Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2011, Article ID 869182, 13 pages
doi:10.1155/2011/869182

Research Article

Towards Support for Software Model Checking: Improving the Efficiency of Formal Specifications

Salamah Salamah,[1] Ann Q. Gates,[2] Steve Roach,[2] and Matthew Engskow[1]

[1] *Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University (ERAU), Daytona Beach, FL 32114, USA*
[2] *Department of Computer Science, University of Texas at El Paso (UTEP), El Paso, TX 79968, USA*

Correspondence should be addressed to Salamah Salamah, salamahs@erau.edu

Received 10 December 2010; Accepted 3 March 2011

Academic Editor: Phillip Laplante

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 improvement for model checking software systems using such tools as the highly acclaimed Spin model checker.

1. Introduction

The process of model checking a 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) [1] or Computational Tree Logic (CTL) [2]. 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 a Büchi Automaton (BA) [3]. 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 $M \cap S' = \emptyset$, 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 [4], 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 classical problem of state space explosion is to minimize the number of states generated by the negation of the specification. This will reduce the number of states generated by the automaton of the intersection, and as a result, it will reduce the time required to model check a software system.

The Property Specification (Prospec) [5–7] builds on the Property Specification Patterns system (SPS) [8, 9], and it uses property pattern and scope 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 behaviors 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 we believe this reduces the number of states in the never-claim. The formulas are compared in terms of the number of states in the never-claim generated by the negation of these formulas since it is actually the complement of a formula that the model checker uses. To generate Büchi automata, we used the model checker SPIN, and the LTL to BA translators: LTL2BA [10], the Temporal Message Parlor [11], and LTL2NBA [12], all of which efficiently convert LTL specifications into BAs. Notice that some of these tools produce a BA in the form of a never-claim, which is a specific representation of BA used by the model checker Spin. In this article we use the terms Büchi automata and never-claims interchangeably. This paper also shows an approach for proving the equivalence of different LTL formulas. Finally, this paper describes the impact that more efficient formulas have when using composite propositions [6] to define pattern and scope limits.

The paper first presents a background on LTL and Büchi automata (BA), including the semantics of the languages. The Prospec tool is introduced in Section 3. Section 4 describes the new formulas and the process used to verify the semantic equivalence of the two sets of formulas. Section 4 also provides the results of the comparisons of the formulas. Finally, the impact of the work on composite propositions is presented in Section 5 followed by brief discussion and references.

2. Background

2.1. Linear Temporal Logic. This section briefly describes Linear Temporal Logic (LTL) and its semantics. A more detailed description of LTL can be found in Manna [13].

Temporal Logic has been used to verify concurrent systems. There are three well used types of temporal Logic: Linear Temporal Logic (LTL), Computational Tree Logic (CTL), and CTL* [3]. Both LTL and CTL are subsets of CTL*. While CTL allows for branching type of behavior (i.e., semantically, formulas are defined both using the universal and existential quantifiers), LTL formulas describe specific path behavior (i.e., semantically, formulas are defined only using the universal quantifier). Formulas in this paper deal only with LTL.

LTL is used in various model checkers such as SPIN [4], NUSMV [14], and Java Path-Finder [15] and is also used in runtime verification of Java programs [16].

In LTL, a temporal formula is constructed inductively from a set of propositions P by applying Boolean connectives $\neg$ and $\vee$ and temporal operators *next* (X) and *until* (U) as follows.

(i) A proposition is a temporal formula.

(ii) If p and q are temporal formulas, then the followings are also temporal formulas:

  (i) $\neg p$,

  (ii) $p \vee q$,

  (iii) Xp,

  (iv) pUq.

A temporal formula that makes use only of Boolean connectives $\neg$ and $\vee$ is called a *state formula*, whereas a formula that makes use of temporal operators X and U is called a *temporal formula*.

Let P be a set of propositions; let σ be an infinite sequence of states or computation, denoted by σ: $s_0, s_1, \ldots$, let $\sum$ be a set of states; let $I(s)$ be an interpretation such that for all $s \in \sum$, $I(s) \subseteq P$ specifies the propositions that are true in state s. For a state formula, the satisfaction relation $\models$ is defined as follows: for a state $s \in \sum$ and a proposition p in P, s satisfies p, denoted $s \models p$, if and only if $p \in I(s)$. In addition, an inductive definition for the notion that a computation σ satisfies formula p at $i > 0$, denoted $(\sigma, i) \models p$, follows [1, 13]:

(i) $(\sigma, i) \models p$ if and only if $s_i \models p$, where p is a state formula,

(ii) $(\sigma, i) \models \neg p$ if and only if $(\sigma, i) \not\models p$,

(iii) $(\sigma, i) \models p \vee q$ if and only if $(\sigma, i) \models p$ or $(\sigma, i) \models q$,

(iv) $(\sigma, i) \models Xp$ if and only if $(\sigma, i+1) \models p$,

(vi) $(\sigma, i) \models pUq$ if and only if $(\sigma, k) \models q$ for some $k \geq i$, and $(\sigma, j) \models p$ for all $j, i \leq j < k$.

Additionally, the temporal operators (eventually, always, and weak-until) are derived as follows:

(i) $\diamond p \equiv trueUp$,

(ii) $[]p \equiv \neg \diamond \neg p$,

(iii) $pWq \equiv []p \vee (p\ U\ q)$.

2.2. Büchi Automata. Classical LTL model checking is based on a variation of the classic theory of finite automata [4]. While a finite automaton accepts only terminating executions, model checking requires a different type of machines that can handle executions that might not terminate. Such machines are necessary to model nonterminating systems such as operating systems, traffic lights, or ATMs. One such machine is a Büchi automaton. A Büchi automaton (BA) is a tuple $(Q, \sum, \delta, Q_0, F)$, where

(i) Q is a finite set of states,

(ii) $Q_0 \subseteq Q$ is a set of initial states,

(iii) $\sum$ is an alphabet,

(iv) $\delta : Q \times \sum \to 2^Q$ is a transition function,

(v) $F \subseteq Q$ is a set of accepting states.

An execution is a sequence $s_0, s_1, \ldots$, where for all $is_i \in Q$ and for all $i \geq 0$, $(s_i, s_{i+1}) \in \delta$. 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

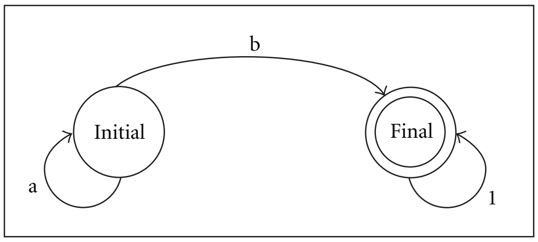


Figure 1: BA for “$a \cup b$”.

in a state that is not an accepting state or has no accepting state that is reachable from the initial state and that is visited infinitely often. 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 ($a \cup b$). This formula specifies that b holds in the initial state of the computation, or a holds until b holds. The language of the BA in Figure 1 accepts the set of traces {b..., ab..., aab..., ..., aaab}. 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).

Various work has been done on the translation of LTL to BA to reduce the number of states in the resulting BA and to speed up the process of the BA generation. 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.

(i) Temporal Message Parlor (TMP) [11] is the work of Kousha Etessami at Lucent Technologies.

(ii) LTL2BA [10] is the work of Denis Oddoux and Paul Gastin at the University of Paris 7, France.

(iii) LTL2NBA [12] is the work of Carsten Fritz of the Department of Computer Science at Christian-Albrechts University.

