$ and $\pi^k \vDash (\psi_n$ **In** $\mathfrak{I})$, and for all j, l, $0 \le j < i < l < k$, $\pi^j \nvDash \mathfrak{I}$, and $\pi^l \nvDash \mathfrak{I}$.

$\Leftrightarrow$ There exist i, k, $0 \le i < k < |\pi|$, such that $\mathfrak{I} \in L(s_i)$, $\mathfrak{I} \in L(s_k)$, and $\pi^k \vDash (\psi_n$ **In** $\mathfrak{I})$, and for all j, $0 \le j < i$, $\mathfrak{I} \notin L(s_j)$, and for all l, $i+1 \le l < k$, $\mathfrak{I} \notin L(s_l)$.

$\Leftrightarrow$ There exist i, k, $0 \le i < k < |\pi|$, such that (by Definition 4) s_i is the first state in $\pi^i{}_{\downarrow\mathfrak{I}}$, s_k is the first state in $\pi^k{}_{\downarrow\mathfrak{I}}$, such that (by induction hypothesis) $\pi^k{}_{\downarrow\mathfrak{I}} \vDash \psi_n$, and for all j, $0 \le j < i$, s_j is not in $\pi^j{}_{\downarrow\mathfrak{I}}$, and for all l, $i+1 \le l < k$, s_l is not in $\pi^l{}_{\downarrow\mathfrak{I}}$.

$\Leftrightarrow$ There exist i, k, $0 \le i < k < |\pi|$, such that $\pi = \langle \ldots, s_i, \ldots, s_k, \ldots \rangle$, $\pi_{\downarrow\mathfrak{I}} = \langle s_i, s_k, \ldots \rangle$, and $\pi^k{}_{\downarrow\mathfrak{I}} \vDash \psi_n$.

$\Leftrightarrow$ (By semantics of X) $\pi_{\downarrow\mathfrak{I}} \vDash X\ \psi_n$.

The base case and the induction step imply that the theorem holds for all cases of *n*. QED

Given Proposition 1 and Theorem 1, we have the following theorem.

Theorem 2. *Let $\mathfrak{I}$ be a propositional logic formula, and let $M = (S, T, S_0, L)$ and $M_{\mathfrak{I}} = (S_{\mathfrak{I}}, T_{\mathfrak{I}}, S_{0\mathfrak{I}}, L_{\mathfrak{I}})$ be two Kripke structures, $M_{\mathfrak{I}}$ is the projection of M onto $S_{\mathfrak{I}} \subseteq S$ such that $\mathfrak{I}$ is true in all $s \in S_{\mathfrak{I}}$. For any LTL formula φ, $M \vDash \varphi$* **In** *$\mathfrak{I}$ if and only if $M_{\mathfrak{I}} \vDash \varphi$.*

4. EXAMPLE

We illustrate our approach with an example commonly used in automated planning [28], where model checking is being increasingly used. The system is the so-called Dock-Worker Robots or DWR domain, which represents a harbor with several locations corresponding to docks, docked ships, storage areas, etc [28]. In the harbor, robot carts are used in the loading and unloading of docked ships. In this example, we consider a property that expresses a desired robot behavior.

Robot1 visits Room1 then Room2 then returns to Room1 and does this exactly twice.

In this property, *Robot1* is assumed to start in *Room1* while the rooms are adjacent. We generalize the property to express a more realistic situation, assuming that the robot can start in any location and the two rooms are not adjacent (they are interleaved with other locations where the robot is allowed to pass). Therefore, given the stated property, we are not interested in the robot movement in any other locations other than *Room1* and *Room2*. Using standard LTL, the generalized property is non trivial and tricky to specify:

$$(\neg Room1 \wedge \neg Room2)\ U\ (Room1 \wedge \neg Room2 \wedge X\ (\neg Room2\ U\ (Room2 \wedge \neg Room1 \wedge X(\neg Room1\ U\ (Room1 \wedge \neg Room2 \wedge X\ (\neg Room2\ U\ (Room2 \wedge \neg Room1 \wedge X\ (\neg Room1\ U\ (Room1 \wedge \neg Room2 \wedge X\ (G\ (\neg Room2)))))))))) \quad (1)$$

If we use now the **In** operator, the scope of the property includes states where the robot can be in *Room1* or in *Room2*. Therefore, the scope can be written as: (*Room1* + *Room2*). Note that exclusive disjunction is used to account for a situation when the robot cart or train did not completely leave one location, while entering in another one. In the case when both locations are adjacent, the robot could be stuck between two locations instead of performing a repetitive movement.

