Clausal Temporal Resolution

MICHAEL FISHER and CLARE DIXON
Manchester Metropolitan University
and
MARTIN PEIM
Victoria University of Manchester

In this article, we examine how clausal resolution can be applied to a specific, but widely used, nonclassical logic, namely discrete linear temporal logic. Thus, we first define a normal form for temporal formulae and show how arbitrary temporal formulae can be translated into the normal form, while preserving satisfiability. We then introduce novel resolution rules that can be applied to formulae in this normal form, provide a range of examples, and examine the correctness and complexity of this approach. Finally, we describe related work and future developments concerning this work.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Temporal Logic*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Resolution*

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Theorem proving, resolution, temporal logic

1. INTRODUCTION

Temporal logic is a nonclassical logic that was originally developed in order to represent tense in natural language [Prior 1967]. More recently, it has achieved a significant role in the formal specification and verification of concurrent and distributed systems [Pnueli 1977]. It is commonly recognized that such *reactive systems* [Harel and Pnueli 1985] represent one of the most important classes of systems in computer science, and, although analysis of these systems is difficult, it has been successfully tackled using modal and temporal logics [Pnueli 1977; Emerson 1990; Stirling 1992]. In particular, a number of useful concepts, such as safety, liveness, and fairness can be formally, and concisely, specified using temporal logics [Manna and Pnueli 1992; Emerson 1990].

This work was partially supported by a SERC research grant GR/H/44646, an EPSRC PhD Studentship and EPSRC Research Grants GR/K57282 and GR/L87491.
Authors' addresses: M. Fisher and C. Dixon, Manchester Metropolitan University, Department of Computing and Mathematics, Manchester M1 5GD, U.K.; M. Peim, Victoria University of Manchester, Department of Computer Science, Manchester M13 9PL, U.K.

There are now a wide variety of temporal logics, differing in both their underlying model of time (for example, branching [Emerson and Srinivasan 1988] versus linear [Pnueli 1977; Manna and Pnueli 1992], and dense [Burgess and Gurevich 1985] versus discrete) and their intended area of application (for example, program specification [Manna and Pnueli 1992], temporal databases [Tansel 1993], knowledge representation [Artale and Franconi 1999], executable temporal logics [Barringer et al. 1996], natural language [Steedman 1997]). In this article we concentrate on a specific but widely used temporal logic, Propositional Linear Temporal Logic (PLTL), a discrete, linear temporal logic with finite past and infinite future; see for example Gabbay et al. [1980], Manna and Pnueli [1992], and Manna and Pnueli [1995].

Given a specification of some computational system in PLTL, we may want to establish that particular properties of the specification hold. Thus, for concurrent systems, we must often show the absence of deadlock, preservation of mutual exclusion, etc. (see for example Lamport [1983]). There are two main approaches to temporal verification that could be used here. If we can generate a finite-state structure representing *all* models of the system, then *model checking* techniques can be applied [Holzmann 1997]. Model checking involves establishing that a specific temporal formula is satisfied in the set of models representing the system. An alternative approach involves direct proof in PLTL. We consider this second approach, since not only may it be the case that models are not readily available, but even if they are, many systems we are interested in have very large, sometimes infinite, state spaces. Importantly, the use of direct proof methods may obviate the need to traverse all of a possible model structure.

The development of proof methods for temporal logic have followed three main approaches: tableaux, automata, and resolution. To show a formula φ valid, each of these methods is applied to the negation of φ, i.e., $\neg\varphi$. Tableaux-based approaches, for example Wolper [1983] and Gough [1984], attempt to systematically construct a structure from which a model can be extracted for $\neg\varphi$. The inability to construct such a model means that $\neg\varphi$ is unsatisfiable, and therefore φ is valid. The use of automata-based approaches depends on the fact that models for PLTL are simply infinite sequences of choices for truth values of proposition symbols. That is, an interpretation of a PLTL formula can be viewed as an infinite word over the alphabet that is the powerset of proposition symbols. Translations from PLTL into Büchi Automata are given in Sistla et al. [1987]. If the automaton for $\neg\varphi$ is empty then it accepts no infinite words; hence $\neg\varphi$ is unsatisfiable, and φ is valid.

Resolution-based approaches to proof in PLTL fall into two main classes: nonclausal and clausal. The nonclausal method described in Abadi and Manna [1985], and extended to first-order temporal logic in Abadi and Manna [1990], requires a large number of resolution rules, making implementation of this method difficult. Clausal resolution was suggested as a proof method for classical logic by Robinson [1965] and was claimed to be *machine oriented*, i.e., suitable to be performed by computer, as it has one rule of inference that may be applied many times. Again, to show a formula φ is valid it is negated, and $\neg\varphi$ is translated into a normal form. The resolution inference rule is applied until either no new inferences can be made or a contradiction is obtained. The generation of a contradiction means that $\neg\varphi$ is unsatisfiable and therefore that φ is valid.

Since clausal resolution is a simple and adaptable proof method for classical logics with a bank of research into heuristics and strategies, it is perhaps surprising that few attempts have been made to extend this to temporal logics. However, discrete temporal logics, such as PLTL, are difficult to reason about as the interaction between the $\square$-operator (meaning *always in the future*) and the $\bigcirc$-operator (meaning *in the next moment in time*) encodes a form of induction. Thus, a special temporal resolution rule is needed to handle this. There have been two previous attempts (known to the authors) at developing clausal resolution for temporal logics. The method described in Cavalli and Fariñas del Cerro [1984] is only applicable to a subset of the operators allowed in this article, i.e., for a less expressive language, and contains a more complex normal form. The method described in Venkatesh [1986] is the closest to that described in this article, the main difference being that the reasoning is carried out forward into the future while our approach involves reasoning backward until a contradiction is generated in the initial state. Both of these are discussed further in Section 8.

The development of the new resolution method described in this article is motivated not only by our wish to show that such a resolution system can be both simple and elegant, but also by our view that clausal resolution techniques will, in the future, provide the basis for the most efficient temporal theorem provers. While, in previous years, the most successful theorem provers for modal and temporal logics have been tableau-based (e.g., Horrocks [1998]), the use of resolution has now been shown to be at least competitive [Hustadt and Schmidt 1999]. In the classical framework, clausal resolution has led to many refinements aimed at guiding the search for a refutation, e.g., Chang and Lee [1973] and Wos et al. [1984]. In addition, several efficient, fast, and widely used resolution-based theorem provers have been developed, e.g., OTTER [McCune 1994] and SPASS [Weidenbach 1997]. It is our view that a clausal temporal resolution system has the potential to utilize a range of such efficient improvements developed for both classical and modal resolution.