3. Prospec

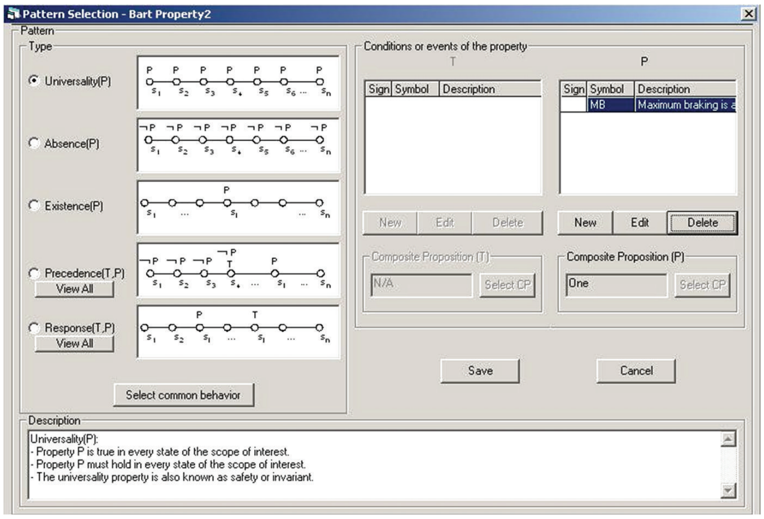
The Property Specification tool (Prospec) [5–7] builds on the Specification Patterns System (SPS) [8, 9] by facilitating the identification of SPS patterns and scopes as well as validation of specifications. SPS defines patterns and scopes to assist the practitioner in formally specifying software properties. Patterns capture the expertise of developers by describing solutions to recurrent problems [17]. Each pattern describes the structure of specific behavior, defines the pattern's relationship with other patterns, and defines the scope over which the property holds.

The main patterns defined by SPS are universality, absence, existence, precedence, and response. *Universality P* states that property P is true at every point of the execution; *absence P* states that P is never true during the execution; *existence P* states that P is true at some point in the execution; *precedence* (T, P) states that P holds before T holds; *response* (P, T) states that if P holds, then T must hold at a future state. Response properties represent a temporal relation called cause-effect between two propositions. 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 through a series of decisions in selecting the appropriate pattern and/or scope. Given a computation 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. Figure 2 shows the Prospec window for selecting a pattern and the traces of computation that are given to elucidate each pattern. Prospec enhances the definition of patterns and scope characteristics provided in the SPS website [18] by explicitly defining the relationships among the proposition that define a pattern and/or a scope and the boundaries defined by each scope. Tables 1 and 2 give those characteristics for pattern and scope, respectively, as they appear in the Prospec tool. These are the characteristics that were used by Salamah et al. [19] to verify the correctness of the LTL formulas for patterns and scopes. Prospec extends SPS by introducing a classification for defining sequential and concurrent behavior. This is accomplished by including composite propositions (CPs) as shown in Figure 3. Section 4 discusses composite propositions in more detail. 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. (Comparing the new formulas in Section 4 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). Salamah et al. [19] provide a listing of those modifications and the justification behind the changes. Table 3 presents Prospec's LTL mappings for each pattern and scope combination. These formulas along with the ones defined in Section 3 are the ones being compared in this paper.

Note that Prospec added the definition of *strict precedence*, which is not available in SPS. This pattern describes the situation where S *strictly precedes* P, where S and P are events or conditions. Unlike the regular *precedence* pattern, S and P cannot hold at the same state in this pattern. Use of *Strict Precedence* enforces that S and P cannot hold at the same state.

4. New Pattern Formulas

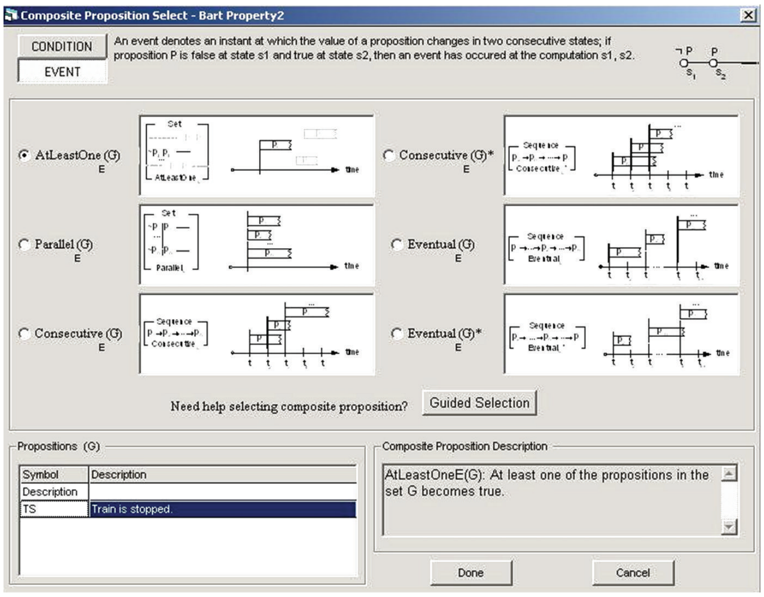
The goal of the work presented by this paper is to improve the efficiency of the LTL specifications generated by Prospec. The efficiency is measured by the number of states in the BA corresponding to the negation of LTL specification. The BAs were produced using the SPIN model checker Version 4.2.7, and the LTL to BA translation tools LTL2BA, LTL2NBA, and the Temporal Message Parlor (TMP), all of which are available at a website maintained by Carsten Fritz [20].



Sign	Symbol	Description
	MB	Maximum braking is a

Universality(P):
- Property P is true in every state of the scope of interest.
- Property P must hold in every state of the scope of interest.
- The universality property is also known as safety or invariant.

Figure 2: Prospec's pattern screen.



An event denotes an instant at which the value of a proposition changes in two consecutive states; if proposition P is false at state s1 and true at state s2, then an event has occured at the computation s1, s2.

Symbol	Description
Description	
TS	Train is stopped.

AtLeastOneE(G): At least one of the propositions in the set G becomes true.

Figure 3: Prospec's composite proposition screen.

In order to reduce the number of states in the BA, our goal is to reduce the number of temporal operators within each formula. We were able to achieve this 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 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 case of the strict precedence with the *after L until R* scope. In this case, the generated BAs by LTL2BA, TMP, and LTL2NBA produced one fewer 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 to produce never-claims that have to be inserted into the Promela code.

4.1. Equivalence of LTL Formulas. 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.

Given two LTL formulas F_1 and F_2, let $B_1 = (Q_1, \Sigma, \delta_1, Q_{01}, F_1)$ be the BA for F_1 and let $B_2 = (Q_2, \Sigma, \delta_2, Q_{02}, F_2)$ be the BA for F_2. Label each transition in δ_1 and δ_2 with the

Table 1: Summary of characteristics for patterns in Prospec.

