Improving Pattern-Based LTL Formulas for Automata Model Checking

Salamah Salamah
Dept. Comp. and Software Engineering, Embry-Riddle Aeronautical Univ., salamahs@erau.edu
Ann Q. Gates, Steve Roach
Computer Science Dept., University of Texas at El Paso, {agates,sroach}@utep.edu

Abstract

The Property Specification (Prospec) tool uses patterns and scopes defined by Dwyer et. al., to generate formal specifications in Linear Temporal Logic (LTL) and other languages. The work presented in this paper provides improved LTL specifications for patterns and scopes over those originally provided by Prospec. This improvement comes in the efficiency of the LTL formulas as measured in terms of the number of states in the Büchi automaton generated for the formula. Minimizing the size of the Büchi automata for an LTL specification provides a significant support to the area of model checking.

Key Words- LTL, Büchi Automaton, Pattern, Scope, Composite Propositions, Prospec.

1 Introduction

The process of model checking a software system consists of developing a model of the system to be verified and writing specifications in a temporal logic such as Linear Temporal Logic (LTL) [7] or Computational Tree Logic (CTL). In automata-based model checking, both the model M and the complement of the temporal specification S are represented by a special type of state machine called Büchi Automaton (BA) [1]. To check the consistency of M with S, the model checker calculates the intersection of M and S' where S' is the complement of S. If the intersection is empty, then M is consistent with S. In other words, if M and S' each represent a set of specifications and if $M \cap S' = \varnothing$, then the system satisfies the specification; otherwise, the system is inconsistent with the specification and a counter example is returned.

The process of writing formal specifications is not easy because of the required mathematical sophistication and depth of knowledge in the specification language. For this reason, tools that simplify the creation of formal specifications in logics such as LTL are of interest to the model checking community and others. In the case of automata-based model checkers such as Spin [6], it is important that these tools generate efficient formulas, since the model checker complements the formulas, translates the result into a BA, and intersects the BA with the automaton of the system. The size of the automaton that results from the intersection of two automata has as its upper bound the product of the number of states in each of the two. One way to avoid the problem of state space explosion is to minimize the number of states generated by the negation of the specification. This reduces the number of states generated by the automaton of the intersection, and as a result, it reduces the time required to model check a system.

The Property Specification tool (Prospec) [8, 9, 10] builds on the Specification Patterns System (SPS) [3], and it uses property patterns and scopes to assist in the specification of formal properties in LTL as well as other languages. Patterns are high-level abstractions that provide descriptions of common properties, and scopes describe the extent of program execution over which the property holds. Prospec also introduces the notion of composite propositions to allow for the definition of more complex behavior to represent the limits for patterns and scopes.

This paper introduces more efficient LTL formulas for patterns and scopes than those originally generated by Prospec. In defining the new formulas, we tried to limit the number of temporal operators in a formula because this tends to reduce the number of states in the BA. The formulas are compared in terms of the number of states in the BA generated by the negation of these formulas, since it is actually the complement of a formula that the model checker uses. To generate BAs, we used the model checker Spin, and the LTL to BA translators: LTL2BA [11], the Temporal Massage Parlor [4], and LTL2NBA [5], all of which efficiently convert LTL specifications into BAs. This paper also describes the impact that more efficient formulas have when using composite propositions [9] to define pattern and scope limits.

The paper first presents a background on LTL, BA, and the Prospec tool. Section 3 describes the new formulas and the process used to verify the semantic equivalence of the

978-0-7695-3099-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ITNG.2008.228

two sets of formulas. Section 3 also provides the results of the comparisons of the formulas. The impact of the work on the use of composite propositions is presented in Section 4 followed by brief discussion, and the references.

2 Background

2.1 Linear Temporal Logic

LTL is a prominent formal specification language that is highly expressive and widely used in formal verification tools such as the model checkers SPIN [6] and NUSMV [2]. LTL is also used in the runtime verification of Java programs [15].

