Properties and Scopes in Web Model Checking

May Haydar[1,2] Sergiy Boroday[1] Alexandre Petrenko[1] Houari Sahraoui[2]

[1] Centre de recherche informatique de Montréal
550 Sherbrooke West, #100, Montreal, QC, H3A1B9, Canada

{May.Haydar,Sergiy.Boroday,Alexandre.Petrenko}@crim.ca

[2] Département d'informatique et de recherche opérationnelle, Université de Montréal
CP 6128 succ. Centre-Ville, Montreal, QC, H3C 3J7, Canada

Sahraouh@iro.umontreal.ca

ABSTRACT

We consider a formal framework for property verification of web applications using Spin model checker. Some of the web related properties concern all states of the model, while others – only a proper subset of them. To be able to discriminate states of interest in the state space, we solve 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 them. 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. Using the proposed operators, the user can specify web properties more concisely and intuitively. We illustrate the proposed solution in specifying properties of web applications and discuss other potential applications.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: *Formal methods, Model checking, Validation*; F.4 [**Mathematical Logic and Formal Languages**]: *Temporal logic.*

General Terms

Languages, Formal Methods, Temporal Logic, Verification.

Keywords

Web Applications, Model Checking, Linear Temporal Logic.

1. INTRODUCTION

In recent years, it has been realized that applying model checking [2] to the verification of software systems is not always straightforward. The complexity of modern programming languages and thus software systems is high, which often makes it rather difficult to extract models of the given applications. Meanwhile, it is difficult even to the expert, while virtually impossible [1] for the novice, to specify meaningful (often complex) properties using the usual temporal logic formalisms of model checking. The problem of property specification becomes even more difficult to solve when we want to express certain specifications on a part of the given system behavior 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 checking the properties over a subset of the states of a given system while ignoring the rest of the states. We propose a generic and practical solution to ease the problem of property specification in LTL over subsets of states. Our solution does not require any changes in the system model. The idea is to define specialized operators in LTL for the specification of properties over a subset of states to define the required scope of the property.

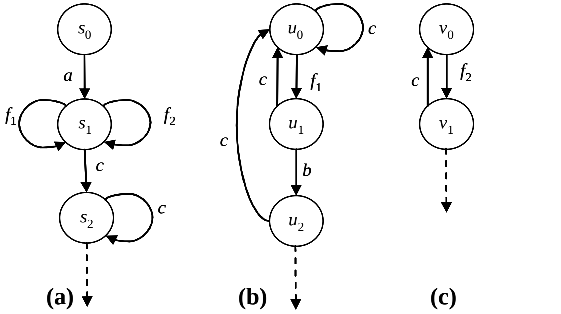
The new operators do not affect the expressiveness of LTL, but rather help specifying the properties of interest more intuitively and succinctly. We address the problem of property scopes in the context of our formal framework for analysis and validation of web applications (WA) [4, 5], where we use communicating automata to model the different components of a given web application, namely windows, frames, and framesets. Each component is modeled by an automaton where states represent the pages displayed in the entity and transitions represent HTTP requests of these pages. The behavior of the web application is then represented by the composition of all the component automata. A state is *stable* if it represents a page that is fully loaded and displayed to the user; otherwise, the state is called *transient*. Distinction between stable and transient states becomes evident in applications that use frames, where a state is stable (loading is complete) when the pages displayed in a frame are all fully loaded and shown to the user. We argue that some properties are relevant to all the states (stable and transient), while others should be verified only on either stable or transient states. Thus, the scopes constitute an important element in web model checking. The usage of the proposed operators is not limited to web or verification only, previously we discussed the use of similar operators in the context of automated planning [6, 7].

The rest of this paper is organized as follows. In Section 2, we describe how we identify the scope of states of interest in our modeling approach of web applications. In Section 3, we describe a variant of LTL, 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 4 describes our solution, namely the syntax and semantics of the new operators. In Section 5, we illustrate the proposed technique with several examples of web related properties. In Section 6 we discuss the related work. We conclude in Section 7.

ASE'05, November 7–11, 2005, Long Beach, California, USA.