Pattern	Characteristics
	(1) Event or condition P does not hold within the states defined by the scope of interest
	(2) The *absence* property is also known as alarm
Existence of (P)	(1) Event or condition P holds at least once within the states defined by the scope of interest
	(2) The *existence* property is also known as eventually
Universality of (P)	(1) Event or condition P holds in every state of the scope of interest
	(2) The *universality* property is also known as safety or invariant
(T) Precedes (P)	(1) T holds before P holds, where T and P are events or conditions
	(2) T may hold several times before P holds
	(3) P does not hold before T holds
	(4) P may hold at the same state as T
	(5) If T holds, then P may or may not hold
	(6) If T holds, then T may or may not hold when P holds
	(7) The *precedence* property represents a cause-effect relation, where T denotes a cause and P denotes an effect
	(8) There is no effect P without a cause T
	(9) T precedes P is also known as T before P
(T) Strictly Precedes (P)	(1) T holds before P holds, where T and P are events or conditions
	(2) T may hold several times before P holds
	(3) P does not hold before T holds
	(4) P does not hold at the same state at which T holds
	(5) If T holds, then P may or may not hold
	(6) If T holds, then T does not hold when P holds
	(7) The *precedence* property represents a cause-effect relation, where T denotes a cause and P denotes an effect
	(8) There is no effect P without a cause T
	(9) T precedes P is also known as T before P
(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, then P may or may not hold at a previous state
	(6) The *response* property represents a cause-effect relation, where P denotes a cause and T denotes an effect
	(7) If cause P holds, then at some future state effect T holds
	(8) T responds to P is also knows as T follows P

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 let L_2 be the language accepted by B_2. To show that F_1 and F_2 are equivalent, we show that $L_1 \cap \neg L_2 \equiv L_2 \cap \neg L_1 \equiv \emptyset$. (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 $B_1 \equiv B_2$, and $F_1 \equiv F_2$.)

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 to generate a BA_1 for $(F_P \wedge \neg F_N)$ and a BA_2 for $(\neg F_P \wedge F_N)$. To assert that $F_P \equiv F_N$ we have to check that BA_1 and BA_2 are both empty. To do this we developed a simple computer tool that checks the emptiness of a BA.

4.1.1. Tool for Checking Emptiness. By definition a BA is empty if it does not contain a reachable accepting state that is visited infinitely often [3]. The LTL Validation Tool is designed to ensure that a given LTL formula is valid (i.e., its BA contains at least one reachable accepting state that is visited infinitely often). In this work, the tool is used to prove the equivalence of two formulas. As discussed above we show equivalence of two LTL formulas F_P and F_N by checking that resulting BAs for $(F_P \wedge \neg F_N)$ and $(\neg F_P \wedge F_N)$ are both empty.

This validation tool makes use of LTL2BA's Java interface (JLTL2BA) to translate the formulas $(F_P \wedge \neg F_N)$ and

Table 2: Summary of characteristics for scopes in Prospec.

Scope	Characteristics
Global	(1) The scope denotes the entire computation
	(2) The scope includes all the states in the computation
	(3) The interval defined by the scope occurs once in a computation
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
After *L*	(1) The scope denotes a subsequence of states or events (an interval) that begins with the first event or state at which *L* holds and ends with termination of computation
	(2) The interval includes the state or event associated with *L*
	(3) The interval defined by the scope occurs once in a computation
	(4) One or more events (conditions) may be associated with *L*; a condition is a proposition and an event is a change in value of the proposition from one state to the next
Between *L* and *R*	(1) The scope denotes a subsequence of states or events (an interval) that begins when *L* holds and ends with the state or event immediately preceding the event or state at which *R* holds
	(2) Event or condition *L* must hold and, at a different event or state in the future, *R* must hold
	(3) The interval includes the state or event associated with *L*
	(4) The interval does not include the state or event associated with *R*
	(5) The interval defined by the scope may occur more than once in a computation
	(6) Multiple intervals may be defined within an interval when *L* holds more than once before *R* holds
	(7) One or more events (conditions) may be associated with *L* and *R*
After *L* Until *R*	(1) The scope denotes a subsequence of states or events (an interval) that begins when *L* holds and ends either with the state or event immediately preceding the event or state at which *R* holds, or begins when *L* holds and ends with the termination of computation
	(2) The interval includes the state or event associated with *L*
	(3) The interval does not include the state or event associated with *R*
	(4) The interval may repeat during a computation
	(5) If *L* holds and *R* does not hold, the interval ends with termination of a computation
	(6) The interval defined by the scope may occur more than once in a computation
	(7) Multiple intervals may be defined within an interval when *L* holds more than once before *R* holds
	(8) One or more events (conditions) may be associated with *L* and *R*

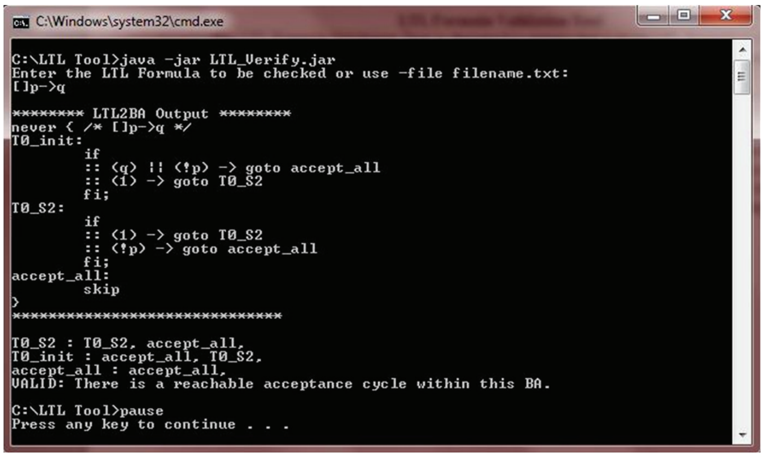
$(\neg F_P \wedge F_N)$ into the corresponding BAs. Once the BAs are available, the tool tries to assert that there are no reachable accepting states that are visited infinitely often. This is done by first finding all states reachable from every given state, then by linking these together to form cycles. Should the tool find a single acceptance cycle, that is, an acceptance state which is within a cycle then we declare that the BA is not empty.

Figure 4 shows the never-claim generated by the LTL Validation tool (through interfacing with LTL2BA) for the LTL formula "$[]p \rightarrow q$" and the result generated by the tool to check for the emptiness of the BA (never-claim). Obviously the BA for this formula is not empty, and as a result the tool states that the formula is Valid (i.e., there is a reachable acceptance state that is within a cycle). Figure 5, on the other hand, shows the never-claim and the tool's result for the formula $(\neg([]((l \wedge \neg r) \rightarrow (([]p) \vee (pUr)))) \wedge ([]((l \wedge \neg r) \rightarrow (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r)))))$. This formula is the result of ANDing the negation of Prospec's original LTL formula for Universality of *P* within the *After L Until R* scope with the new LTL formula for the same pattern/scope combination. The result returned by the tool specify that there is no reachable acceptance state that is within a cycle (i.e., the BA is empty). Figure 6 shows the graph of the never-claim in Figure 5.

Similarly, we used the LTL Validation tool to generate the BA for the LTL formula of ANDing Prospec's original LTL formula for Universality of *P* within the *After L Until R* scope with the negation of new LTL formula for the same pattern/scope combination. Figure 7 shows the graph