Formulas in LTL are constructed from elementary propositions and the usual Boolean operators not, and, or, and $imply$ (neg, $\wedge$, $\vee$, $\rightarrow$, respectively). In addition, LTL provides the temporal operators $next$ (X), $eventually$ $(\diamond)$, $always$ $(\Box)$, $until$, (U), $weak\ until$ (W), and $release$ (R). In this work, we only use the first four of these operators. These formulas assume discrete time, i.e., the execution of a program can be represented by a sequence of states $s = 0, 1, 2, \ldots$ The formula XP holds at state s if P holds at the next state $s + 1$. $P\,U\,Q$ is true at state s if there is a state $s' \geq s$ at which Q is true and, if s' is such a state, then P is true at all states s_i for which $s \leq s_i < s'$. The formula $\diamond P$ is true at state s if P is true at some state $s' \geq s$. Finally, the formula $\Box P$ holds at state s if P is true at all moments of time $s' \geq s$.

2.2 Büchi Automata

LTL model checking is based on a variation of the classic theory of finite automata [6]. While a finite automaton accepts only terminating executions, model checking requires a different type of machine that can handle executions that might not terminate. Such machines are necessary to model non-terminating systems such as operating systems, traffic lights, or ATMs. One such machine is a Büchi automaton. A Büchi automaton (BA) is a tuple (Q,$\sum$, δ,Q_0, F) where:

- Q is a finite set of states
- $Q_0 \subseteq Q$ is a set of initial states
- $\sum$ is an alphabet
- δ: Q x $\sum \rightarrow 2^Q$ is a transition function and
- $F \subseteq Q$ is a set of accepting states

An execution is a sequence s_0, s_1, ..., where $\forall\ i\ s_i \in Q$ and $\forall\ i > 0$, (s_i, s_{i+1}) $\in \delta$. A finite execution is an accepting execution if it terminates in a final state $s_f \in F$. An infinite execution, also called an w-execution or w-run, is accepting if it passes through a state $s_f \in F$ infinitely often. An empty BA (accepts no words) is one that either terminates in a state that is not an accepting state or has no accepting state that is visited infinitely often that is reachable from the initial state. The set of executions accepted by a BA is called the language of the BA.

Languages of BAs represent a superset of those of LTL; every LTL formula can be represented by a BA. When a BA is generated from an LTL formula, the language of the BA represents only the traces accepted by the LTL formula. For example the BA in Figure 1 represents the language accepted by the LTL formula (a U b). This formula specifies that b holds in the initial state of the computation or a holds in the current state and (a U b) holds in the next state. The language of the BA in Figure 1 accepts the set of traces {b..., ab..., aab..., aaaaaab...}. Notice that each of these traces passes through the accepting state $Final$. This state is both reachable from the initial state and is visited infinitely often (by virtue of the self transition marked 1).

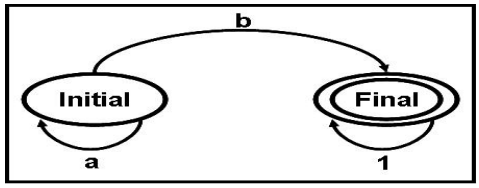

Figure 1.
BA for "a U b"

This paper compares three LTL to BA translators (along with the Spin model checker) in the number of states in the BA generated from those LTL formulas for patterns and scopes. The translators are: Temporal Massage Parlor (TMP) [4], LTL2BA [11], and LTL2NBA [5].

2.3 Prospec

The Property Specification tool (Prospec) [8, 9, 10] builds on the Specification Patterns System (SPS) [3] by facilitating the identification of SPS patterns and scopes as well as validation of specifications. SPS defines patterns and scopes to assist the practitioner in formally specifying recurring software properties. The main patterns defined by SPS are: $Universality$, $Absence$, $Existence$, $Precedence$, and $Response$.

In SPS, each pattern is associated with a *scope* that defines the extent of program execution over which a property pattern is considered. There are five scopes defined in SPS: $Global$, $Before\ R$, $After\ L$, $Between\ L\ And\ R$, and $After\ L\ Until\ R$. A detailed description of these patterns and scopes provide by Dwyer et. al., [3].