Thus, our approach is clausal. In particular, we define a very simple (and flexible) normal form, called Separated Normal Form (SNF), that removes all but a core set of temporal operators. Two types of resolution rule are then defined, one analogous to the classical resolution rule and the other a new *temporal resolution* rule. However, due to the interaction between the $\square$ and $\bigcirc$ operators mentioned previously, the application of the temporal resolution rule is nontrivial, requiring specialized algorithms [Dixon 1996]. It is not our intention here to analyze experimental results concerning use of the resolution method (which still remain part of our future work), but simply to provide a logically complete basis for clausal temporal resolution. While short reports on this work have appeared previously, notably in Fisher [1991], this article provides the first exposition of the full completeness result for this temporal resolution method. In addition, it provides important properties of the translation into the normal form, and presents a simpler future-time formulation of the method.

The structure of the article is as follows. In Section 2 we give the syntax and semantics of PLTL. In Section 3, we define the normal form (SNF), show how any PLTL formula may be translated into SNF, and consider the properties of this translation. The resolution rules for formulae in SNF are given in Section 4 while example refutations are provided in Section 5. Issues of correctness and complexity are considered in Section 6 and Section 7, respectively. Related work is examined in Section 8, and conclusions and future work are provided in Section 9.

2. PROPOSITIONAL TEMPORAL LOGIC

Propositional Temporal Logic (PLTL) was originally developed from work on tense logics [Prior 1967], but has come to prominence through its application in the specification and verification of both software and hardware [Pnueli 1977]. The particular variety of temporal logic we consider is based on a linear, discrete model of time with finite past and infinite future [Gabbay et al. 1980; Lichtenstein et al. 1985]. Thus, the temporal operators supplied operate over a sequence of distinct "moments" in time.

There are several ways to view this logic. One is as a classical propositional logic augmented with temporal connectives (or operators). An alternative characterization can be given in terms of a multimodal language with two different modalities, one representing the "next" moment in time, the other representing all future moments in time ($\bigcirc$ and $\square$ below, respectively).

While it is possible to include past-time operators in the definition of the logic, we choose not to do so in this exposition, since, as models have a finite past, such operators add no extra expressive power [Gabbay et al. 1980; Lichtenstein et al. 1985]. However, if the addition of past-time operators makes the expression of certain properties easier (see, for example Lichtenstein et al. [1985]) they can be easily incorporated (see Section 3 for more details).

The future-time connectives that we use include $\diamondsuit$ (*sometime in the future*), $\square$ (*always in the future*), $\bigcirc$ (*in the next moment in time*), $\mathcal{U}$ (*until*), and $\mathcal{W}$ (*unless*, or *weak until*). To assist readers who may be unfamiliar with the semantics of the temporal operators we introduce, in the next section, all operators as basic. Alternatively we could have provided the syntax and semantics of just a subset of the operators and introduced the remainder as abbreviations.

2.1 Syntax

PLTL formulae are constructed from the following elements:

—A set, $\mathcal{P}$, of propositional symbols.

—Propositional connectives, **true**, **false**, $\neg$, $\vee$, $\wedge$, and $\Rightarrow$.

—Temporal connectives, $\bigcirc$, $\diamondsuit$, $\square$, $\mathcal{U}$, and $\mathcal{W}$.

The set of well-formed formulae of PLTL, denoted by WFF, is inductively defined as the smallest set satisfying the following:

—Any element of $\mathcal{P}$ is in WFF.

—**true** and **false** are in WFF.

—If A and B are in WFF then so are

$$\neg A \quad A \vee B \quad A \wedge B \quad A \Rightarrow B \quad \Diamond A \quad \Box A \quad A\,\mathcal{U}\,B \quad A\,\mathcal{W}\,B \quad \bigcirc A.$$

A *literal* is defined as either a proposition symbol or the negation of a proposition symbol. An *eventuality* is defined as a formula of the form $\Diamond A$.

2.2 Semantics

PLTL is interpreted over discrete, linear structures, for example the natural numbers, $\mathbb{N}$. A model of PLTL, σ, can be characterized as a sequence of *states*

$$\sigma = s_0, s_1, s_2, s_3, \ldots$$

where each state, s_i, is a set of proposition symbols, representing those proposition symbols which are satisfied in the ith moment in time. As formulae in PLTL are interpreted at a particular state in the sequence (i.e., at a particular moment in time), the notation

$$(\sigma, i) \models A$$

denotes the truth (or otherwise) of formula A in the model σ at state index $i \in \mathbb{N}$. For any formula A, model σ, and state index $i \in \mathbb{N}$, then either $(\sigma, i) \models A$ holds or $(\sigma, i) \models A$ does not hold, denoted by $(\sigma, i) \not\models A$. If there is some σ such that $(\sigma, 0) \models A$, then A is said to be *satisfiable*. If $(\sigma, 0) \models A$ for all models, σ, then A is said to be *valid* and is written $\models A$. Note that formulae here are interpreted at s_0; this is an alternative, but equivalent, definition to the one commonly used [Emerson 1990].

The semantics of WFF can now be given, as follows.

$(\sigma, i) \models p$	iff	$p \in s_i$	[where $p \in \mathcal{P}$]
$(\sigma, i) \models \mathbf{true}$			
$(\sigma, i) \not\models \mathbf{false}$			
$(\sigma, i) \models A \wedge B$	iff	$(\sigma, i) \models A$ and $(\sigma, i) \models B$	
$(\sigma, i) \models A \vee B$	iff	$(\sigma, i) \models A$ or $(\sigma, i) \models B$	
$(\sigma, i) \models A \Rightarrow B$	iff	$(\sigma, i) \models \neg A$ or $(\sigma, i) \models B$	
$(\sigma, i) \models \neg A$	iff	$(\sigma, i) \not\models A$	
$(\sigma, i) \models \bigcirc A$	iff	$(\sigma, i+1) \models A$	
$(\sigma, i) \models \Diamond A$	iff	there exists a $k \in \mathbb{N}$ such that $k \geqslant i$ and $(\sigma, k) \models A$	
$(\sigma, i) \models \Box A$	iff	for all $j \in \mathbb{N}$, if $j \geqslant i$ then $(\sigma, j) \models A$	
$(\sigma, i) \models A\,\mathcal{U}\,B$	iff	there exists a $k \in \mathbb{N}$, such that $k \geqslant i$ and $(\sigma, k) \models B$ and for all $j \in \mathbb{N}$, if $i \leqslant j < k$ then $(\sigma, j) \models A$	
$(\sigma, i) \models A\,\mathcal{W}\,B$	iff	$(\sigma, i) \models A\,\mathcal{U}\,B$ or $(\sigma, i) \models \Box A$	