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

Pattern	Scope	LTL Formula
Absence	Global	$\neg(\diamond P)$
	Before R	$\diamond R \rightarrow \neg(\neg(R) U P)$
	After L	$\neg(L) W (L \wedge \neg(\diamond P))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow \neg(\neg(R) U P))$
	After L Until R	$[](L \wedge \neg(R) \rightarrow \neg(\neg(R) U P))$
Existence	Global	$(\diamond P)$
	Before R	$\diamond R \rightarrow (\neg(R) U (P \wedge \neg(R)))$
	After L	$\neg(L) W (L \wedge (\diamond P))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(R) U (P \wedge \neg(R))))$
	After L Until R	$[]((L \wedge \neg(R)) \rightarrow (\neg(R) U (P \wedge \neg(R))))$
Universality	Global	$[]P$
	Before R	$\diamond R \rightarrow (P U R)$
	After L	$\neg(L) W (L \wedge []P)$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (P U R))$
	After L Until R	$[]((L \wedge \neg(R)) \rightarrow (P W R))$
Precedence	Global	$\neg(P) W T$
	Before R	$\diamond R \rightarrow (\neg(P) U (T \vee R))$
	After L	$\neg(L) W (L \wedge (\neg(P) W (T)))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(P) U ((T \vee R))))$
	After L Until R	$[](L \wedge \neg(R) \rightarrow (\neg(P) W (T \vee R)))$
Response	Global	$[](P \rightarrow \diamond T)$
	Before R	$\diamond R \rightarrow ((P \rightarrow (\neg(R) U (T \wedge \neg(R)))) U R)$
	After L	$(\neg L) W (L \wedge [](P \rightarrow \diamond T))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (P \rightarrow (\neg(R) U (T \wedge \neg(R)))) U R$
	After L Until R	$[]((L \wedge \neg(R) \rightarrow (P \rightarrow (\neg(R) U (T \wedge \neg(R)))) W R)$
Strict Precedence	Global	$\neg(P) W (T \wedge \neg(P))$
	Before R	$\diamond R \rightarrow (\neg(P) U ((T \wedge \neg(P)) \vee R))$
	After L	$\neg(L) W (L \wedge (\neg(P) W (T \wedge \neg(P))))$
	Between L and R	$[]((L \wedge \neg(R) \wedge \diamond R) \rightarrow (\neg(P) U ((T \wedge \neg(P)) \vee R)))$
	After L Until R	$[](L \wedge \neg(R) \rightarrow (\neg(P) W ((T \wedge \neg(P)) \vee R)))$



```
C:\LTL Tool>java -jar LTL_Verify.jar
Enter the LTL Formula to be checked or use -file filename.txt:
[]p->q

******** LTL2BA Output ********
never { /* []p->q */
T0_init:
        if
        :: (q) || (!p) -> goto accept_all
        :: (1) -> goto T0_S2
        fi;
T0_S2:
        if
        :: (1) -> goto T0_S2
        :: (!p) -> goto accept_all
        fi;
accept_all:
        skip
}
******************************

T0_S2 : T0_S2, accept_all,
T0_init : accept_all, T0_S2,
accept_all : accept_all,
VALID: There is a reachable acceptance cycle within this BA.

C:\LTL Tool>pause
Press any key to continue . . .
```

Figure 4: LTL validation tool's output for the formula $[]p \rightarrow q$.

Table 4: Prospec's new LTL formulas.

Pattern	Scope	LTL Formula
Absence	After L	$\neg((\neg L)U(L \wedge \diamond P))$
	After L Until R	$[](L \wedge \neg(R) \rightarrow \neg(\neg(R)UP))$
Existence	Before R	$\neg((\neg P)UR)$
	After L	$\neg((\neg L)U(L \wedge \neg \diamond P))$
	Between L and R	$[]((L \wedge \neg R) \rightarrow (\neg((\neg P)UR)))$
Universality	After L	$\neg((\neg L)U(L \wedge \diamond \neg P))$
	After L Until R	$[]((L \wedge \neg R) \rightarrow (\neg((P \wedge \neg R)U((\neg P) \wedge \neg R))))$
Precedence	Global	$\neg((\neg T)U(P \wedge \neg T))$
	After L	$\neg((\neg L)U(L \wedge ((\neg T)U(P \wedge \neg T))))$
	After L Until R	$[]((L \wedge \neg R) \rightarrow (\neg(((\neg T) \wedge \neg R)U(P \wedge (\neg T) \wedge \neg R)))$
Response	Before R	$\neg((\neg R)U(P \wedge (\neg R) \wedge ((\neg T)UR)))$
	After L	$\neg((\neg L)U(L \wedge (\neg [](P \rightarrow \diamond T))))$
	Between L and R	$[]((L \wedge \neg R) \rightarrow \neg((\neg R)U(P \wedge (\neg R) \wedge ((\neg T)UR))))$
	After L Until R	$[]((L \wedge \neg R) \rightarrow \neg((\neg R)U(P \wedge (\neg R) \wedge (([]((\neg T) \wedge \neg R)) \vee ((\neg T)UR))))))$
Strict Precedence	Global	$\neg((\neg(T \wedge \neg P))UP))$
	After L	$\neg((\neg L)U(L \wedge ((\neg(T \wedge \neg P))UP)))$
	Between L and R	$[]((L \wedge \neg R) \rightarrow (\neg(((\neg(T \wedge \neg P)) \wedge \neg R)U(P \wedge (\neg(T \wedge \neg P)) \wedge (\neg R) \wedge \diamond R))))$
	After L Until R	$\neg \diamond (L \wedge (\neg R) \wedge (((\neg T) \wedge \neg R)U(P \wedge \neg R)))$

Table 5: Comparison results 1.

Pattern/Scope	SPIN	LTL2NBA	TMP	LTL2BA
	Old, New	Old, New	Old, New	Old, New
Absence After L	14, 3	6, 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 and R	7, 3	3, 3	3, 3	3, 3
Universality After L	14, 3	4, 3	3, 3	4, 3
Universality After L Until R	6, 3	3, 3	3, 3	3, 3
Precedence Global	5, 2	2, 2	2, 2	2, 2
Precedence After L	29, 3	6, 3	4, 3	4, 3
Precedence After L Until 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	10, 3	3, 3	7, 3
Response Between L and R	8, 4	4, 4	4, 4	4, 4
Response After L Until 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	6, 3	4, 3	4, 3
S-Precedence Between L and R	5, 4	4, 4	4, 4	4, 4
S-Precedence After L Until R	7, 3	3, 3	3, 3	3, 3

of the never-claim for the formula $(([]((l \wedge \neg r) \rightarrow (([]p) \vee (pUr))))) \wedge \neg([]((l \wedge \neg r) \rightarrow (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r)))))$. 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). As expected, the LTL Validation tool returned the message "There's no reachable acceptance cycle within this BA". Since both the BAs for ANDing the negation of Prospec's original LTL with the new LTL formula and for ANDing Prospec's original LTL formula with the negation of the new LTL formula are both empty, we conclude that both formulas are equivalent.

In our work we performed the above mentioned procedure for all the new and original formulas in Tables 3 and 4, and the results showed in every case that the

Table 6: CP semantics in LTL (subscripts C and E stand for condition and event, resp.).

	CP Class	Semantics in LTL
(1)	AtLeastOne$_C$	$P_1 \vee P_2 \vee \cdots \vee P_n$
(2)	AtLeastOne$_E$	$(\neg P_1 \wedge \cdots \wedge \neg P_n) \wedge ((\neg P_1 \wedge \cdots \wedge \neg P_n)\ U\ (P_1 \vee \cdots \vee P_n))$
(3)	Parallel$_C$	$P_1 \wedge P_2 \wedge \cdots \wedge P_n$
(4)	Parallel$_E$	$(\neg P_1 \wedge \neg P_2 \wedge \cdots \wedge \neg P_n) \wedge ((\neg P_1 \wedge \neg P_2 \wedge \cdots \wedge \neg P_n) U (P_1 \wedge P_2 \wedge \cdots \wedge P_n))$
(5)	Consecutive$_C$	$P_1 \wedge X(P_2 \wedge X(\cdots \wedge X(P_n)) \cdots)))$
(6)	Consecutive$_E$	$(\neg P_1 \wedge \neg P_2 \wedge \cdots \wedge \neg P_n) \wedge ((\neg P_1 \wedge \neg P_2 \wedge \cdots \wedge \neg P_n) U ((P_1 \wedge \neg P_2 \wedge \neg P_3 \wedge \cdots \wedge \neg P_n) \wedge X((P_2 \wedge \neg P_3 \wedge \cdots \wedge \neg P_n) \wedge \cdots \wedge X(P_n) \cdots))$
(7)	Eventual$_C$	$P_1 \wedge (\neg P_2\ U\ (P_2 \wedge \cdots \wedge (\neg P_n\ U\ P_n)) \cdots)$
(8)	Eventual$_E$	$(\neg P_1 \wedge \cdots \wedge \neg P_n) \wedge ((\neg P_1 \wedge \cdots \wedge \neg P_n)\ U\ (P_1 \wedge ((\neg P_2 \wedge \cdots \wedge \neg P_n)\ U\ (P_2 \wedge (\cdots \wedge (P_{n-1} \wedge (\neg P_n\ U\ P_n)) \cdots)))))$
(9)	Strict Eventual$_C$	$P_1 \wedge X(\neg P_2\ U\ (P_2 \wedge \cdots \wedge X(\neg P_n\ U\ P_n)) \cdots)$
(10)	Strict Eventual$_E$	$(\neg P_1 \wedge \cdots \wedge \neg P_n) \wedge ((\neg P_1 \wedge \cdots \wedge \neg P_n)\ U\ (P_1 \wedge \neg P_2 \wedge \cdots \wedge \neg P_n \wedge ((\neg P_2 \wedge \cdots \wedge \neg P_n)\ U\ (P_2 \wedge \neg P_3 \wedge \cdots \wedge \neg P_n \wedge (\cdots \wedge (P_{n-1} \wedge \neg P_n \wedge (\neg P_n\ U\ P_n)) \cdots)))))$