Prospec displays traces of computation to illustrate the subtle issues that exist within the different patterns and scopes, and it displays a decision tree to guide the user in selecting the appropriate pattern and/or scope. Given a com-

Table 1.
Summary of characteristics for the Response pattern in Prospec

PATTERN	CHARACTERISTICS
T $Responds$ $to\ P$	1) P must be followed by T, where P and T are events or conditions 2) Some T follows each time that P holds 3) The same state at which T holds may follow two or more states at which P holds 4) T may hold at the same state as P holds 5) If T holds, P may or may not hold at a previous state 6) The $response$ property represents a cause-effect relation 7) If cause P holds, at some future state effect T holds 8) T responds to P is also known as T follows P

Table 2.
Summary of characteristics for the Before R scope in Prospec

SCOPE	CHARACTERISTICS
$Before$ R	1) The scope denotes a subsequence of states or events (an interval) that begins with the start of computation and ends with the state or event immediately preceding the event or state at which R holds for first time in the computation. 2) The interval does not include the state or event associated with R. 3) The interval defined by the scope occurs once in a computation. 4) One or more events (conditions) may be associated with R; a condition is a proposition and an event is a change in value of the proposition from one state to the next.

putation represented as a sequence of states and a finite set of events, E, a trace of computation is a list indicating, for each moment of time t, which events from the set E occur at t.

Prospec enhances the definition of patterns and scope characteristics by explicitly defining the relationships among the proposition that define a pattern and/or a scope and the boundaries defined by each scope. For the lack of space, Tables 1 and 2 give only sample characteristics of the $Response$ pattern and the $Before\ R$ scope. The characteristics for the remaining patterns and scopes can be found in [10].

Prospec extends SPS by introducing a classification for defining sequential and concurrent behavior. This is accomplished through the notion of composite propositions (CP) [9]. Section 4 discusses composite propositions in details.

While Prospec builds on SPS in defining the mapping of patterns and scopes into LTL, it makes some changes to those LTL formulas for patterns and scopes defined in SPS. Salamah et. al [14] provides a listing of those modifications and the justification behind the changes. Table 3 presents Prospec's original LTL mappings for each pattern and scope

Table 3.
Prospec's original LTL formulas for pattern and scope

PATTERN/SCOPE	LTL Formula
$Absence$, $After\ L$	$\neg(L)W(L \wedge \neg(\diamond P))$
$Existence$, $Before\ R$	$\diamond R \rightarrow (\neg(R)U(P \wedge \neg(R)))$
$Existence$, $After\ L$	$\neg(L)W(L \wedge (\diamond P))$
$Existence$, $Between\ L$, R	$\Box((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(R)U(P \wedge \neg(R))))$
$Universality$, $After\ L$	$\neg(L)W(L \wedge \Box P)$
$Universality$, $After\ L\ U\ R$	$\Box((L \wedge \neg(R)) \rightarrow (PWR))$
$Precedence$, $Global$	$\neg(P)WT$
$Precedence$, $After\ L$	$\neg(L)W(L \wedge (\neg(P)W(T)))$
$Precedence$, $After\ L\ U\ R$	$\Box(L \wedge \neg(R) \rightarrow (\neg(P)W(T \vee R)))$
$Response$, $Before\ R$	$\diamond R \rightarrow ((P \rightarrow (\neg(R)U(T \wedge \neg(R))))UR)$
$Response$, $After\ L$	$(\neg L)W(L \wedge \Box(P \rightarrow \diamond T))$
$Response$, $Between\ L$, R	$\Box((L \wedge \neg(R) \wedge \diamond R) \rightarrow (P \rightarrow (\neg(R)U(T \wedge \neg(R))))UR$
$Response$, $After\ L\ U\ R$	$\Box((L \wedge \neg(R) \rightarrow (P \rightarrow (\neg(R)U(T \wedge \neg(R))))WR)$
$S-Precedence$, $Global$	$\neg(P)W(T \wedge \neg(P))$
$S-Precedence$, $After\ L$	$\neg(L)W(L \wedge (\neg(P)W(T \wedge \neg(P))))$
$S-Precedence$ $Between\ L$, R	$\Box((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(P)U((T \wedge \neg(P)) \vee R)))$
$S-Precedence$ $After\ L\ U\ R$	$\Box(L \wedge \neg(R) \rightarrow (\neg(P)W((T \wedge \neg(P)) \vee R)))$

combination being compared in this paper[1][2]. These formulas along with the ones defined in Section 3 (Table 4) are the ones being compared in this paper[3].