2. SCOPES IN WEB ANALYSIS

In the model of a given WA, a stable state represents a stable display where a page is fully loaded in the browser window. In case of a web application with frames, a page is loaded in each frame. When all the pages in the frames (in the browser window) are completely loaded, the display (as well as the state) of the web application becomes stable; otherwise, the display (state) is unstable and is in a transient mode. The reason is that when, in a given window, a request for a page with frames is sent to the server, the response is a frameset document containing URIs of the frames' source pages. Then, the browser, without the intervention of the user, initiates the requests for the source pages of the frames. These requests cannot be initiated simultaneously, and their response pages are not loaded at the same time. At this point, a user action can interleave with these requests and responses. This includes, for example, clicking a link in one frame while the source page in the second frame is not yet fully loaded. Such scenario is considered in our web analysis framework [4, 5], where the frameset document is a state (representing the transient state of the display) in the automaton of the window and the browser initiated requests are looping rendezvous events in the automata of the corresponding frames. The communicating automata model is used to represent web applications where properties are verified over the whole global state space defined by composition of the automata of the different entities. On the other hand, there exist properties of web applications that should be verified by considering only a subset of the states. Examples of such properties are given in Section 6.



(a) **(b)** **(c)**

Figure 1. a) A_1 for *Browser Window*, b) A_2 for *Frame1*, c) A_3 for *Frame2*

To define a way of distinguishing states, we consider the example of a web application with two frames. In Figure 2, we show a fragment of the model which has three entities, the *browser window*, *Frame1*, and *Frame2*. These entities are modeled by three automata, A_1, A_2, and A_3, respectively, which communicate by common events. Initially, the three automata are in their inactive states s_0, u_0, and v_0, respectively. The event a is a link, clicked by the user, which makes A_1 to move to state s_1 that is the frameset document containing URIs of *Frame1* and *Frame2*. The events f_1 and f_2 are from the browser window received by the two frames, respectively, represent the browser triggered requests for frames source pages. A_2 and A_3 are then active, while A_1 remains in s_1. In *Frame1*, the user can click the link b so that A_1 moves to state u_2 by executing the transition labeled by action b. In *Frame2*, the user can click the link c whose target is _top, such that the corresponding page is loaded in the full window, thus canceling the two frames. In this case, c is a multi-rendezvous of A_3, A_1, and A_2; as a result, A_2 and A_3 move to their inactive states u_0 and v_0, and A_1 moves to state s_2. Note that different possible behaviors of the web application that can occur in reality are represented in the model as well. For example, if the server is slow, *Frame2* can be activated before *Frame1* and the user can click on c before the browser triggers the request for the source page of *Frame1*. Similarly, the link b can be clicked before *Frame2* becomes active. These possible behaviors can be seen in the composition of the automata shown in Figure 2.

In the example, the states that correspond to the "completed" stabilized web application displays can be distinguished from the "uncompleted" transient ones, and eventually treated differently in web model checking. In the web application, once the request for the page with two frames is sent to the server, the display is considered unstable until both frames are active, or the two frames are canceled and replaced by another page in the browser window. Therefore, the stable states of the model are (s_0,u_0,v_0), (s_1,u_1,v_1), (s_1,u_2,v_1), and (s_2,u_0,v_0), while the remaining states are transient (unstable). Note that the designated unstable states (Figure 2) have f_1 and/or f_2 as actions labeling outgoing transitions. These actions correspond to the requests triggered by the browser. Thus, we conclude that any global state with at least one enabled browser triggered action is transient (unstable).

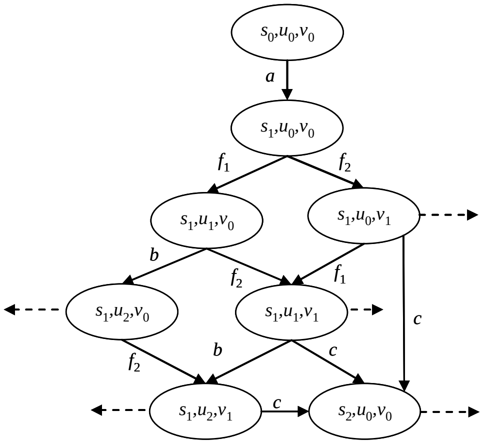


Figure 2. Composition of A_1, A_2, and A_3

Here, we propose a mechanism to identify stable and unstable states that is recognizable by a model checker (Spin). To distinguish states in our modeling framework, we introduce a Boolean variable to each automaton of a given web application model; we call it the *stability* variable (flag). If the state has at least one enabled event which represents a browser triggered action, then the state is a transient state and the flag takes the value zero. If the state's enabled actions represent only user triggered actions, then the state is stable and the flag takes the value one. Then we say that a global state of the model is stable if and only if the stability variables of all its component states are all set to one; otherwise, the global state of the model is transient. In this context, the stable states of a given web application constitute the scope of certain properties to be validated in the model.