Table 7: Comparison results 2.

Pattern/Scope	SPIN	LTL2NBA	TMP	LTL2BA
	Old, New	Old, New	Old, New	Old, New
Absence of P_4 After L	25, 4	10, 4	4, 4	6, 4
Absence of P_7 After L^*	26, 4	12, 4	17, 4	8, 4
Absence of P_9 After L^*	33, 5	12, 4	20, 4	8, 4
Absence of P After $L_4{}^*$	40, 6	15, 5	4, 5	14, 6
Absence of P After L_7^*	45, 6	14, 4	9, 4	10, 6
Absence of P After L_9^*	51, 8	14, 5	43, 4	9, 6
Existence of P_4 After L	15, 5	5, 3	5, 3	5, 3
Existence of P_7 After L	29, 9	7, 4	7, 4	9, 5
Existence of P_9 After L	29, 9	7, 4	7, 4	9, 5
Existence of P After L_4	23, 5	9, 4	6, 4	6, 5
Existence of P After L_7^*	22, 5	7, 3	11, 3	7, 5
Existence of P After L_9^*	27, 7	6, 3	9, 3	7, 5
Universality of P After L_4	40, 6	13, 5	4, 5	14, 6
Universality of P After L_7^*	44, 6	17, 4	9, 4	10, 6
Universality of P After L_9^*	51, 8	14, 4	45, 4	9, 6
Q Responds to P_4 After L^*	35, 7	19, 5	4, 4	11, 4
Q Responds to P_7 After L^*	20, 7	9, 6	14, 9	8, 6
Q Responds to P_9 After L^*	25, 8	8, 6	23, 9	8, 6
Q_4 Responds to P After L^*	32, 4	16, 4	4, 4	11, 4
Q_7 Responds to P After L	43, 10	15, 5	5, 5	20, 6
Q_9 Responds to P After L	65, 10	22, 5	5, 5	19, 6
Q Responds to P After L_4^*	63, 8	25, 5	4, 5	23, 6
Q Responds to P After L_7^*	68, 6	28, 4	15, 4	18, 6
Q Responds to P After L_9^*	104, 8	25, 4	6, 4	17, 6

new formula is semantically equivalent to the original one.

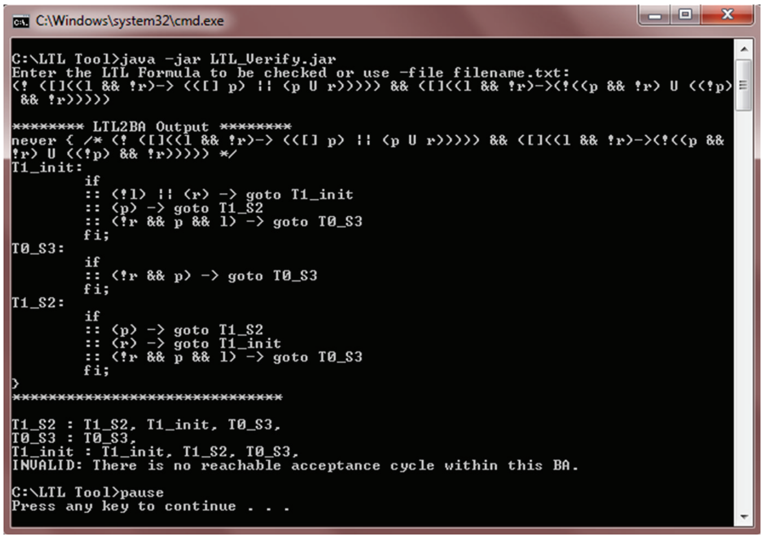
5. Impact on the Use of CP

Mondragon et al. [7, 21] introduced composite propositions (CPs) classes 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, CP classes can expose incompleteness or ambiguities.

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 of the CP classes in LTL.

Table 8: LTL formulas generated using the original Prospec's formulas.