3 New Pattern Formulas

The goal of the work presented by this paper is to improve the efficiency of the LTL specifications generated by Prospec. The efficiency is measured by the number of states in the BA corresponding to the negation of LTL specification. The BAs were produced using the Spin model checker Version 4.2.7, and the LTL to BA translation tools LTL2BA, LTL2NBA, and Temporal Massage Parlor (TMP), all of which are available at a website maintained by Carsten Fritz [16].

The approach is to reduce the number of states in the BA by reducing the number of temporal operators within each formula. This was achieved in 17 out of 30 formulas originally defined by Prospec. Table 4 lists the new improved formulas, and Table 5 lists the results of the comparison

[1]Table 3 only shows those original formulas (17 formulas) that we were able to change to more efficient ones.

[2]In Tables 3-5 in the pattern/scope column, the term After L U R indicates that the scope is After L Until R. Similarly, the term Between L, R indicates that the scope is Between L and R

[3]Comparing the new formulas in Section 3 with the original SPS formulas showed that the new formulas were at least as efficient in all cases with the exception of the case of the $Response$ pattern within the $After\ L\ Until\ R$ scope in which SPS formula produced one fewer state in the never claim as produced by LTL2BA, LTL2NBA, and TMP.

Table 4.

Prospec's new LTL formulas for pattern and scope

PATTERN/SCOPE	LTL Formula
$Absence\,,After\,L$	$\neg((\neg L)U(L \wedge \diamond P))$
$Existence\,,Before\,R$	$\neg((\neg P)UR)$
$Existence\,,After\,L$	$\neg((\neg L)U(L \wedge \neg \diamond P))$
$Existence\,,Between\,L\,,R$	$\Box((L \wedge \neg R) \rightarrow (\neg((\neg P)UR)))$
$Universality\,,After\,L$	$\neg((\neg L)U(L \wedge \diamond \neg P))$
$Universality\,,After\,L\,U\,R$	$\Box((L \wedge \neg R) \rightarrow (\neg((P \wedge \neg R)U((\neg P) \wedge \neg R))))$
$Precedence\,,Global$	$\neg((\neg T)U(P \wedge \neg T))$
$Precedence\,,After\,L$	$\neg((\neg L)U(L \wedge ((\neg T)U(P \wedge \neg T))))$
$Precedence\,,After\,L\,U\,R$	$\Box((L \wedge \neg R) \rightarrow (\neg(((\neg T) \wedge \neg R)U(P \wedge (\neg T) \wedge \neg R)))$
$Response\,,Before\,R$	$\neg((\neg R)U(P \wedge (\neg R) \wedge ((\neg T)UR)))$
$Response\,,After\,L$	$\neg((\neg L)U(L \wedge (\neg\Box(P \rightarrow \diamond T))))$
$Response\,,Between\,L\,,R$	$\Box((L \wedge \neg R) \rightarrow \neg((\neg R)U(P \wedge (\neg R) \wedge ((\neg T)UR))))$
$Response\,,After\,L\,U\,R$	$\Box((L \wedge \neg R) \rightarrow \neg((\neg R)U(P \wedge (\neg R) \wedge ((\Box((\neg T) \wedge \neg R)) \vee ((\neg T)UR))))))$
$S-Precedence\,,Global$	$\neg((\neg(T \wedge \neg P))UP))$
$S-Precedence\,,After\,L$	$\neg((\neg L)U(L \wedge ((\neg(T \wedge \neg P))UP)))$
$S-Precedence$ $Between\,L\,,R$	$\Box((L \wedge \neg R) \rightarrow$ $(\neg(((\neg(T \wedge \neg P)) \wedge \neg R)U$ $(P \wedge (\neg(T \wedge \neg P)) \wedge (\neg R) \wedge \diamond R))))$
$S-Precedence$ $After\,L\,U\,R$	$\neg \diamond (L \wedge (\neg R) \wedge$ $(((\neg T) \wedge \neg R)U(P \wedge \neg R)))$

between Prospec's original formulas and the new ones. In all the cases where we were able to reduce the number of temporal operators, the BAs generated by Spin contained fewer states. Only in the case of the $Response$ pattern with an $After\ L\ Until\ R$ scope did the new formula generate more states when used by the three BA generators. All the translation tools produced BAs with the same number of states for each new formula except in the cases of the $strict\ precedence$ with the $after\ L\ until\ R$ scope and $strict\ precedence$ with the $after\ L$. In this case, the generated BAs by LTL2BA, TMP, and LTL2NBA produced one less state than the one generated by Spin. This is significant, as it allows Spin users to use the new formulas without having to use the other translators.