3. LINEAR TEMPORAL LOGIC

The propositional linear temporal logic (PLTL) or simply LTL, extends traditional propositional logic with temporal operators [2, 9, 12]. Formally, an LTL formula φ has the following syntax:

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

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 $\circ$) is the *next* operator. LTL semantics was originally defined [12] over infinite sequences of states that correspond to infinite or non-terminating sequences of computations. A more recent variant of LTL semantics that applies both to infinite and finite sequences is developed in the runtime verification context [15]. We favor this variant of the semantic to account for finite scopes, which could arise in web analysis. Note that this variant may differ from the classical one in boundary case.

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. $\pi \vDash p \Leftrightarrow |\pi| > 0$, and $p \in L(s_0)$,
2. $\pi \vDash \neg\varphi \Leftrightarrow \pi \nvDash \varphi$,
3. $\pi \vDash \varphi \wedge \psi \Leftrightarrow \pi \vDash \varphi$ and $\pi \vDash \psi$,
4. $\pi \vDash \mathrm{X}\ \varphi \Leftrightarrow |\pi| > 0$, and $\pi^1 \vDash \varphi$,
5. $\pi \vDash \mathrm{G}\ \varphi \Leftrightarrow$ for all i, $0 \leq i < |\pi|$, $\pi^i \vDash \varphi$,
6. $\pi \vDash \mathrm{F}\ \varphi \Leftrightarrow$ for some i, $0 \leq i < |\pi|$, $\pi^i \vDash \varphi$,
7. $\pi \vDash \varphi\ \mathrm{U}\ \psi \Leftrightarrow$ there exists an i, $0 \leq i < |\pi|$ such that $\pi^i \vDash \psi$, and for all j, $0 \leq j < i$, $\pi^j \vDash \varphi$.

Unlike the classical definition of LTL [12], our definition, inspired by [15], takes into account both infinite and finite cases. Note that for the case of $\pi = \varepsilon$, $\pi \vDash \mathrm{F}\varphi$, and $\pi \vDash \mathrm{X}\ \varphi$ do not hold. If $|\pi| = 1$, then $\pi \vDash \mathrm{X}\ \varphi$ does not hold. Thus, X is defined to be strong (existential). A Kripke structure satisfies a given formula φ ($M \vDash \varphi$) if and only if for every path π of M, $\pi \vDash \varphi$.

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]. In particular, the challenge increases when expressing non-trivial properties which are relevant only to a subset of the states of a system. A suitable approach might be to add 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 their validity in the remaining states. There are several motivations and means to partition the state space of a system's behavior. One could consider partitioning states into stable and transient (as in our formal framework discussed in Section 3), or *final* and *intermediate* [11, 14]. The distinction between states could be used to express various levels of granularity at which the behavior of the system is described.

A straightforward solution to property specification over partitioned state spaces 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 the projection of a given Kripke structure onto the subset of states that are of interest as follows.

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*

1. $T' = \{(s_i, s_k) \mid s_i, s_k \in S'$, *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* $s_{i+1}, \ldots, s_{k-1} \notin S'\}$,
2. $S_0' = \{s_i \mid s_i \in S_0 \cap S'$ *or there exists a path* $\pi = \langle s_0, s_1, \ldots, s_{i-1}, s_i, \ldots \rangle$ of M, *such that* $s_0 \in S_0 \backslash S'$ *and* $s_0, \ldots, s_{i-1} \notin S'\}$, *and*
3. $L'(s) = L(s)$ *for all* $s \in S'$.

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

Then, a standard model checking algorithm [2] 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 not be applicable for model checkers, like Spin [8], which use Kripke representation internally, and where the user specifies a modular system in a high level language, such as Promela.