2.3 Proof Theory

The standard axioms and inference rules for PLTL are as follows (taking the temporal operators $\bigcirc$, $\Box$, and $\mathcal{U}$ as primitive and the remaining as abbreviations—see Section 2.3.1). The axioms are all substitution instances of the following:

(1) all classical tautologies,

(2) $\vdash \ \square(A \Rightarrow B) \Rightarrow (\square A \Rightarrow \ \square B)$

(3) $\vdash \ \bigcirc\neg A \Rightarrow \neg\bigcirc A$

(4) $\vdash \ \neg\bigcirc A \Rightarrow \bigcirc\neg A$

(5) $\vdash \ \bigcirc(A \Rightarrow B) \Rightarrow (\bigcirc A \Rightarrow \bigcirc B)$

(6) $\vdash \ \square A \Rightarrow A \wedge \bigcirc \square A$

(7) $\vdash \ \square(A \Rightarrow \bigcirc A) \Rightarrow (A \Rightarrow \ \square A)$

(8) $\vdash \ (A\,\mathcal{U}\,B) \Rightarrow \lozenge B$

(9) $\vdash \ (A\,\mathcal{U}\,B) \Rightarrow (B \vee (A \wedge \bigcirc(A\,\mathcal{U}\,B)))$

(10) $\vdash \ (B \vee (A \wedge \bigcirc(A\,\mathcal{U}\,B))) \Rightarrow (A\,\mathcal{U}\,B)$

The inference rules are modus ponens

$$\frac{\vdash A \quad \vdash A \Rightarrow B}{\vdash B}$$

and generalization

$$\frac{\vdash A}{\vdash \ \square A}\ .$$

THEOREM 2.3.1 (SOUNDNESS)[GABBAY ET AL. 1980]. *If* $\vdash A$ *then* A *is valid in PLTL.*

THEOREM 2.3.2 (COMPLETENESS)[GABBAY ET AL. 1980]. *If* A *is valid in PLTL then* $\vdash A$.

A complete axiom system for PLTL with future-time temporal operators is given in Gabbay et al. [1980]. The axiom system presented here is slightly different from the original due to slight differences in the semantics of the connectives used. We note that it is difficult to use such an axiom system for automated theorem proving, as it is not always clear which step should be taken next to move toward a proof.

2.3.1 *Some Equivalences.* To assist the understanding of the translation to the normal form given in Section 3 we list some equivalent PLTL formulae.

$$\begin{aligned}
\bigcirc(A \wedge B) &\equiv \bigcirc A \wedge \bigcirc B \\
\neg\bigcirc A &\equiv \bigcirc\neg A \\
\square A &\equiv A \wedge \bigcirc \square A \\
\lozenge A &\equiv A \vee \bigcirc\lozenge A \\
\neg\square A &\equiv \lozenge\neg A \\
(A\,\mathcal{U}\,B) &\equiv B \vee (A \wedge \bigcirc(A\,\mathcal{U}\,B)) \\
(A\,\mathcal{U}\,B) &\equiv (A\,\mathcal{W}\,B) \wedge \lozenge B \\
\neg(A\,\mathcal{U}\,B) &\equiv \neg B\,\mathcal{W}(\neg A \wedge \neg B) \\
(A\,\mathcal{W}\,B) &\equiv B \vee (A \wedge \bigcirc(A\,\mathcal{W}\,B)) \\
\neg(A\,\mathcal{W}\,B) &\equiv \neg B\,\mathcal{U}(\neg A \wedge \neg B)
\end{aligned}$$

These are standard and are given in Gough [1984] for example.

3. A NORMAL FORM FOR PROPOSITIONAL TEMPORAL LOGIC

3.1 Separated Normal Form

The resolution method is clausal, and so works on formulae transformed into a normal form. The normal form, called Separated Normal Form (SNF), was inspired by (but does not require) Gabbay's separation result [Gabbay 1987], which states that temporal formulae can be transformed into their past, present, and future-time components. The normal form we present comprises formulae that are implications with present-time formulae on the left-hand side and (present or) future-time formulae on the right-hand side. The transformation into the normal form reduces most of the temporal operators to a core set and rewrites formulae to be in a particular form. The transformation into SNF depends on three main operations: the renaming of complex subformulae; the removal of temporal operators; and classical style rewrite operations.

Renaming, as suggested in Plaisted and Greenbaum [1986], is a way of preserving the structure of a formula when translating into a normal form in classical logic. Here, complex subformulae can be replaced by a new proposition symbol, and the truth value of the new proposition symbol is linked to the subformula it represents at all points in time. The removal of temporal operators is carried out by using (fixed point) equivalences, for example

$$\square p \equiv (p \wedge \bigcirc \square p)$$

that "unwind" the temporal operators to give formulae that need to hold both now and in the future. Classical rewrite operations allow us to manipulate formulae into the required form.

To assist in the definition of the normal form we introduce a further (nullary) connective **start** that holds only at the beginning of time, i.e.,

$$(\sigma, i) \models \mathbf{start} \quad \text{iff} \quad i = 0.$$

This allows the general form of the (PLTL-clauses of the) normal form to be implications. An alternative would be to allow disjunctions of literals as part of the normal form representing the clauses holding at the beginning of time.

Formulae in SNF are of the general form

$$\square \bigwedge_i A_i$$

where each A_i is known as a *PLTL-clause* (analogous to a "clause" in classical logic) and must be one of the following forms with each particular k_a, k_b, l_c, l_d, and l representing a literal.

$$\mathbf{start} \Rightarrow \bigvee_c l_c \qquad \text{(an } \textit{initial} \text{ PLTL-clause)}$$

$$\bigwedge_a k_a \Rightarrow \bigcirc \bigvee_d l_d \qquad \text{(a } \textit{step} \text{ PLTL-clause)}$$

$$\bigwedge_b k_b \Rightarrow \Diamond l \qquad \text{(a } \textit{sometime} \text{ PLTL-clause)}$$

For convenience, the outer $\square$ and $\wedge$ connectives are usually omitted, and the set of PLTL-clauses $\{A_i\}$ is considered. Different variants of the normal form have been suggested [Fisher 1992; Fisher and Noël 1992; Fisher 1997]. For example, where PLTL is extended to allow past-time operators the normal form has **start** or $\bullet A$ (where $\bullet$ means in the *previous moment* in time and A is a conjunction of literals) on the left-hand side of the PLTL-clauses and a present-time formula or eventuality (i.e., $\Diamond l$) on the right-hand side. Other versions allow PLTL-clauses of the form **start** $\Rightarrow \Diamond l$. These are all expressively equivalent when models with finite past are considered.