Now, the property can be written, in a more intuitive and succinct way, with the **In** operator as follows:

$$Room1 \wedge X(Room2 \wedge X(Room1 \wedge X(Room2 \wedge X(Room1 \wedge G(\neg Room2)))))$$
$$\textbf{In}\ (Room1 + Room2) \qquad (2)$$

We leave the task of unfolding the formula in (2) and comparing the result to (1) as an exercise for the reader.

5. USING $\mathcal{I}$-SCOPE IN PROPERTY PATTERNS

Using scopes to limit the domain over which a property is verified was only recently addressed in [7,8] by Dwyer et al. The authors identify several scopes which are used to define the part of a system execution, where a property must hold. A scope is determined by specifying starting and ending state/event for the property. The defined scopes are then used within a system of property specification patterns that are useful for non-experts to read and write formal specifications of systems. We believe that the $\mathcal{I}$-scope, when combined with the existing scope definitions, provides a possibility to further enrich the expressiveness of patterns and make them more useful in practice.

5.1 System of Property Patterns

In [7,8], patterns are classified into two categories: Order and Occurrence, and they include:
Absence - A state/event does not occur within a scope;
Existence - A state/event must occur within a scope;
Universality - A state/event occurs throughout a scope;
Response - A state/event *P* must be always followed by a state/event *Q* within a scope;
Precedence - A state/event *P* must be always preceded by a state/event *Q* within a scope.

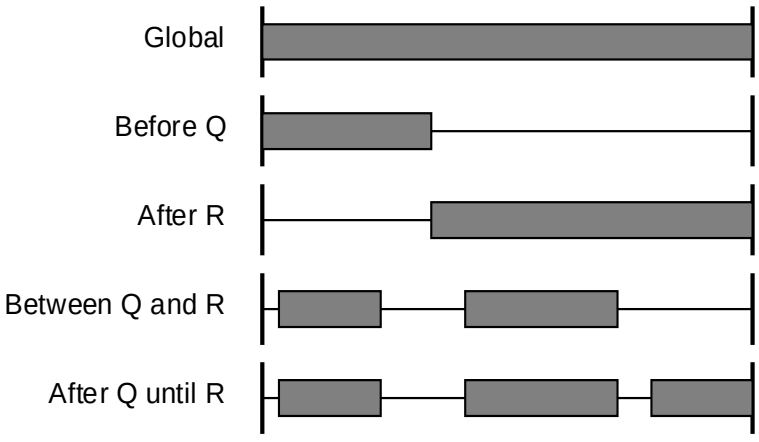

Figure 2. Pattern Scopes

In addition, there are five scopes (Figure 2) that can be associated to patterns:
Global - The entire execution path;
Before - The execution path up to a given state/event;
After - The execution path after a given state/event;
Between - Any part of the execution path from one given state/event to another given state/event;
After-until - Similar to scope Between, except that the designated part of the execution path continues even if the second state/event does not occur.

5.2 Combining $\mathcal{I}$-Scope with Pattern Scopes

In this section, we show how the defined $\mathcal{I}$-scope can be combined with the pattern scopes introduced in [7,8] to provide the user of the pattern system with more flexibility to specify real world properties. For example, given the existence pattern in the global scope, we can verify it also in some states of interest defined by a given propositional logic expression $\mathcal{I}$. Thus the pattern becomes "Exist Globally in $\mathcal{I}$-scope states" and the corresponding LTL formula is: F ($P \wedge \mathcal{I}$).