4. PROPOSITIONAL SCOPES IN LTL

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 out-of-scope states. For this purpose, we introduce new LTL operators that can be used to formulate properties in the in-scope states. We denote $\Im$ a propositional logic expression that valuates to True in every state where a given property should be verified. The set of states in which $\Im$ holds constitutes what we call $\Im$*-scope*. Since any LTL property is expressible with the $\neg$, $\wedge$, U, and X operators, we define the operators $\neg_\Im$ (*not in scope*), $\wedge_\Im$ (*and in scope*), $\mathrm{U}_\Im$ (*until in scope*), and $\mathrm{X}_\Im$ (*next in scope*), and use them to derive $\mathrm{F}_\Im$ (*eventually in scope*) and $\mathrm{G}_\Im$ (*always in scope*).

Let $\Im$ be a propositional logic expression, the operators $\neg_\Im$, $\wedge_\Im$, $\mathrm{U}_\Im$, $\mathrm{X}_\Im$, $\mathrm{F}_\Im$, *and* $\mathrm{G}_\Im$ *are as follows:*

1. $\neg_\Im\ \varphi = \neg\ \Im\ \mathrm{U}\ (\neg\varphi \wedge \Im)$
2. $\wedge_\Im\ \psi = \neg\Im\ \mathrm{U}\ ((\varphi \wedge \psi) \wedge \Im)$
3. $\varphi\ \mathrm{U}_\Im\ \psi = (\Im \rightarrow \varphi)\ \mathrm{U}\ (\psi \wedge \Im)$
4. $\mathrm{X}_\Im\ \varphi = \neg\Im\ \mathrm{U}\ [\Im \wedge \mathrm{X}\ (\neg\Im\ \mathrm{U}\ (\Im \wedge \varphi))]$
5. $\mathrm{F}_\Im\ \varphi = \mathrm{F}\ (\varphi \wedge \Im)$
6. $\mathrm{G}_\Im\ \varphi = \mathrm{G}\ (\Im \rightarrow \varphi)$

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

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

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

The semantics of the auxiliary in-scope operators follows directly from their definitions and LTL semantics:

1. $\pi \vDash p\ \mathbf{In}\ \Im \Leftrightarrow$ *there exists an i*, $0 \le i < |\pi|$, *such that* $\pi^i \vDash p$ *and* $\pi^i \vDash \Im$, *and for all j*, $0 \le j < i$, $\pi^j \nvDash \Im$.
2. $\pi \vDash \neg_{\Im}\ \varphi \Leftrightarrow$ *there exists an i*, $0 \le i < |\pi|$, *such that* $\pi^i \nvDash \varphi$ *and* $\pi^i \vDash \Im$, *and for all j*, $0 \le j < i$, $\pi^j \nvDash \Im$.
3. $\pi \vDash \varphi\ \mathrm{U}_{\Im}\ \psi \Leftrightarrow$ *there exists an i*, $0 \le i < |\pi|$, *such that* $\pi^i \vDash \psi$ *and* $\pi^i \vDash \Im$, *and for all j*, $0 \le j < i\ \pi^j \nvDash \Im$ *or* $\pi^j \vDash \varphi$.
4. $\pi \vDash \mathrm{X}_{\Im}\ \varphi \Leftrightarrow$ *there exist i, k*, $0 \le i < k \le |\pi|$, *such that* $\pi^i \vDash \Im$, $\pi^k \vDash \mathfrak{I}$ *and* $\pi^k \vDash \varphi$, *and for all j, l*, $0 \le j < i < l < k$, $\pi^j \nvDash \Im$ *and* $\pi^l \nvDash \Im$.
5. $\pi \vDash \mathrm{F}_{\Im}\ \varphi \Leftrightarrow$ *for some i*, $0 \le i < |\pi|$, $\pi^i \vDash \varphi$ *and* $\pi^i \vDash \Im$.
6. $\pi \vDash \mathrm{G}_{\Im}\ \varphi \Leftrightarrow$ *for all i*, $0 \le i < |\pi|$, *where* $\pi^i \vDash \Im$, $\pi^i \vDash \varphi$.

Finally, we state a theorem in which we show that if a formula (φ **In** $\Im$) holds in a given model (including in-scope and out-of-scope states), the corresponding LTL formula φ must hold in the projection of the model that includes only the in-scope states.

Theorem 1. *Let $\Im$ be a propositional logic formula, and let M and $M_{\Im}$ be two Kripke structures, $M_{\mathfrak{I}}$ is the projection of M onto set of all states of M in which $\Im$ is true. For any LTL formula φ, $M \vDash \varphi$* **In** *$\Im$ if and only if $M_{\Im} \vDash \varphi$.*

The theorem could be proved by induction over the formula depth. For detailed proof we refer the reader to [6, 7].