	Formulas
(1)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge \neg \Diamond (((\neg p \wedge \neg pp \wedge \neg ppp) \wedge ((\neg p \wedge \neg pp \wedge \neg ppp)\ U\ (p \wedge pp \wedge ppp)))))))$
(2)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge \neg \Diamond ((p \wedge ((\neg pp)\ U\ pp))))))$
(3)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge \neg \Diamond ((p \wedge X(\neg pp\ U\ pp))))))$
(4)	$((\Box(\neg((\neg l \wedge \neg ll) \wedge ((\neg l \wedge \neg ll)\ U\ (l \wedge ll))))) \vee ((\neg((\neg l \wedge \neg ll) \wedge ((\neg l \wedge \neg ll)$ $U\ (l \wedge ll))))\ U\ ((((\neg l \wedge \neg ll) \wedge ((\neg l \wedge \neg ll)\ U\ ((l \wedge ll) \wedge \neg \Diamond (p))))))))$
(5)	$((\Box(\neg(l \wedge ((\neg ll)\ U\ ll)))) \vee ((\neg(l \wedge ((\neg ll)\ U\ ll)))\ U\ (((l \wedge ((\neg ll)\ U\ (ll \wedge \neg \Diamond (p))))))))$
(6)	$((\Box(\neg(l \wedge X(\neg ll\ U\ ll)))) \vee ((\neg(l \wedge X(\neg ll\ U\ ll)))\ \mathrm{U}\ (((l \wedge X(\neg ll\ U\ (ll \wedge \neg \Diamond (p))))))))$
(7)	$((\Box \neg l) \vee ((\neg l)\ U\ (l \wedge \Diamond(((\neg p \wedge \neg pp \wedge \neg ppp) \wedge ((\neg p \wedge \neg pp \wedge \neg ppp)\ U\ (p \wedge pp \wedge ppp)))))))$
(8)	$((\Box \neg l) \vee ((\neg l)\ U\ (l \wedge \Diamond((p \wedge ((\neg pp)\ U\ (pp \wedge ((\neg ppp)\ U\ ppp))))))))$
(9)	$((\Box \neg l) \vee ((\neg l)\ U\ (l \wedge \Diamond((p \wedge X(\neg pp\ U\ (pp \wedge X(\neg ppp\ U\ ppp))))))))$
(10)	$((\Box \neg(((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)\ U\ (l \wedge ll \wedge lll))))) \vee ((\neg((\neg l \wedge \neg ll \wedge \neg lll)$ $\wedge((\neg l \wedge \neg ll \wedge \neg lll)\ U\ (l \wedge ll \wedge lll))))\ U\ ((((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)\ U\ ((l \wedge ll \wedge lll) \wedge \Diamond p)))))))$
(11)	$((\Box \neg((l \wedge ((\neg ll)\ U\ ll)))) \vee ((\neg(l \wedge ((\neg ll)\ U\ ll)))\ U\ (((l \wedge ((\neg ll)\ U\ (ll \wedge \Diamond p)))))))$
(12)	$((\Box \neg((l \wedge X(\neg ll\ U\ ll)))) \vee ((\neg(l \wedge X(\neg ll\ U\ ll)))\ U\ (((l \wedge X(\neg ll\ U\ (ll \wedge \Diamond p)))))))$
(13)	$((\Box(\neg((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)\ U\ (l \wedge ll \wedge lll))))) \vee ((\neg((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)$ $U\ (l \wedge ll \wedge lll))))\ U\ ((((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)\ U\ ((l \wedge ll \wedge lll) \wedge \Box p)))))))$
(14)	$((\Box(\neg(l \wedge ((\neg ll)\ U\ ll)))) \vee ((\neg(l \wedge ((\neg ll)\ U\ ll)))\ U\ ((((l \wedge ((\neg ll)\ U\ (ll \wedge \mathrm{W}p))))))))$
(15)	$((\Box(\neg(l \wedge X(\neg ll\ U\ ll)))) \vee ((\neg(l \wedge X(\neg ll\ U\ ll)))\ U\ ((((l \wedge X(\neg ll\ U\ (ll \wedge \mathrm{W}p))))))))$
(16)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge (\Box((((\neg p) \wedge (\neg pp)) \wedge (((\neg p) \wedge (\neg pp))\ U\ (p \wedge pp))) \rightarrow (((\neg p) \wedge (\neg pp))$ $\wedge(((\neg p) \wedge (\neg pp))\ U\ (p \wedge pp \wedge \Diamond q))))))))$
(17)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge (\Box((p \wedge ((\neg pp)\ U\ (pp)))) \rightarrow (p \wedge ((\neg pp)\ U\ (pp \wedge \Diamond q)))))))$
(18)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge (\Box((p \wedge X((\neg pp)\ U\ (pp)))) \rightarrow (p \wedge X((\neg pp)\ U\ (pp \wedge \Diamond q)))))))$
(19)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge \Box(p \rightarrow \Diamond(((\neg q) \wedge (\neg qq)) \wedge (((\neg q) \wedge (\neg qq))\ U\ (q \wedge qq)))))))$
(20)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge \Box(p \rightarrow \Diamond(q \wedge ((\neg qq)\ U\ (qq \wedge ((\neg qqq)\ U\ qqq))))))))$
(21)	$((\Box(\neg l)) \vee ((\neg l)\ U\ (l \wedge \Box(p \rightarrow \Diamond(q \wedge X((\neg qq)\ U\ (qq \wedge X((\neg qqq)\ U\ qqq))))))))$
(22)	$((\Box(\neg(((\neg l) \wedge (\neg ll)) \wedge (((\neg l) \wedge (\neg ll))\ U\ (l \wedge ll))))) \vee ((\neg(((\neg l) \wedge (\neg ll)) \wedge (((\neg l) \wedge (\neg ll))\ U\ (l \wedge ll))))$ $U\ ((((\neg l) \wedge (\neg ll)) \wedge (((\neg l) \wedge (\neg ll))\ U\ (l \wedge ll \wedge \Box(p \rightarrow \Diamond q)))))))$
(23)	$((\Box(\neg(l \wedge ((\neg ll)\ U\ (ll))))) \vee ((\neg(l \wedge ((\neg ll)\ U\ (ll))))\ U\ ((l \wedge ((\neg ll)\ U\ (ll \wedge \Box(p \rightarrow \Diamond q)))))))$
(24)	$((\Box(\neg(l \wedge X((\neg ll)\ U\ (ll))))) \vee ((\neg(l \wedge X((\neg ll)\ U\ (ll))))\ U\ ((l \wedge X((\neg ll)\ U\ (ll \wedge \Box(p \rightarrow \Diamond q)))))))$



```
C:\LTL Tool>java -jar LTL_Verify.jar
Enter the LTL Formula to be checked or use -file filename.txt:
(! ([]((l && !r)-> (([] p) || (p U r))))) && ([]((l && !r)->(!((p && !r) U ((!p)
 && !r)))))

********* LTL2BA Output *********
never { /* (! ([]((l && !r)-> (([] p) || (p U r))))) && ([]((l && !r)->(!((p &&
!r) U ((!p) && !r))))) */
T1_init:
        if
        :: (!l) || (r) -> goto T1_init
        :: (p) -> goto T1_S2
        :: (!r && p && l) -> goto T0_S3
        fi;
T0_S3:
        if
        :: (!r && p) -> goto T0_S3
        fi;
T1_S2:
        if
        :: (p) -> goto T1_S2
        :: (r) -> goto T1_init
        :: (!r && p && l) -> goto T0_S3
        fi;
}
******************************

T1_S2 : T1_S2, T1_init, T0_S3,
T0_S3 : T0_S3,
T1_init : T1_init, T1_S2, T0_S3,
INVALID: There is no reachable acceptance cycle within this BA.

C:\LTL Tool>pause
Press any key to continue . . .
```

Figure 5: LTL validation tool's output for the formula $(\neg([]((l \wedge \neg r) \rightarrow (([]p) \vee (pUr))))) \wedge ([]((l \wedge \neg r) \rightarrow (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r)))))$.

Table 9: LTL formulas generated using the New Prospec's formulas.