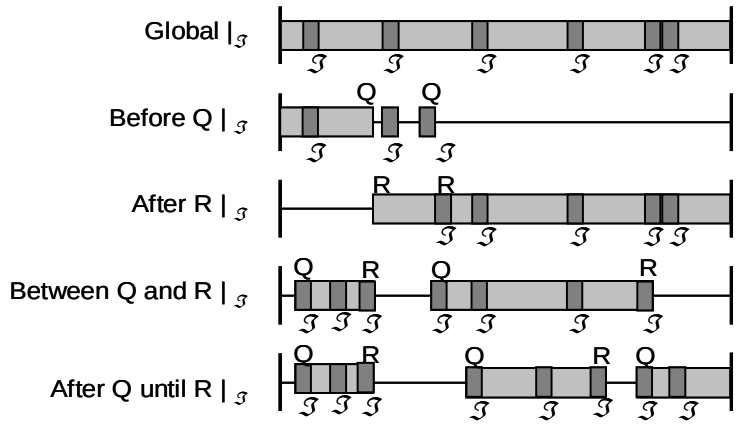

Figure 3. New Scopes

Figure 3 shows the combination of the original scopes (represented in light gray) and the $\mathcal{I}$-scope. The resulting scopes are shown using dark gray rectangles.

Consequently, we introduce LTL templates for the specification patterns with the modified scopes. These templates can be loaded into the LTL Property Manager of Spin where the propositional expression of the scope can be instantiated in the macro definition of the scope within the templates as needed by the user.

Figures 4 and 5 are examples of the new scopes applied to the patterns Absence, Existence, and Precedence, they also show the resulting templates. Note that new patterns are obtained

by rewriting the old patterns using the operator **In** and the proposition $\mathcal{I}$, and unfolding and simplifying the resulting formulae.

Absence pattern: *P* is false

Globally in $\mathcal{I}$-scope:	$G\ (\mathcal{I} \rightarrow \neg P)$
Before *R* in $\mathcal{I}$-scope:	$F\ (R \wedge \mathcal{I}) \rightarrow (\mathcal{I} \rightarrow \neg P)\ U\ (R \wedge \mathcal{I})$
After *Q* in $\mathcal{I}$-scope:	$G\ ((\mathcal{I} \rightarrow Q) \rightarrow G\ (\mathcal{I} \rightarrow \neg P))$
Between Q and R in $\mathcal{I}$-scope:	$G\ ((\mathcal{I} \rightarrow Q \wedge \neg R \wedge F\ (R \wedge \mathcal{I})) \rightarrow (\mathcal{I} \rightarrow \neg P)\ U\ (\mathcal{I} \wedge R))$
After Q until R in $\mathcal{I}$-scope:	$G\ ((\mathcal{I} \rightarrow Q \wedge \neg R) \rightarrow (\mathcal{I} \rightarrow \neg P)\ W\ (\mathcal{I} \wedge R))$

Existence pattern: *P* becomes true

Globally in $\mathcal{I}$-scope:	$F\ (P \wedge \mathcal{I})$
Before *R* in $\mathcal{I}$-scope:	$(\mathcal{I} \rightarrow \neg R)\ U\ ((\mathcal{I} \wedge P \wedge \neg R) \mid (\mathcal{I} \rightarrow \neg R))$
After *Q* in $\mathcal{I}$-scope:	$G\ (\mathcal{I} \rightarrow \neg Q) \mid F\ (\mathcal{I} \wedge Q \wedge F\ (\mathcal{I} \wedge P))$
Between Q and R in $\mathcal{I}$-scope:	$G\ ((\mathcal{I} \rightarrow Q \wedge \neg R) \rightarrow (\mathcal{I} \rightarrow \neg R)\ W\ (\mathcal{I} \wedge P \wedge \neg R))$
After Q until R in $\mathcal{I}$-scope:	$G\ ((\mathcal{I} \rightarrow Q \wedge \neg R) \rightarrow (\mathcal{I} \rightarrow \neg R)\ U\ (P \wedge \neg R \wedge \mathcal{I}))$