3.1 Equivalence of LTL Formulas

In this work we compare two sets of LTL formulas in terms of efficiency. It is important that we verify that these sets of formulas are semantically equivalent. An approach to demonstrating the equivalence of two LTL formulas is to compare the languages of the BAs generated from them. If the languages are identical, then the two LTL formulas are equivalent. To show this, we show that the first language is a subset of the second, and that the second is a subset of the first.

Table 5.

Comparison of the number of states generated by the original and new Prospec formulas

PATTERN/SCOPE	SPIN Old, New	LTL2BA Old, New	LTL2NBA Old, New	TMP Old, New
Absence After L	14, 3	4, 3	3, 3	4, 3
Existence Before R	6, 2	2, 2	2, 2	2, 2
Existence After L	7, 2	3, 2	3, 2	3, 2
Existence Between L, R	7, 3	3, 3	3, 3	3, 3
Universality After L	14, 3	6, 3	3, 3	4, 3
Universality After L U R	7, 3	3, 3	3, 3	3, 3
Precedence Global	5, 2	2, 2	2, 2	2, 2
Precedence After L	29, 3	7, 3	4, 3	4, 3
Precedence After L U R	6, 3	3, 3	3, 3	3, 3
Response Before R	7, 3	3, 3	3, 3	3, 3
Response After L	22, 3	7, 3	3, 3	7, 3
Response Between L, R	8, 4	4, 4	4, 4	4, 4
Response After L U R	9, 5	4, 5	4, 5	4, 5
S-Precedence Global	6, 3	2, 2	2, 2	2, 2
S-Precedence After L	27, 4	7, 3	4, 3	4, 3
S-Precedence Between L,R	5, 4	4, 4	4, 4	4, 4
S-Precedence After L U R	7, 3	3, 3	3, 3	3, 3

Given two LTL formulas F_1 and F_2, let B_1=(Q_1,Σ,δ_1,Q_{01},F_1) be the BA for F_1 and B_2=(Q_2,Σ,δ_2,Q_{02},F_2) be the BA for F_2. Label each transition in δ_1 and δ_2 with the names of the propositions that are true in the state entered by the transition. Let Σ be the union of all the transition labels in B_1 and B_2. Let L_1 be the language accepted by B_1 and L_2 be the language accepted by B_2. To show that F_1 and F_2 are equivalent, we show that $L_1 \cap \neg L_2 \equiv L_2 \cap \neg L_1 \equiv \varnothing$ [4]

Given two formulas F_P and F_N, where F_P is the original Prospec formula and F_N is the new formula, we used LTL2BA[11] to generate a BA for ($F_P \wedge \neg F_N$) and a BA ($F_N \wedge \neg F_P$). We visually inspected the BAs to ensure they were empty, i.e., the intersection was empty. We did this both ways; generating a BA for the original Prospec formula and the negation of the new formula and generating another BA for the new formula and the negation of the original formula. In all cases, the BAs were verified (manually) to be empty.

By definition, a BA is empty if it does not contain a reachable accepting state that is visited infinitely often [1]. Note that if the BA is complex, then checking for emptiness may require automation. On the other hand, if the BA is fairly simple, then this can be done by visual inspection, which was the case in all BAs generated in this work. To illustrate, we show the equivalence of the original Prospec formula for the $Universality$ of P pattern within

[4] Since $L_1 \cap \neg L_2$ is empty, $\neg L_2$ must not contain any element of L_1. Since L_1 and L_2 are both subsets of Σ^*, L_2 must contain every element of L_1. Thus, $L_1 \subset L_2$. Similarly, $L_2 \subset L_1$. Thus, $L_2 \equiv L_1$. B_1 and B_2 are equivalent if they accept the same language, thus $B_1 \equiv B_1$, and $F_1 \equiv F_2$.

the $After\ L\ Until\ R$ scope and the new formula. First, we used LTL2BA to translate the LTL formula for original formula and the negation of the new formula into a BA. The resulting BA is shown in Figure 2. This BA obviously does not contain a cycle that passes through an accepting state. The only accepting state in the BA is $accept - S3$. Although this state is reachable from the initial, state it is not visited infinitely often (i.e., there is no cycle that contains this state). On the other hand, the BA of the new formula and the negation of Prospec's original formula for the $Universality$ of P pattern within the $After\ L\ Until\ R$ scope contains no accepting states, and hence is empty. Figure 3 in the shows the corresponding BA.