	Formulas
(1)	$(\neg((\neg l)\ U\ (l \wedge \Diamond(((\neg p \wedge \neg pp \wedge \neg ppp) \wedge ((\neg p \wedge \neg pp \wedge \neg ppp)\ U\ (p \wedge pp \wedge ppp)))))))$
(2)	$(\neg((\neg l)\ U\ (l \wedge \Diamond((p \wedge ((\neg pp)\ U\ pp))))))$
(3)	$(\neg((\neg l)\ U\ (l \wedge \Diamond((p \wedge X(\neg pp\ U\ pp))))))$
(4)	$(\neg((\neg((\neg l \wedge \neg ll) \wedge ((\neg l \wedge \neg ll)\ U\ (l \wedge ll))))\ U\ ((((\neg l \wedge \neg ll) \wedge ((\neg l \wedge \neg ll)\ U\ ((l \wedge ll) \wedge \Diamond p)))))))$
(5)	$(\neg((\neg(l \wedge ((\neg ll)\ U\ ll)))\ U\ (((l \wedge ((\neg ll)\ U\ (ll \wedge \Diamond p)))))))$
(6)	$(\neg((\neg(l \wedge X(\neg ll\ U\ ll)))\ U\ (((l \wedge X(\neg ll\ U\ (ll \wedge \Diamond p)))))))$
(7)	$(\neg((\neg l)\ U\ (l \wedge \neg \Diamond\ (((\neg p \wedge \neg pp \wedge \neg ppp) \wedge ((\neg p \wedge \neg pp \wedge \neg ppp)\ U\ (p \wedge pp \wedge ppp)))))))$
(8)	$(\neg((\neg l)\ U\ (l \wedge \neg \Diamond\ ((p \wedge ((\neg pp)\ U\ (pp \wedge ((\neg ppp)\ U\ ppp))))))))$
(9)	$(\neg((\neg l)\ U\ (l \wedge \neg \Diamond\ ((p \wedge X(\neg pp\ U\ (pp \wedge X(\neg ppp\ U\ ppp))))))))$
(10)	$(\neg((\neg((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)\ U\ (l \wedge ll \wedge lll))))\ U\ ((((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)\ U\ ((l \wedge ll \wedge lll) \wedge \neg \Diamond p)))))))$
(11)	$(\neg((\neg(l \wedge ((\neg ll)\ U\ ll)))\ U\ (((l \wedge ((\neg ll)\ U\ (ll \wedge \neg \Diamond\ p)))))))$
(12)	$(\neg((\neg(l \wedge X(\neg ll\ U\ ll)))\ U\ (((l \wedge X(\neg ll\ U\ (ll \wedge \neg \Diamond\ p)))))))$
(13)	$(\neg((\neg((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)\ U\ (l \wedge ll \wedge lll))))\ U\ ((((\neg l \wedge \neg ll \wedge \neg lll) \wedge ((\neg l \wedge \neg ll \wedge \neg lll)\ U\ ((l \wedge ll \wedge lll) \wedge \Diamond \neg p)))))))$
(14)	$(\neg((\neg(l \wedge ((\neg ll)\ U\ ll)))\ U\ (((l \wedge ((\neg ll)\ U\ (ll \wedge \Diamond \neg p)))))))$
(15)	$(\neg((\neg(l \wedge X(\neg ll\ U\ ll)))\ U\ (((l \wedge X(\neg ll\ U\ (ll \wedge \Diamond \neg p)))))))$
(16)	$(\neg((\neg l)\ U\ (l \wedge \neg \Box((((\neg p) \wedge (\neg pp)) \wedge (((\neg p) \wedge (\neg pp))\ U\ (p \wedge pp)))\ \rightarrow\ (((\neg p) \wedge (\neg pp)) \wedge (((\neg p) \wedge (\neg pp))\ U\ (p \wedge pp \wedge \Diamond q)))))))$
(17)	$(\neg((\neg l)\ U\ (l \wedge \neg \Box((p \wedge ((\neg pp)\ U\ (pp))))\ \rightarrow\ (p \wedge ((\neg pp)\ U\ (pp \wedge \Diamond q))))))$
(18)	$(\neg((\neg l)\ U\ (l \wedge \neg \Box((p \wedge X((\neg pp)\ U\ (pp))))\ \rightarrow\ (p \wedge X((\neg pp)\ U\ (pp \wedge \Diamond q))))))$
(19)	$(\neg((\neg l)\ U\ (l \wedge \neg \Box(p\ \rightarrow\ \Diamond(((\neg q) \wedge (\neg qq)) \wedge (((\neg q) \wedge (\neg qq))\ U\ (q \wedge qq)))))))$
(20)	$(\neg((\neg l)\ U\ (l \wedge \neg \Box(p\ \rightarrow\ \Diamond(q \wedge ((\neg qq)\ U\ (qq \wedge ((\neg qqq)\ U\ qqq))))))))$
(21)	$(\neg((\neg l)\ U\ (l \wedge \neg \Box(\mathrm{p}\ \rightarrow\ \Diamond(q \wedge X((\neg qq)\ U\ (qq \wedge X((\neg qqq)\ U\ qqq))))))))$
(22)	$(\neg((\neg(((\neg l) \wedge (\neg ll)) \wedge (((\neg l) \wedge (\neg ll))\ U\ (l \wedge ll)))\ U\ ((((\neg l) \wedge (\neg ll)) \wedge (((\neg l) \wedge (\neg ll))\ U\ (l \wedge ll \wedge \neg \Box(p\ \rightarrow\ \Diamond q))))))))$
(23)	$(\neg((\neg(l \wedge ((\neg ll)\ U\ (ll))))\ U\ ((l \wedge ((\neg ll)\ U\ (ll \wedge \neg \Box(p\ \rightarrow\ \Diamond q)))))))$
(24)	$(\neg((\neg(l \wedge X((\neg ll)\ U\ (ll))))\ U\ ((l \wedge X((\neg ll)\ U\ (ll \wedge \neg \Box(p\ \rightarrow\ \Diamond q)))))))$

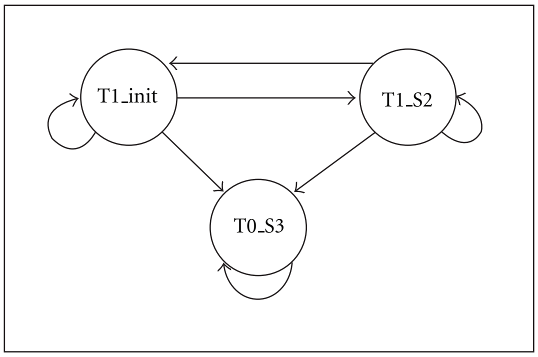


Figure 6: LTL2BA generated BA for $(\neg([]((l \wedge \neg r)\ \rightarrow\ (([]p) \vee (pUr)))) \wedge ([]((l \wedge \neg r)\ \rightarrow\ (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r)))))$.

CP classes 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 naïve use of CP classes 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 these CP classes; otherwise the BAs generated by the LTL formulas would be too large to handle by 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 the CP class $Eventual_C$. (*P* might also be of type *Strict* $Eventual_C$ depending on the specification). $((au \wedge (\Diamond(md \wedge (\Diamond(rp \wedge (\Diamond cr)))))))$, and L is w. Using direct substitution in the original Prospec formula to specify this property the generated formula is $((\Box\neg w) \vee ((\neg w)U(w \wedge \Diamond(au \wedge ((\neg md)U(md \wedge ((\neg rp)U(rp \wedge ((\neg cr)\ U\ cr)))))))))$. The negation of this formula produces a BA with

(i) 40 states using SPIN,

(ii) 9 states using LTL2NBA and TMP,

(iii) 17 states using LTL2BA.

On the other hand, by direct substitution in the new formula, then the generated LTL formula is $\neg((\neg w)U(w \wedge \Box\neg(au \wedge ((\neg md)U(md\ \wedge\ ((\neg rp)\ U\ (rp\ \wedge\ ((\neg cr)Ucr))))))))$. The

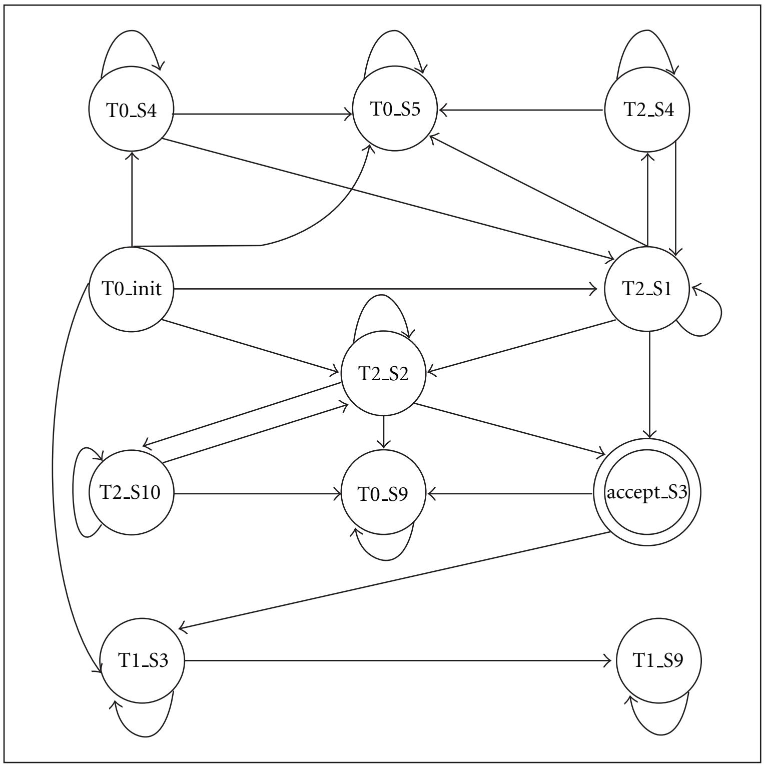


Figure 7: LTL2BA generated BA for $(([]((l \wedge \neg r) \rightarrow (([]p) \vee (pUr))))) \wedge \neg([]((l \wedge \neg r) \rightarrow (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r)))))$.

negation of this formula produces a BA with

(i) 6 states using SPIN,

(ii) 5 states using LTL2NBA and TMP,

(iii) 9 states using LTL2BA.