Precedence pattern: *S* precedes *P*

Globally in $\mathcal{I}$-scope:	$(\mathcal{I} \rightarrow \neg P)\ W\ (\mathcal{I} \wedge S)$
Before *R* in $\mathcal{I}$-scope:	$F(\mathcal{I} \wedge R) \rightarrow (\mathcal{I} \rightarrow P)\ U\ (\mathcal{I} \wedge (S \mid R))$
After *Q* in $\mathcal{I}$-scope:	$G\ (\mathcal{I} \rightarrow \neg Q) \mid F\ (\mathcal{I} \wedge Q \wedge (\mathcal{I} \rightarrow \neg P)\ W\ (\mathcal{I} \wedge S))$
Between Q and R in $\mathcal{I}$-scope:	$G\ ((\mathcal{I} \rightarrow Q \wedge \neg R \wedge F\ (R \wedge \mathcal{I})) \rightarrow (\mathcal{I} \rightarrow \neg P)\ U\ (\mathcal{I} \wedge (S \mid R)))$
After Q until R in $\mathcal{I}$-scope:	$G\ ((\mathcal{I} \rightarrow Q \wedge \neg R) \rightarrow (\mathcal{I} \rightarrow \neg P)\ W\ (\mathcal{I} \wedge (S \mid R)))$

Figure 4. Examples of Patterns with new Combined Scopes

```
#define p ?
#define i ?

/*
 * Formula As Typed: <>  (p && i)
 * The Never Claim Below Corresponds
 * To The Negated Formula !(<> (p && i))
 * (formalizing violations of the
 * original)
 */

never {    /*  !(<>  (p && i))  */
accept_init:
T0_init:
   if
   :: ((!((i)) || !((p)))) -> goto T0_init
   fi;
}
```

```
#define p ?
#define i ?
#define r ?
/*
 * Formula As Typed: (i ->  ! r) U ((i && p
 * &&  ! r)  ||  (i ->  ! r))
 * The Never Claim Below Corresponds
 * To The Negated Formula !((i ->  ! r) U ((i
 * && p &&  ! r)  ||  (i ->  ! r)))
 * (formalizing violations of the original)
 */
never { /*  !((i ->  ! r) U ((i && p &&  ! r)
          || (i ->  ! r)))" */
accept_init:
T0_init:
   if
   :: ((i) && (r)) -> goto T0_init
   :: ((i) && (r)) -> goto accept_all
   fi;
accept_all:
   skip
}
```

Figure 5. Templates for Existence pattern Globally in $\mathcal{I}$-scope, and Before R in $\mathcal{I}$-scope

In the future, we may consider temporal extension of the proposed scope operator, which will allows the application of not only propositional, but also above mentioned temporal scopes, to arbitrary formulae, possibly in an iterative manner.

6. RELATED WORK

There exist several works on introducing and/or extending specification logics based on existing ones, such as LTL and CTL, to ease the expression of properties of systems or allow a wider range of specifications to be expressed using these logics. Examples of such logics are QCTL [15], an extension of CTL with quantification over propositional formulae, XCTL [3], an extension of CTL to De Morgan Algebras, and DLTL [12], a logic that extends LTL by indexing the until operator with regular programs. The most recent work we know about is the one of Chaki et al [2]. The authors introduce a framework for modeling and verification of concurrent software systems, where both states and events are incorporated. For this purpose, they introduce SE-LTL, a specification logic based on LTL that allows both state and event requirements to be easily expressed.

Beer et al [1] propose the so-called Temporal Logic Sugar. They extend CTL with regular expressions and introduce new operators to formulate properties in CTL. These operators do not add expressive power to CTL, but make it easier for the non-expert user of CTL to specify properties of interest. The operator "next" defined in Sugar is close to the "next in scope" operator we provide, but our definition is more general. The Sugar next is defined as the next state in which a Boolean expression is valid. Our definition states that the property has to be true in the next state, in which a propositional logic expression is valid, relatively to the first state in which the propositional logic expression is valid and not to the first state of the execution path.