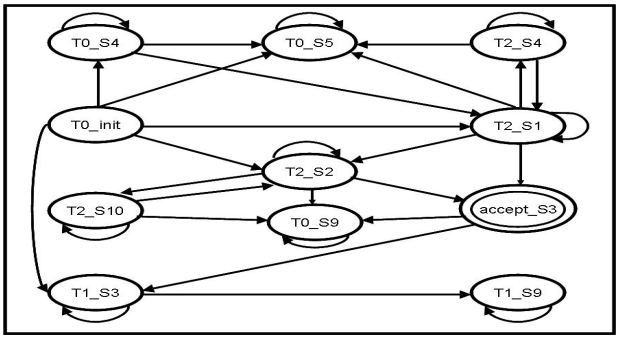

Figure 2.
BA for Original Prospec Formula and the Negation of the New Formula for the Universality Pattern Within the After L Until R Scope

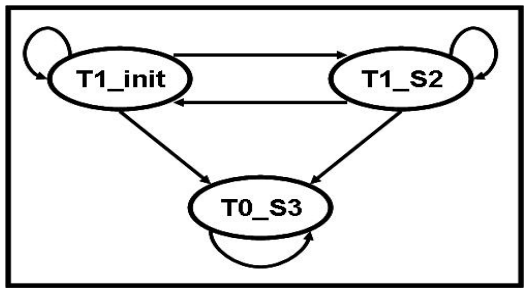

Figure 3.
BA for the New Formula and the Negation of the Original Prospec Formula for the Universality Pattern Within the After L Until R Scope

4 Impact on the Use of CP

Mondragon et al. [9, 10] introduced composite propositions (CPs) to define the structure of multiple propositions to capture sequential and concurrent behavior. The work provided a CP taxonomy that can be used in the property elicitation and specification process. The taxonomy guides practitioners in formally specifying properties, illuminating the subtleties associated with multiple events and conditions. When relations have not been carefully analyzed, composite propositions can expose incompleteness or ambiguities.

Table 6.
Description of Composite Proposition Classes in LTL

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

CP classes defined as conditions are used to describe concurrency, and those defined as events are used to describe activation or synchronization of processes or actions. Table 6 presents the semantics in LTL for the CP classes as introduced by Mondragon et. al.[9]. CPs can be used to define boundaries of scopes and patterns with multiple propositions. For instance, an ordered sequence can define the left boundary of an $after\ L$ scope, and multiple events can define the cause part of a response pattern.

The naíve use of CPs in LTL formulas can result in a state explosion when using a model checker like Spin. It is important to start with efficient LTL formulas such as the ones presented in Table 4 and to build on them when using CP, otherwise the BAs generated by the LTL formulas would be too large for the model checker.

For example, consider the following property of an Automated Teller Machine (ATM): "After a user selects a withdrawal transaction, user's account is updated, money is dispensed, receipt is printed, and ATM card is returned." Assume the following symbol assignments; $'w'$: Withdrawal transaction is selected, $'au'$: Account is updated, $'md'$: Money is dispensed, $'rp'$: Receipt is printed, and $'cr'$: Card is returned. This property can be described using the $Existence$ of P pattern within the $After\ L$ scope where P is a the CP class $Eventual_C$ $(au \wedge X(\neg md\, U\, (md \wedge X(\neg rp\, U\, (rp \wedge X(\neg cr\, U\, cr))))))$ and L is w. Using direct