To apply the temporal resolution rule (see Section 4.2), one or more step PLTL-clauses may need to be combined. Consequently, a variant on SNF called *merged-SNF (SNF$_m$)* [Fisher 1991] is also defined. Given a set of PLTL-clauses in SNF, any PLTL-clause in SNF is also a PLTL-clause in SNF$_m$. Any two PLTL-clauses in SNF$_m$ may be combined to produce a PLTL-clause in SNF$_m$ as follows.

$$\frac{\begin{array}{rcl} A & \Rightarrow & \bigcirc C \\ B & \Rightarrow & \bigcirc D \end{array}}{(A \wedge B) \;\Rightarrow\; \bigcirc(C \wedge D)}$$

Thus, any possible conjunctive combination of SNF PLTL-clauses can be represented in SNF$_m$.

3.2 Translation into SNF

In this section, we review the translation of an arbitrary PLTL formula into the normal form (this extends the exposition provided in Fisher [1997]). The procedure uses the technique of renaming complex subformulae by a new proposition symbol, and the truth value of the new proposition symbol is linked to that of the renamed formula at all moments in time. Thus, in the exposition below the new proposition symbols introduced, namely those indicated by **v**, **y** and **z**, must be new at each iteration of the procedure. In the remainder of Section 3 and in the Appendix, where the proofs for this section are located, we show such new proposition symbols in bold face type.

Take any formula A of PLTL and translate into SNF by applying the τ_0 and τ_1 transformations described below (where **y** is a new proposition symbol).

$$\tau_0[A] \longrightarrow \square(\textbf{start} \Rightarrow \mathbf{y}) \wedge \tau_1[\square(\mathbf{y} \Rightarrow A)]$$

Next, we give the τ_1 transformation where x is a proposition symbol. If the main operator on the right of the implication is a classical operator (other than nonnegated disjunction) remove it as follows.

$$\begin{array}{rcl}
\tau_1[\square(x \Rightarrow (A \wedge B))] & \longrightarrow & \tau_1[\square(x \Rightarrow A)] \wedge \tau_1[\square(x \Rightarrow B)] \\
\tau_1[\square(x \Rightarrow (A \Rightarrow B))] & \longrightarrow & \tau_1[\square(x \Rightarrow (\neg A \vee B))] \\
\tau_1[\square(x \Rightarrow \neg(A \wedge B))] & \longrightarrow & \tau_1[\square(x \Rightarrow (\neg A \vee \neg B))] \\
\tau_1[\square(x \Rightarrow \neg(A \Rightarrow B))] & \longrightarrow & \tau_1[\square(x \Rightarrow A)] \wedge \tau_1[\square(x \Rightarrow \neg B)] \\
\tau_1[\square(x \Rightarrow \neg(A \vee B))] & \longrightarrow & \tau_1[\square(x \Rightarrow \neg A)] \wedge \tau_1[\square(x \Rightarrow \neg B)]
\end{array}$$

Complex subformulae enclosed in any temporal operators are renamed as follows (where $\mathbf{v}$, $\mathbf{y}$, and $\mathbf{z}$ are new proposition symbols).

$$
\begin{array}{rcll}
\tau_1[\Box(x \Rightarrow \bigcirc A)] & \longrightarrow & \Box(x \Rightarrow \bigcirc \mathbf{y}) \wedge \tau_1[\Box(\mathbf{y} \Rightarrow A)] & A \text{ neither literal nor disjunction of literals.} \\
\tau_1[\Box(x \Rightarrow \neg\bigcirc A)] & \longrightarrow & \Box(x \Rightarrow \bigcirc \mathbf{y}) \wedge \tau_1[\Box(\mathbf{y} \Rightarrow \neg A)] & \\
\tau_1[\Box(x \Rightarrow \Box A)] & \longrightarrow & \tau_1[\Box(x \Rightarrow \Box \mathbf{y})] \wedge \tau_1[\Box(\mathbf{y} \Rightarrow A)] & A \text{ not a literal.} \\
\tau_1[\Box(x \Rightarrow \neg\Box A)] & \longrightarrow & \Box(x \Rightarrow \Diamond \mathbf{y}) \wedge \tau_1[\Box(\mathbf{y} \Rightarrow \neg A)] & \\
\tau_1[\Box(x \Rightarrow \Diamond A)] & \longrightarrow & \Box(x \Rightarrow \Diamond \mathbf{y}) \wedge \tau_1[\Box(\mathbf{y} \Rightarrow A)] & A \text{ not a literal.} \\
\tau_1[\Box(x \Rightarrow \neg\Diamond A)] & \longrightarrow & \tau_1[\Box(x \Rightarrow \Box \mathbf{y})] \wedge \tau_1[\Box(\mathbf{y} \Rightarrow \neg A)] & \\
\tau_1[\Box(x \Rightarrow A\,\mathcal{U}\,B)] & \longrightarrow & \tau_1[\Box(x \Rightarrow \mathbf{y}\,\mathcal{U}\,B)] \wedge \tau_1[\Box(\mathbf{y} \Rightarrow A)] & A \text{ not a literal.} \\
\tau_1[\Box(x \Rightarrow A\,\mathcal{U}\,B)] & \longrightarrow & \tau_1[\Box(x \Rightarrow A\,\mathcal{U}\,\mathbf{y})] \wedge \tau_1[\Box(\mathbf{y} \Rightarrow B)] & B \text{ not a literal.} \\
\tau_1[\Box(x \Rightarrow \neg(A\,\mathcal{U}\,B))] & \longrightarrow & \tau_1[\Box(x \Rightarrow (\mathbf{y}\,\mathcal{W}\,\mathbf{v}))] \wedge & \\
 & & \tau_1[\Box(\mathbf{v} \Rightarrow (\mathbf{y} \wedge \mathbf{z}))] \wedge & \\
 & & \tau_1[\Box(\mathbf{y} \Rightarrow \neg B)] \wedge \tau_1[\Box(\mathbf{z} \Rightarrow \neg A)] & \\
\tau_1[\Box(x \Rightarrow A\,\mathcal{W}\,B)] & \longrightarrow & \tau_1[\Box(x \Rightarrow \mathbf{y}\,\mathcal{W}\,B)] \wedge \tau_1[\Box(\mathbf{y} \Rightarrow A)] & A \text{ not a literal.} \\
\tau_1[\Box(x \Rightarrow A\,\mathcal{W}\,B)] & \longrightarrow & \tau_1[\Box(x \Rightarrow A\,\mathcal{W}\,\mathbf{y})] \wedge \tau_1[\Box(\mathbf{y} \Rightarrow B)] & B \text{ not a literal.} \\
\tau_1[\Box(x \Rightarrow \neg(A\,\mathcal{W}\,B))] & \longrightarrow & \tau_1[\Box(x \Rightarrow (\mathbf{y}\,\mathcal{U}\,\mathbf{v}))] \wedge & \\
 & & \tau_1[\Box(\mathbf{v} \Rightarrow (\mathbf{y} \wedge \mathbf{z}))] \wedge & \\
 & & \tau_1[\Box(\mathbf{y} \Rightarrow \neg B)] \wedge \tau_1[\Box(\mathbf{z} \Rightarrow \neg A)] &
\end{array}
$$