Dwyer et al [7,8] present and analyze over 500 temporal properties, classified in the proposed specification pattern system [26]. The patterns introduced constitute abstractions of specifications formulated for different formalisms in which such abstractions are not supported. The patterns are defined on five scopes that represent intervals/regions in which properties should be validated. These scopes have a start state and an end state. However, the authors did not address the problem of specifying these patterns in a scope of arbitrary set of states. Although the authors mention a class of properties that could be defined based on Boolean variables true in a state, but they state that the specification of these properties is not trivial and offer no solution to this problem. Moreover, there is no methodology to impose those scopes on a given pattern/formula. Since the specification pattern system does not use automatic combinations of scopes and basic patterns, some attempts are made to translate the system to lower-level automata specification languages. Such translations make pattern visualization, fiddling, and elucidation [10,21] possible.

Chechik et al [4,17] extend the pattern system of [7,8] and introduce edge based LTL property formulations in the same scopes introduced by Dwyer. The notion of an edge/event is represented by the "next" operator.

Dillon et al [6] introduce in the Graphical Interval logic (GIL) which is similar to the pattern system. GIL provides a mechanism to identify the region/interval of the system execution over which the property should be validated. The operators that identify the

intervals introduce a wider and more general range of scopes than the five scopes introduced by Dwyer. However, the operators are used to define segments of the execution path that must have a start and an end. There is no means offered to identify an arbitrary set of states in which properties should be verified.

7. CONCLUSION AND FUTURE WORK

In this paper, we presented new specialized LTL operators for specifying properties within a scope of arbitrary set of states of interest. These states are characterized by a propositional logic formula $\mathcal{I}$ and constitute what we call $\mathcal{I}$-scope. These operators do not extend the LTL formalism; they rather help formulate complex specifications more intuitively and succinctly. Thus, these operators do not require designated model checking algorithms. The new operators could be used to easily specify properties of the systems, in particular in the case when some states of a system are of technical character and should be skipped. We also demonstrated that the semantics of these operators follow from LTL semantics and proved their correctness. We believe that the proposed operators can be used in models that exhibit both infinite and finite behaviors.

Our future work will focus on the generalization of these results on temporal scopes, and mixed state/event properties, that could contribute to the improvement and elucidation of specification patterns. Generalizing our approach to introduce temporal scopes provides a richer and more flexible framework and methodology that make it is easier and more realizable to compose any two given LTL formulae and not only existing patterns. We also plan to develop a tool that integrates with the Spin LTL Property Manager. This tool will translate any given LTL property that includes scope operators into standard LTL and apply optimization and simplification rules on the resulting LTL formula.

REFERENCES

1. I. Beer, S. Ben-David, and C. Eisner, "The Temporal Logic Sugar" in Proc. of 13th Int. Conference on Computer Aided Verification (CAV 2001), LNCS, Vol. 2102, pp. 363–367.

2. S. Chaki, E.M. Clarke, and J. Ouaknine, N. Sharygina, N. Sinha, "State/Event-based Software Model Checking", in Proc. of 4th Int. Conference on Integrated Formal Methods (IFM 04), LNCS, Vol. 2999. Canterbury, England, April 2004.

3. M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel, "Multi-valued Symbolic Model-Checking" ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 12 , Issue 4, pp. 371 – 408, October 2003.

4. M. Chechik, and D. Paun, "Events in Property Patterns", in Proc. of 6th Int. SPIN Workshop on Theoretical and Practical Aspects of SPIN Model Checking, LNCS 1680, pp. 154-167, September 1999.

5. M. Clarke, O. Grumberg, D. A. Peled, "Model Checking", MIT Press, 2000.

6. L.K. Dillon, G. Kutty, L.E. Moser, "A Graphical Interval Logic for Specifying Concurrent Systems", ACM Transactions on Software Engineering and Methodology, Vol. 3(2), pp. 131-165, April 1994.

7. M. Dwyer, G.S. Avrunin, and J.C. Corbett, "Patterns in Property Specifications for Finite-state Verification", in Proc. of 21st Int. Conference on Software Engineering, May, 1999.