5. WEB PROPERTIES WITH SCOPES

We illustrate our approach and show its effectiveness by providing examples of web related properties taken from [16, 17] that are fine grained with propositional scopes that not only designate stable and transient states, but also in general scopes that designate states of interest identified by a proposition.

The following property is related to e-commerce web applications and states the following:

1. *Promotions of certain products are only present either on the Home page or on Shopping pages and their number does not exceed 2.*

This property is to ensure that a promoted product is not oversold. Using standard LTL, the property is written as follows:

$$\mathrm{G}\ (\ ((\neg \mathit{Home} \wedge \neg \mathit{Shopping}) \rightarrow (\mathit{Promotions} = 0)) \wedge ((\mathit{Home} \vee \mathit{Shopping}) \rightarrow (\mathit{Promotions} \le 2))\ ) \qquad (1.1)$$

The states that concern the property are the ones that designate the home page and the shopping pages. Therefore, the scope of states is the propositional formula (*Home* ∨ *Shopping*). Using the scope operator the property becomes more intuitive to write as follows:

$$\mathrm{G}(((\mathit{Promotions} \le 2)\ \mathbf{In}\ \ (\mathit{Home} \vee \mathit{Shopping})) \vee (\mathit{Promotions} = 0)) \qquad (1.2)$$

The following property is another example that relates to web applications that comprise highly secure information.

2. *The User visits Authentication page then Secure page then returns to Authentication and does this exactly twice.*

In this property, the *User* is able to visit a certain secure page which he is allowed to visit only twice and every time with authentication information. Therefore, given the stated property, we are not interested in the in any other pages other than *Authentication* page and *Secure* page. Using standard LTL, the generalized property is non trivial and tricky to specify:

$$(\neg \mathit{Authentication} \wedge \neg \mathit{Secure})\ \mathrm{U}\ (\mathit{Authentication} \wedge \neg \mathit{Secure} \wedge \mathrm{X}\ (\neg \mathit{Secure}\ \mathrm{U}\ (\mathit{Secure} \wedge \neg \mathit{Authentication} \wedge \mathrm{X}\ (\neg \mathit{Authentication}\ \mathrm{U}\ (\mathit{Authentication} \wedge \neg \mathit{Secure} \wedge \mathrm{X}\ (\neg \mathit{Secure}\ \mathrm{U}\ (\mathit{Secure} \wedge \neg \mathit{Authentication} \wedge \mathrm{X}\ (\neg \mathit{Authentication}\ \mathrm{U}\ (\mathit{Authentication} \wedge \neg \mathit{Secure} \wedge \mathrm{X}\ (\mathrm{G}\ (\neg \mathit{Secure}))))))))))) \qquad (2.1)$$

If we use now the **In** operator, the scope of the property includes states where the user can be in *Authentication* or in *Secure* pages. Therefore, the scope can be written as: (*Authentication* ∨ *Secure*). Now, the property can be written, in a more intuitive and succinct way, with the **In** operator as follows:

$$\mathit{Authentication} \wedge \mathrm{X}(\mathit{Secure} \wedge \mathrm{X}\ (\mathit{Authentication} \wedge \mathrm{X}\ (\mathit{Secure} \wedge \mathrm{X}\ (\mathit{Authentication} \wedge \mathrm{G}\ (\neg \mathit{Secure})))))\ \mathbf{In}\ (\mathit{Authentication} + \mathit{Secure}) \qquad (2.2)$$

Assume that the same property has to be verified in a web application that include frames and where only stable states have to be checked. Without using the **In** operator, the formula in (2.1) would be even more complex if we take into account stable states only, while if we use **In** operator, then we reuse formula (2.2) and modify the scope as follows:

$$\mathit{Authentication} \wedge \mathrm{X}\ (\mathit{Secure} \wedge \mathrm{X}\ (\mathit{Authentication} \wedge \mathrm{X}(\mathit{Secure} \wedge \mathrm{X}\ (\mathit{Authentication} \wedge \mathrm{G}\ (\neg \mathit{Secure})))))\ \mathbf{In}\ ((\mathit{Authentication} + \mathit{Secure}) \wedge \mathit{stable}) \qquad (2.3)$$

The proposition *stable* is the conjunction of the stability Boolean variables of the model's corresponding component automata.