The negated $\mathcal{W}$ and $\mathcal{U}$ operators involve the introduction of three new proposition symbols. Consider the transformation applied to $x \Rightarrow \neg(A\,\mathcal{U}\,B)$. Applying the equivalence provided in Section 2.3.1 we have $x \Rightarrow (\neg B\,\mathcal{W}(\neg A \wedge \neg B))$. To avoid repeating the subformula $\neg B$ in the translation, and so that the resultant *unless* operator is applied to proposition symbols, we introduce three new variables: $\mathbf{y}$ replaces $\neg B$, $\mathbf{z}$ replaces $\neg A$, $\mathbf{v}$ replaces $\mathbf{y} \wedge \mathbf{z}$.

Then, any temporal operators, applied to literals, that are not allowed in the normal form are removed as follows (where, again, $\mathbf{y}$ is a new proposition symbol and l and m are literals).

$$
\tau_1[\Box(x \Rightarrow \Box l)] \longrightarrow
\begin{array}{l}
\tau_1[\Box(x \Rightarrow l)] \wedge \\
\tau_1[\Box(x \Rightarrow \mathbf{y})] \wedge \\
\quad \Box(\mathbf{y} \Rightarrow \bigcirc l) \wedge \\
\quad \Box(\mathbf{y} \Rightarrow \bigcirc \mathbf{y})
\end{array}
$$

$$
\tau_1[\Box(x \Rightarrow l\,\mathcal{U}\,m)] \longrightarrow
\begin{array}{l}
\quad \Box(x \Rightarrow \Diamond m) \wedge \\
\tau_1[\Box(x \Rightarrow (l \vee m))] \wedge \\
\tau_1[\Box(x \Rightarrow (\mathbf{y} \vee m))] \wedge \\
\quad \Box(\mathbf{y} \Rightarrow \bigcirc(l \vee m)) \wedge \\
\quad \Box(\mathbf{y} \Rightarrow \bigcirc(\mathbf{y} \vee m))
\end{array}
$$

$$\tau_1[\,\Box(x \Rightarrow l\,\mathcal{W}\,m)] \;\longrightarrow\; \begin{array}{l} \tau_1[\,\Box(x \;\Rightarrow\; (l \vee m))] \,\wedge \\ \tau_1[\,\Box(x \;\Rightarrow\; (\mathbf{y} \vee m))] \,\wedge \\ \quad \Box(\mathbf{y} \;\Rightarrow\; \bigcirc(l \vee m)) \,\wedge \\ \quad \Box(\mathbf{y} \;\Rightarrow\; \bigcirc(\mathbf{y} \vee m)) \end{array}$$

Next, we use renaming on formulae whose right-hand side has disjunction as its main operator but may not be in the correct form, where **y** is a new proposition symbol, D is a disjunction of formulae, and A is neither a literal nor a disjunction of literals.

$$\tau_1[\,\Box(x \Rightarrow D \vee A)] \;\longrightarrow\; \begin{array}{l} \tau_1[\,\Box(x \;\Rightarrow\; D \vee \mathbf{y})] \;\wedge \\ \tau_1[\,\Box(\mathbf{y} \;\Rightarrow\; A)] \end{array}$$

Finally, we rewrite formulae, containing no temporal operators, whose right-hand side is a disjunction of literals, **true** or **false** (note that ¬**true** and ¬**false** are rewritten to **false** and **true**, respectively) into PLTL-clause form and stop applying the transformation to PLTL-clauses already in the correct form (where D is a literal or disjunction of literals and l and each l_i are literals).

$$\tau_1[\,\Box(x \Rightarrow D)] \;\longrightarrow\; \begin{array}{l} \Box(\mathbf{start} \;\Rightarrow\; \neg x \vee D) \wedge \\ \Box(\mathbf{true} \;\Rightarrow\; \bigcirc(\neg x \vee D)) \end{array}$$

$$\tau_1[\,\Box(x \Rightarrow \mathbf{true})] \;\longrightarrow\; \begin{array}{l} \Box(\mathbf{start} \;\Rightarrow\; \mathbf{true}) \wedge \\ \Box(\mathbf{true} \;\Rightarrow\; \bigcirc\mathbf{true}) \end{array}$$

$$\tau_1[\,\Box(x \Rightarrow \mathbf{false})] \;\longrightarrow\; \begin{array}{l} \Box(\mathbf{start} \;\Rightarrow\; \neg x) \wedge \\ \Box(\mathbf{true} \;\Rightarrow\; \bigcirc\neg x) \end{array}$$

$$\tau_1[\,\Box(x \Rightarrow \Diamond l)] \;\longrightarrow\; \Box(x \Rightarrow \Diamond l)$$

$$\tau_1[\,\Box(x \Rightarrow \bigcirc(l_1 \vee \ldots \vee l_n))] \;\longrightarrow\; \Box(x \Rightarrow \bigcirc(l_1 \vee \ldots \vee l_n))$$

Thus, the above transformations are applied until the formula is in the form

$$\bigwedge_i \Box A_i$$

where each A_i is one of the three required formats. This, in turn, is equivalent to

$$\Box \bigwedge_i A_i.$$

3.3 Properties of the Translation to SNF

Our aim is to show that the transformation is satisfiability preserving. This is shown in two parts. Firstly any model for a transformed formula is also a model for the original, and secondly given a model for a PLTL formula there is always a model for its transformation into the normal form. These proofs (Lemmas A.1–A.4) are given in the Appendix.

THEOREM 3.3.1. *A PLTL formula A is satisfiable if, and only if,* $\tau_0[A]$ *is satisfiable.*

PROOF. Lemmas A.1 and A.2 in the Appendix show that if $\tau_0[A]$ is satisfiable in a model, then A is satisfiable in the same model. Lemmas A.3 and A.4 in the Appendix show that, given a model for A, then we can construct a model for $\tau_0[A]$. □

3.4 Example

We illustrate the translation to the normal form by carrying out a simple example transformation. Assume we want to show

$$(\Diamond p \wedge \Box(p \Rightarrow \bigcirc p)) \Rightarrow \Diamond \Box p$$

is valid. We negate, obtaining

$$(\Diamond p \wedge \Box(p \Rightarrow \bigcirc p)) \wedge \Box\Diamond\neg p,$$

and begin to translate this into SNF. First, we anchor to the beginning of time and split the conjuncts.