substitution in the original Prospec formula to specify this property the generated formula is $(\Box\neg w) \vee ((\neg w)U(w \wedge (au \wedge X(\neg md\, U\, (md \wedge X(\neg rp\, U\, (rp \wedge X(\neg cr\, U\, cr))))))))$. Running this formula in Spin produces a BA with 28 states, while running it in LTL2BA, TMP, and LTL2NBA produces BAs with 9 states. On the other hand, by direct substitution in the new formula, then the generated LTL formula is: $\neg((\neg w)U(w \wedge \Box\neg(au \wedge X(\neg md\, U\, (md \wedge X(\neg rp\, U\, (rp \wedge X(\neg cr\, U\, cr))))))))$. When this formula is run in Spin, the number of states in the generated BA is 8, and using LTL2BA, TMP, and LTL2NBA the number is 5. As shown in Table 5, the new formula for the $Existence$ of P pattern within the $After\ L$ scope when used by Spin generates a BA with 5 fewer states than that generated using the original formula, and 1 fewer state when used by the LTL2BA, LTL2NBA and TMP. However, when CPs are used to specify P, the difference in the number of states is 20 when using Spin and 4 when using the other three translators.

The above example shows that the difference in the number of states in the BA can be significant when using CPs. This is especially true considering that in order to model check a model against this specification, the model checker calculates a product formula with size up to the product of the number of states produced by the specification and the number of states in the BA of the model.

5 Summary

Tools that assist in the generation of formal specifications in LTL are important to the model checking community as they relieve the user from the burden of writing specifications in a language that is hard to read and write. Without the help of tools such as Prospec, the user might create faulty specifications. As well as providing specifications that correspond to the intent of the user[14], these tools must generate efficient formulas, since one of the main challenges of model checking is the state explosion problem. The smaller the size of the automata an LTL formula generates the less likely that this problem will occur. Although this might not appear significant in the basic pattern-scope formulas as generated originally by Prospec, an example given in the previous section shows the effect on the number of states generated when using less efficient formulas as the base formulas when incorporating CPs. This work provided a more effecient pattern-based LTL formulas when compared in terms of the number of states in the BA they generate, compared to the Original formulas provided by Prospec.

Using the Prospec's new LTL formulas, we developed techniques for generating LTL specifications for pattern, scope, and CP combinations. The details of that work is provided by Salamah et. al. [12, 13]

References

[1] Clarke, O. Grumberg, and D. Peled. Model Checking. MIT Publishers, 1999.

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

[3] Dwyer, M. B., G. S. Avrunin, and J. C. Corbett, "Patterns in Property Specification for Finite–State Verification," *Proceedings of the 21st Intl. Conference on Software Engineering*, Los Angeles, CA, USA, 1999, 411–420.

[4] Etessami, K., and Holzmann, G. "Optimizing Büchi automata", *Proceedings of 11th International Conference on Concurrency Theory* 2000.

[5] Fritz, C., "Constructing Büchi Automata from Linear Temporal Logic Using Simulation Relations for Alternating Büchi Automata," *Eighth Conference on Implementation and Application of Automata* 2003.

[6] Holzmann, G. J. The Spin Model Checker: Primer and Reference Manual. Addisson Wesley, 2004.

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

[8] 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.), *Proc. of Runtime Verification Workshop, ENTCS*, 89(2), 2004.

[9] 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.

[10] Mondragon, O., "Elucidation and Specification of Software Properties through Patterns and Composite Propositions to Support Formal Verification Techniques," Dissertation, The University of Texas at El Paso, May 2004.

[11] Oddoux, D., and Gastin, P., "Fast LTL to Bchi Automata Translation," *13th International Conferenc on Computer Aided Verification, CAV*, Jul. 2001.

[12] Salamah, S., Gates, A., Kreinovich, V., Roach, S. "Using Patterns and Composite Propositions to Automate the Generation of Complex LTL Specifications", Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis ATVA'2007, Tokyo, Japan, October 22-25, 2007 (to appear).

[13] Salamah, S., Kreinovich, V., and Gates, A., "Generating Linear Temporal Logic Formulas for Pattern-Based Specifications", Proceedings of the Nineteenth International Conference on Software Engineering and Knowledge Engineering SEKE'07, Boston, Massachusetts, July 9-11, 2007, pp. 422-427.

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

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

[16] LTL2NBA, http://www.ti.informatik.uni-kiel.de/ fritz/ABA-Simulation/ltl.cgi, September 2006.