The previous example shows that although the difference in the number of states generated by the original LTL formulas in Table 3 and those in Table 4 might seem negligible, this difference becomes more significant when using the notion of CP classes to specify properties.

To further prove this point, Table 7 provides a sample comparison of the original and new formulas when some propositions are replaced by certain CP classes. The table shows the number of states generated by SPIN, LTL2NBA, TMP, and LTL2BA for formulas of patterns and scopes using CP. Note that these values were generated running the tools on the negated formulas for the specified pattern and scope. Old indicates Prospec's original formula, and New indicates the new formula. In Table 7 only one of the propositions was replaced by a CP class at a time. In generating this sample, we only used three of the CP classes: Parallel$_E$, Eventual$_C$, and Strict Eventual$_C$. We used these CP classes in replacing one of the propositions in four of the pattern/scope combinations; Absence of P After L, Existence of P After L, universality of P After L, and Q Responds to P After L. The choice of the CP classes and the pattern/scope combinations was based on the fact that direct substitution of the CP classes into the formulas for these pattern/scope formulas was possible. The CP class replacing a proposition is indicated by the subscript attached to the proposition. The number in the subscript refers to the number of the CP class in Table 6. Each CP class was comprised of two or three propositions. The goal was to include three propositions in each CP class; however in some cases some translation tools (mostly TMP and SPIN) could not generate BAs for the old Prospec formulas with CP classes containing three propositions. Those cases where CP classes were made of only two propositions are indicated with the symbol(*).

6. Discussion

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. These tools must generate specifications that correspond to the intent of the user. Prospec was demonstrated to provide such support [19]. It is also important that these tools 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 and a sample comparison given in the previous section show the effect on the number of states generated when using less efficient formulas as the base formulas when incorporating CP classes. The sample comparison also shows that in some cases, some translation tools could not generate BAs for the old Prospec formulas in which one of the propositions was replaced with CP classes containing three propositions.

The new formulas provided by this work generate BAs with fewer states in almost all pattern/scope combinations regardless of LTL to BA translator used. In addition, the BAs of the new formulas generated by SPIN seem to always be comparable to those generated by the other more efficient translators. This is significant to SPIN users, since those users do not need to use different LTL to BA translators and manually insert the resulting never-claim into the Promela code.

Another result of this work is that we are able to provide the user with more than one LTL mapping to the same pattern-scope combination. In some cases we can provide the user with three LTL formulas (considering the formulas provided by SPS) for a specific pattern and scope (*Absence of P Before R, e.g.*). This, along with the detailed descriptions of patterns and scopes (such as timelines) provided by the Prospec tool, enhances a user's understanding of LTL and can be used as an educational tool. Table 8 shows the formulas generated using the original Prospec formulas while Table 9 provides those generated using the new formulas.

Acknowledgments

The authors would like to thank Gerard Holzmann for his patience and help in answering many of our questions. They also would like to thank Carsten Fritz and Bjørn Teegen for making it possible to easily visualize and compare the different LTL to BA translations through the development of the website mentioned earlier in the paper. This work was partially supported by National Science Foundation grants HRD 0734825 and CNS 1042341.

References

[1] Z. Manna and A. Pnueli, "An anchored version of the temporal framework," in *Proceedings of the REX Workshop*, vol. 354 of *LNCS*, Springer, Mook, The Netherlands, May 1989.

[2] F. Laroussinie and PH. Schnoebelen, "Specification in CTL + past for verification in CTL," *Information and Computation*, vol. 156, no. 1-2, pp. 236–263, 2000.

[3] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, MIT Publishers, Cambridge, Mass, USA, 1999.

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

[5] O. Mondragon, A. Q. Gates, and S. Roach, "Prospec: support for elicitation and formal specification of software properties," in *Proceedings of the Runtime Verification Workshop*, O. Sokolsky and M. Viswanathan, Eds., vol. 89, ENTCS, Boulder, Colo, USA, July 2003.

[6] O. A. Mondragon and A. Q. Gates, "Supporting elicitation and specification of software properties through patterns and composite propositions," *International Journal of Software Engineering and Knowledge Engineering*, vol. 14, no. 1, pp. 21–41, 2004.

[7] O. Mondragon, *Elucidation and specification of software properties through patterns and composite propositions to support formal verification techniques*, Ph.D. thesis, The University of Texas, El Paso, Tex, USA, 2004.

[8] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Property specification patterns for finite-state verification," in *Proceedings of the 2nd Workshop on Formal Methods in Software Practice*, pp. 7–15, Clearwater Beach, Fla, USA, March 1998.

[9] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite state verification," in *Proceedings of the 21st International Conference on Software Engineering*, pp. 411–420, Los Angeles, Calif, USA, May 1999.

[10] D. Oddoux and P. Gastin, "Fast LTL to Büchi automata translation," in *Proceedings of the 13th International Conference on Computer Aided Verification (CAV '01)*, Paris, France, July 2001.

[11] K. Etessami and G. Holzmann, "Optimizing büchi automata," in *Proceedings of the 11th International Conference on Concurrency Theory*, August 2000.

[12] C. Fritz, "Constructing büchi automata from linear temporal logicusing simulation relations for alternating büchi automata," in *Proceedings of the Eighth Conference on Implementation and Application of Automata*, Santa Barbara, Calif, USA, July 2003.

[13] Z. Manna and A. Pnueli, "Completing the temporal picture," *Theoretical Computer Science*, vol. 83, no. 1, pp. 97–130, 1991.

[14] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri, "NUSMV: a new symbolic model verifier," in *Proceedings of the International Conference on Computer Aided Verification (CAV '99)*, Trento, Italy, July 1999.

[15] K. Havelund and T. Pressburger, "Model checking Java programs using Java PathFinder," *In- ternational Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 366–381, 2000.

[16] V. Stolz and E. Bodden, "Temporal assertions using aspectJ," in *Proceedings of the Fifth Workshop on Runtime Verification*, The University of Edinburgh, Scotland, UK, July 2005.

[17] E. Gamma and R. Helm, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, Boston, Mass, USA, 1995.

[18] "Spec patterns," December 2010, http://patterns.projects.cis.ksu.edu/.

[19] S. Salamah, A. Gates, S. Roach, and O. Mondragon, "Verifying pattern-generated LTL formulas: a case study," in *Proceedings of the 12th International SPIN Workshop*, pp. 200–220, San Francisco, Calif, USA, August 2005.

[20] "LTL2NBA," March 2007, http://www.ti.informatik.uni-kiel.de/ABA-Simulation/ltl.cgi.

[21] O. Mondragon, A. Gates, and S. Roach, "Composite propositions:toward support for formal specification of system properties," in *Proceedings of the 27th Annual IEEE/NASA Goddard Software Engineering Workshop*, Greenbelt, Md, USA, December 2002.