8. M. Dwyer, G.S. Avrunin, and J.C. Corbett, "Property Specification Patterns for Finite-state Verification", in Proc. of 2nd Workshop on Formal Methods in Software Practice, March, 1998.

9. E. Emerson, and J. Halpern, " 'sometimes' and 'not never' Revisited: on Branching versus Linear Temporal Logic", Journal of the ACM, 33(1), pp. 151-178, January 1986.

10. H. Hallal, A. Petrenko, A. Ulrich, and S. Boroday, "Using SDL Tools to Test Properties of Distributed Systems", in Proc. of Formal Approches to Testing of Software (FATES'01), Workshop of the Int. Conference on Concurrency Theory (CONCUR'01). pp. 125-140, Aalborg, Denmark, August 21-24, 2001.

11. M. Haydar, A. Petrenko, and H. Sahraoui, "Formal Verification of Web Applications Modeled by Communicating Automata", in Proc. of 24th IFIP WG 6.1 IFIP Int. Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004), LNCS, Vol. 3235, pp. 115-132, Spain, September 2004.

12. J.G. Henriksen, and P.S. Thiagarajan, “Dynamic Linear Time Temporal Logic”, in Annals of Pure and Applied logic, Vol.96, No.1-3, pp. 187–207, 1999.

13. G. J. Holzmann, “The Spin Model Checker, Primer and Reference Manual”, Addison-Wesley, 2003.

14. M.R.A. Huth, and M.D. Ryan, “Logic in Computer Science: Modelling and Reasoning about Systems”, Cambridge University Press, 2000.

15. O. Kupferman, “Augmenting branching temporal logics with existential quantification over atomic propositions”, Journal of Logic and Computation, Vol. 9(2), p. 135-147, April 1999.

16. L. Lamport, “Sometimes is sometimes ‘not never’ ”, in Proc. of 7th ACM Symposium on Principles of Programming Languages, pp. 174-185, January 1980.

17. D. Paun, and M. Chechick, “Events in Linear-Time Properties”, in Proc. of 4th IEEE Int. Symposium on Requirements Engineering, June 1999.

18. A. Petrenko, N. Yevtushenko, G.v. Bochmann, and R. Dssouli, "Testing in Context: Framework and Test Derivation", Computer Communications Journal, Special issue on Protocol engineering, Vol. 19, pp. 1236-1249, 1996.

19. A. Pnueli, “The Temporal Logic of Programs”, in Proc. of the 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 46-57.

20. M. H. Smith, G. J. Holzmann, and K. Etessami, “Events and Constraints: a Graphical Editor for Capturing Logic Properties of Programs”, In Proc. of the 5th Int. Symposium on Requirements Engineering, August 2001.

21. R.L. Smith, G.S. Avrunin, L.A. Clarke, L.J. Osterweil, “PROPEL: an Approach Supporting Property Elucidation”, in Proc. of 24th Int. Conference on Software Engineering (ICSE 2002), pp. 11 – 21, Orlando, Florida, 2002.

22. J. Tretmans, “Test Generation with Inputs, Outputs and Repetitive Quiescence”, Software-Concepts and Tools, Vol.3, pp. 103-120, 1996.

23. M.Y. Vardi, “Branching vs. Linear Time: Final Showdown”, in Proc. of 7th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001), LNCS, Vol. 2031 pp. 1–22, Italy, April 2001.

24. C.H. West, “An Automated Technique of Communication Protocols Validation”, IEEE Trans. on Comm. Vol. 26, pp. 1271-1275, 1978.

25. Mathworld, The Web's most Extensive Mathematics Resource, http://mathworld.wolfram.com/.

26. The Specification Patterns System, http://patterns.projects.cis.ksu.edu/.

27. H. Barringer, A. Goldberg, K. Havelund, and K. Sen, "Eagle Does Space Efficient LTL Monitoring", Technical Report, CSPP-25, University of Manchester, Department of Computer Science, October 2003.

28. M. Ghallab, D. Nau, and P. Traverso, "Automated Planning: Theory and Practice", Morgan Kaufmann Publishers, May 2004.