1. $\mathbf{start} \Rightarrow \mathbf{f}$
2. $\mathbf{f} \Rightarrow \Diamond p$
3. $\mathbf{f} \Rightarrow \Box(p \Rightarrow \bigcirc p)$
4. $\mathbf{f} \Rightarrow \Box\Diamond\neg p$

Formulae labeled 1 and 2 are now in normal form. We work on formula 3, renaming the subformula $p \Rightarrow \bigcirc p$.

5. $\mathbf{f} \Rightarrow \Box\mathbf{q}$
6. $\mathbf{q} \Rightarrow (p \Rightarrow \bigcirc p)$

Next, we apply the $\Box$ removal rules to formula 5 (to give 7, 8, 9, and 10) and rewrite formula 6 (to give 11).

7. $\mathbf{f} \Rightarrow \mathbf{q}$
8. $\mathbf{f} \Rightarrow \mathbf{r}$
9. $\mathbf{r} \Rightarrow \bigcirc\mathbf{q}$
10. $\mathbf{r} \Rightarrow \bigcirc\mathbf{r}$
11. $\mathbf{q} \Rightarrow (\neg p \vee \bigcirc p)$

Then, formulae 7 and 8 are rewritten into the normal form (giving 12–15) and the subformula $\bigcirc p$ in formula 11 is renamed.

12. $\mathbf{start} \Rightarrow \neg\mathbf{f} \vee \mathbf{q}$
13. $\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{f} \vee \mathbf{q})$
14. $\mathbf{start} \Rightarrow \neg\mathbf{f} \vee \mathbf{r}$
15. $\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{f} \vee \mathbf{r})$
16. $\mathbf{q} \Rightarrow (\neg p \vee \mathbf{s})$
17. $\mathbf{s} \Rightarrow \bigcirc p$

Formula 16 is then rewritten into the correct form.

18. $\mathbf{start} \Rightarrow (\neg\mathbf{q} \vee \neg p \vee \mathbf{s})$
19. $\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{q} \vee \neg p \vee \mathbf{s})$

Next, we work on formula 4 renaming $\Diamond\neg p$ with the new proposition symbol t.

20. $\mathbf{f} \Rightarrow \Box\mathbf{t}$
21. $\mathbf{t} \Rightarrow \Diamond\neg p$

Then, we remove the $\Box$ operator from formula 20 as previously

22. $\mathbf{f} \Rightarrow \mathbf{t}$
23. $\mathbf{f} \Rightarrow \mathbf{u}$
24. $\mathbf{u} \Rightarrow \bigcirc\mathbf{t}$
25. $\mathbf{u} \Rightarrow \bigcirc\mathbf{u}$

and finally write formulae 22 and 23 into the normal form.

26. $\mathbf{start} \Rightarrow \neg\mathbf{f} \vee \mathbf{t}$
27. $\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{f} \vee \mathbf{t})$
28. $\mathbf{start} \Rightarrow \neg\mathbf{f} \vee \mathbf{u}$
29. $\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{f} \vee \mathbf{u})$

The resulting normal form is as follows.

1.	$\mathbf{start} \Rightarrow \mathbf{f}$	18.	$\mathbf{start} \Rightarrow (\neg\mathbf{q} \vee \neg p \vee \mathbf{s})$
2.	$\mathbf{f} \Rightarrow \Diamond p$	19.	$\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{q} \vee \neg p \vee \mathbf{s})$
9.	$\mathbf{r} \Rightarrow \bigcirc\mathbf{q}$	21.	$\mathbf{t} \Rightarrow \Diamond\neg p$
10.	$\mathbf{r} \Rightarrow \bigcirc\mathbf{r}$	24.	$\mathbf{u} \Rightarrow \bigcirc\mathbf{t}$
12.	$\mathbf{start} \Rightarrow \neg\mathbf{f} \vee \mathbf{q}$	25.	$\mathbf{u} \Rightarrow \bigcirc\mathbf{u}$
13.	$\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{f} \vee \mathbf{q})$	26.	$\mathbf{start} \Rightarrow \neg\mathbf{f} \vee \mathbf{t}$
14.	$\mathbf{start} \Rightarrow \neg\mathbf{f} \vee \mathbf{r}$	27.	$\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{f} \vee \mathbf{t})$
15.	$\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{f} \vee \mathbf{r})$	28.	$\mathbf{start} \Rightarrow \neg\mathbf{f} \vee \mathbf{u}$
17.	$\mathbf{s} \Rightarrow \bigcirc p$	29.	$\mathbf{true} \Rightarrow \bigcirc(\neg\mathbf{f} \vee \mathbf{u})$

4. RESOLUTION RULES

Once a formula has been transformed into SNF, both step resolution and temporal resolution operations can be applied. Step resolution effectively consists of the application of the standard classical resolution rule to formulae representing constraints at a particular moment in time, together with simplification rules, subsumption rules, and rules for transferring contradictions within states to constraints on previous states. Temporal resolution resolves a sometime PLTL-clause whose right-hand side is, for example, $\Diamond l$ with a set of SNF_m PLTL-clauses that together imply that l is always false. We also describe *augmentation*, the addition of new variables required to translate the resolvent from temporal resolution into SNF at the start of the proof. This is useful in ensuring that no new proposition symbols need to be added during the proof.

4.1 Step Resolution

Pairs of initial or step PLTL-clauses may be resolved using the following (resolution) operations (where A and B are disjunctions of literals, C and D are conjunctions of literals, and p is a proposition).

$$\frac{\begin{array}{rcl}\textbf{start} & \Rightarrow & A \vee p\\ \textbf{start} & \Rightarrow & B \vee \neg p\end{array}}{\textbf{start} \Rightarrow A \vee B} \qquad \frac{\begin{array}{rcl}C & \Rightarrow & \bigcirc(A \vee p)\\ D & \Rightarrow & \bigcirc(B \vee \neg p)\end{array}}{(C \wedge D) \Rightarrow \bigcirc(A \vee B)}$$

The following is used for PLTL-clauses which imply **false** (where A is a conjunction of literals).

$$\{A \Rightarrow \bigcirc\textbf{false}\} \longrightarrow \left\{\begin{array}{rcl}\textbf{start} & \Rightarrow & \neg A\\ \textbf{true} & \Rightarrow & \bigcirc\neg A\end{array}\right\}$$

Thus, if, by satisfying A, a contradiction is produced in the next moment, then A must never be satisfied. The new constraints generated effectively represent $\Box\neg A$. This rewrite keeps formulae in the suggested normal form and may, in turn, allow further step resolution inferences to be carried out.