6. RELATED WORK

Manna and Wolper [10] propose the so-called relativization rules in the context of synthesis of communicating processes. Relativisation takes a PTL specification of a single process and transforms it into a specification of the global system. Contrary to our approach, the relativization procedure they propose applies only to infinite sequences and scopes, moreover atomic propositions of a given formula are assumed to hold only in the scope. More recently, Eisner et al [18] introduced the so-called temporal clock to LTL. The clock allows a formula to be checked only when the clock ticks. While their defined semantics are similar to the ones we introduce for the scope operator, they assume that clocks do not accumulate. In other words, they assume that states that belong to a certain propositional scope, cannot belong to a different scope at the same time. This limitation in their approach rises from the context in which they address the problem where hardware clocks do not overlap. 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. 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 expression is valid and not to the first state of the execution path. Dwyer et al [3] propose the specification pattern system, which has five scopes each with a start and end state. However, the authors did not address the problem of specifying these patterns in a scope of arbitrary set of states. Also, there is no methodology to impose the scopes on arbitrary formula.

7. CONCLUSION AND FUTURE WORK

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 and constitute what we call in-scope states. 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. This problem was addressed in the context of web application model checking, where some properties may concern certain subsets of states. We showed the effectiveness of our approach and usefulness in specifying web properties more succinctly and intuitively. Although we demonstrated the usefulness of the proposed LTL scope operators in our formal framework for analysis and validation of web applications, the scope operators can be applied in the verification process of a wider range of applications. We applied our scope operators on the patterns provided by Dwyer and the results can be found in [6, 7]. Our future work will focus on the generalization of these results on temporal scopes, and mixed state/event properties, that could contribute to the elucidation of existing specification patterns [13].

8. ACKNOWLEDGMENTS

The first author acknowledges stimulating discussions with Gerard Holzmann.

9. 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] M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[3] 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.

[4] 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, Madrid, Spain, September 2004.

[5] M. Haydar, "Formal Framework for Automated Analysis and Verification of Web-based Applications", *in Proc. of the 19th IEEE Int. Conference on Automated Software Engineering (ASE 2004)*. Linz, Austria, September 20-24, 2004.

[6] M. Haydar, S. Boroday, A. Petrenko, and H. Sahraoui, "Adding Propositional Scopes to Linear Temporal Logic", Technical Report [CRIM 05/05-06**]**, Centre de Recherche Informatique de Montreal, May 2005.

[7] M. Haydar, S. Boroday, A. Petrenko, and H. Sahraoui, "Propositional Scopes in Linear Temporal Logic", *in Proc. of the 5th Int. Conference on Novelles Technologies de la Repartition (NOTERE 2005)*. Gatineau, Canada, August 30-September 1, 2005.

[8] G. J. Holzmann. *The Spin Model Checker, Primer and Reference Manual*. Addison-Wesley, 2003.

[9] M.R.A. Huth, and M.D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.

[10] Z. Manna, and P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications", *in ACM Transactions on Programming Languages and Systems*, Vol. 6, No. 1, January 1984, pp. 68-93.

[11] 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.

[12] A. Pnueli, "The Temporal Logic of Programs", *in Proc. of the 18th IEEE Symposium on Foundations of Computer Science*, 1977, pp. 46-57.

[13] 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.

[14] C.H. West, "An Automated Technique of Communication Protocols Validation", *IEEE Trans. on Comm.*, Vol. 26, pp. 1271-1275, 1978.

[15] 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.

[16] Web Design Guidelines, IBM, http://www-306.ibm.com/ibm/easy/eou_ext.nsf/publish/611.

[17] F. Millerand, O. Martial, "Guide pratique de conception et d'évaluation ergonomique de sites Web", Montréal, Centre de recherche informatique de Montréal, 2001. [CRIM-01/08-21].

[18] C. Eisner, D. Fisman, J. Havlicek, A. McIsaac, and D. Van Campenhout, "The Definition of a Temporal Clock Operator", *in Proc. of 30th Int. Colloquium on Automata Languages and Programming (CALP 2003)*, LNCS, vol. 2719, pp. 857-870, Eindhoven, The Netherlands, June 2003.