PLTL-clauses are kept in their simplest form by performing classical style simplification, for example performing the following contraction operations.

$$\begin{array}{rcllrcl}
(l \wedge A \wedge l) & \Rightarrow & \bigcirc B & \longrightarrow & (l \wedge A) & \Rightarrow & \bigcirc B\\
(l \wedge A \wedge \neg l) & \Rightarrow & \bigcirc B & \longrightarrow & \textbf{false} & \Rightarrow & \bigcirc B\\
(A \wedge \textbf{true}) & \Rightarrow & \bigcirc B & \longrightarrow & A & \Rightarrow & \bigcirc B\\
(A \wedge \textbf{false}) & \Rightarrow & \bigcirc B & \longrightarrow & \textbf{false} & \Rightarrow & \bigcirc B\\
A & \Rightarrow & \bigcirc(l \vee B \vee l) & \longrightarrow & A & \Rightarrow & \bigcirc(l \vee B)\\
A & \Rightarrow & \bigcirc(l \vee B \vee \neg l) & \longrightarrow & A & \Rightarrow & \bigcirc\textbf{true}\\
A & \Rightarrow & \bigcirc(B \vee \textbf{true}) & \longrightarrow & A & \Rightarrow & \bigcirc\textbf{true}\\
A & \Rightarrow & \bigcirc(B \vee \textbf{false}) & \longrightarrow & A & \Rightarrow & \bigcirc B
\end{array}$$

The following SNF PLTL-clauses can be removed during simplification, as they represent valid subformulae and therefore cannot contribute to the generation of a contradiction.

$$\begin{array}{rcl}\textbf{false} & \Rightarrow & \bigcirc A\\ A & \Rightarrow & \bigcirc\textbf{true}\end{array}$$

The first PLTL-clause is valid, as **false** can never be satisfied, and the second is valid, as $\bigcirc$**true** is always satisfied.

Subsumption also forms part of the step resolution process. Here, as in classical resolution, a PLTL-clause may be removed from the PLTL-clause-set if it is subsumed by another PLTL-clause already present. Subsumption may be expressed as the following operation.

$$\left\{\begin{array}{rcl}C & \Rightarrow & A\\ D & \Rightarrow & B\end{array}\right\} \xrightarrow{\vdash C \Rightarrow D \quad \vdash B \Rightarrow A} \{D \Rightarrow B\}$$

The side conditions $\vdash C \Rightarrow D$ and $\vdash B \Rightarrow A$ must hold before this subsumption step can be applied, and, in this case, the PLTL-clause $C \Rightarrow A$ can be deleted without losing information.

The step resolution process terminates when either no new resolvents can be generated or a contradiction is derived by generating the following unsatisfiable formula

$$\textbf{start} \Rightarrow \textbf{false}.$$

4.2 Temporal Resolution

The temporal resolution operation effectively resolves together formulae containing the $\Box$ and $\Diamond$ connectives. However, the inductive interaction between the $\bigcirc$ and $\Box$ connectives in PLTL ensures that the application of such an operation is nontrivial. Further, as the translation to SNF restricts the PLTL-clauses to be of a certain form, the application of such an operation will be between a sometime PLTL-clause and a *set* of step PLTL-clauses that together ensure a complementary literal will *always* hold. Intuitively, temporal resolution may be applied between an eventuality, i.e., a formula $\Diamond l$ from the right-hand side of a sometime PLTL-clause such as $C \Rightarrow \Diamond l$, and a formula which forces l always to be false. Once the left-hand side of the sometime PLTL-clause (i.e., C) is satisfied then, for the formula to be satisfiable, there must be no other PLTL-clauses forcing l to always be false. To resolve with $C \Rightarrow \Diamond l$ then, a set of SNF_m PLTL-clauses (see Section 3) must be identified such that they characterize $A \Rightarrow \bigcirc \Box \neg l$ (where A is in DNF).[1] So, the general temporal resolution operation, written as an inference rule, becomes

$$\begin{array}{rcl} A & \Rightarrow & \bigcirc \Box \neg l \\ C & \Rightarrow & \Diamond l \\ \hline C & \Rightarrow & (\neg A)\,\mathcal{W}\,l \end{array}$$

The intuition behind the resolvent is that once C has occurred then A must not be satisfied until l has occurred (i.e., the eventuality has been satisfied). (Note that the generation of $C \Rightarrow (\neg A)\,\mathcal{U}\,l$ as a resolvent would be sound. However as $(\neg A)\,\mathcal{U}\,l \equiv ((\neg A)\,\mathcal{W}\,l) \wedge \Diamond l$, the resolvent would be equivalent to the pair of resolvents $C \Rightarrow (\neg A)\,\mathcal{W}\,l$ and $C \Rightarrow \Diamond l$. The latter is subsumed by the sometime PLTL-clause we have resolved with. So this leaves only the "$\mathcal{W}$" formula.) The resolvent must next be translated into SNF. In previous presentations, for example, Fisher [1991], two resolvents have been given. As the resolvent given here is sufficient for completeness we omit the second.

In SNF we have no PLTL-clauses of the form $A \Rightarrow \bigcirc \Box \neg l$. So the full temporal resolution operation applies between a sometime PLTL-clause and a set of SNF_m PLTL-clauses that together imply $A \Rightarrow \bigcirc \Box \neg l$. The temporal resolution operation, in detail, is

$$\begin{array}{rcl} A_0 & \Rightarrow & \bigcirc B_0 \\ \ldots & \Rightarrow & \ldots \\ A_n & \Rightarrow & \bigcirc B_n \\ C & \Rightarrow & \Diamond l \\ \hline C & \Rightarrow & \left[\bigwedge_{i=0}^{n} (\neg A_i) \right] \mathcal{W}\,l \end{array}$$

with the side conditions that, for all i $0 \leq i \leq n$,

[1]The $\bigcirc$ operator occurs because it is $\bigcirc \Box \neg l$ rather than $\Box \neg l$ that is actually generated from a set of merged SNF step clauses.

$$\vdash B_i \Rightarrow \neg l\,; \text{ and}$$
$$\vdash B_i \Rightarrow \bigvee_{j=0}^{n} A_j\,.$$

Here, the side conditions are simply propositional formulae, so they must hold in (classical) propositional logic. The first side condition ensures that by satisfying any B_i then $\neg l$ will be satisfied. The second shows that once some B_i is satisfied then one of the left-hand sides (A_j) will also be satisfied. Hence, if any A_i is satisfied, then, in the next moment, B_i is satisfied, as is $\neg l$, as is A_j for some j and so on, so that

$$\left(\bigvee_i A_i\right) \Rightarrow \bigcirc\,\square\neg l\,.$$

The set of SNF$_m$ PLTL-clauses $A_i \Rightarrow \bigcirc B_i$ that satisfy these side conditions are together known as *a loop in* $\neg l$. The disjunction of the left-hand side of this set of SNF$_m$ PLTL-clauses, i.e.,

$$\bigvee_i A_i,$$

is known as a *loop formula* for $\neg l$. The most complex part of this approach is the search for the set of SNF$_m$ PLTL-clauses to use in the application of the temporal resolution operation. Detailed explanation of the techniques developed for this search is beyond the scope of this article but is discussed at length in Dixon et al. [1995] and Dixon [1996; 1998].

The resolvent must be translated into SNF before any further resolution steps. A translation to the normal form is given below that avoids the renaming of the subformula

$$\bigwedge_{i=0}^{n} \neg A_i$$

where t is a new proposition symbol and $i = 0, \ldots, n$. Thus, for each of the PLTL-clauses (1), (2), and (5) there are $n+1$ copies, one for each A_i. (N.B., we will see in Section 6.3 that this is important for completeness.)

$$\begin{aligned}
\textbf{start} &\Rightarrow \neg C \vee l \vee \neg A_i && (1)\\
\textbf{true} &\Rightarrow \bigcirc(\neg C \vee l \vee \neg A_i) && (2)\\
\textbf{start} &\Rightarrow \neg C \vee l \vee t && (3)\\
\textbf{true} &\Rightarrow \bigcirc(\neg C \vee l \vee t) && (4)\\
t &\Rightarrow \bigcirc(l \vee \neg A_i) && (5)\\
t &\Rightarrow \bigcirc(l \vee t) && (6)
\end{aligned}$$

We note that only the resolvents (1), (2), and (5) depend on the particular loop being resolved with, i.e., contain a reference to A_i.

4.3 Augmentation

The introduction of new variables, such as t above, makes proofs about the temporal resolution method more difficult. Furthermore, if a temporal resolution proof involves two temporal resolution inferences involving the same literal, we may introduce two new variables where one would suffice. Thus, for n different eventualities we only require n new proposition symbols. We introduce these new proposition symbols at the start of the proof by adding the resolvents that do not contain $\neg A_i$, that is, have no reference to the loop detected (i.e., the PLTL-clauses above labeled 3, 4, and 6) at the beginning and the rest of the PLTL-clauses, if required, as the proof proceeds. The following definitions formalize this technique. Given an eventuality $\Diamond l$, the new proposition symbol introduced is w_l (rather than t above) which can be thought of as *waiting for* l. Hence having translated to SNF and augmented, we can be sure that no new proposition symbols appear during the application of the resolution rules.

Definition 4.3.1 (Augmented PLTL-Clause Sets). Given a set, S, of SNF PLTL-clauses, we construct an augmented set of PLTL-clauses $Aug(S)$ as follows. For each literal l which occurs as an eventuality in S we introduce a new proposition symbol, w_l, and record the correspondence between l and w_l. The variable w_l will be used to record the condition that we are *waiting* for l to occur. The first defining PLTL-clause for w_l is

$$w_l \Rightarrow \bigcirc(l \vee w_l). \tag{7}$$

Then, for each PLTL-clause $C \Rightarrow \Diamond l$, we add both

$$\mathbf{start} \Rightarrow \neg C \vee l \vee w_l \tag{8}$$

$$\mathbf{true} \Rightarrow \bigcirc(\neg C \vee l \vee w_l). \tag{9}$$

Definition 4.3.2. The *loop resolvents* for a sometime PLTL-clause $C \Rightarrow \Diamond l$ and a loop formula $\bigvee_i A_i$ are

$$\mathbf{start} \Rightarrow \neg C \vee l \vee \neg A_i \tag{10}$$

$$\mathbf{true} \Rightarrow \bigcirc(\neg C \vee l \vee \neg A_i) \tag{11}$$

$$w_l \Rightarrow \bigcirc(l \vee \neg A_i) \tag{12}$$

for each i.

Note, the loop resolvents for a particular sometime clause and loop formula are the only clauses added to the clause-set by applying the temporal resolution rule.

4.4 An Algorithm for the Temporal Resolution Method

Given any temporal formula, A, to be tested for unsatisfiability, the following steps are performed.

(1) Translate A into SNF, giving A_s.
(2) Augment A_s, giving $Aug(A_s)$.
(3) Perform step resolution (including simplification and subsumption) on $Aug(A_s)$ until either

(a) **start** $\Rightarrow$ **false** is derived—terminate noting that A is unsatisfiable; or
(b) no new resolvents are generated—continue to step (4).

(4) Select an eventuality from the right-hand side of a sometime PLTL-clause within $Aug(A_s)$, for example $\Diamond l$. Search for loop-formulae for $\neg l$.
(5) Construct loop resolvents for the loop-formulae detected and each sometime PLTL-clause with $\Diamond l$ on the right-hand side. If any new formulae (i.e., that are not subsumed by PLTL-clauses already present) have been generated, go to step (3).
(6) If all eventualities have been resolved, i.e., no new formulae have been generated for any of the eventualities, terminate declaring A satisfiable; otherwise go to step (4).

We will consider the soundness, completeness, and termination of this method in Section 6.

5. EXAMPLES

We illustrate the method by presenting a selection of examples.

5.1 Step Resolution Example

We prove an instance of one of the PLTL axioms that requires only step resolution, namely

$$\vdash \ \bigcirc(a \Rightarrow b) \Rightarrow (\bigcirc a \Rightarrow \bigcirc b).$$

We negate

$$\bigcirc(a \Rightarrow b) \wedge (\bigcirc a \wedge \bigcirc \neg b)$$

and rewrite into SNF as follows.

$$\begin{array}{rrcl}
1. & \textbf{start} & \Rightarrow & f \\
2. & f & \Rightarrow & \bigcirc x \\
3. & \textbf{start} & \Rightarrow & (\neg x \vee \neg a \vee b) \\
4. & \textbf{true} & \Rightarrow & \bigcirc(\neg x \vee \neg a \vee b) \\
5. & f & \Rightarrow & \bigcirc a \\
6. & f & \Rightarrow & \bigcirc \neg b
\end{array}$$

There are no sometime PLTL-clauses, so augmentation adds no new PLTL-clauses. Resolution can be carried out as follows.

$$\begin{array}{rrcll}
7. & f & \Rightarrow & \bigcirc(\neg x \vee \neg a) & \text{[4, 6 Step Resolution]} \\
8. & f & \Rightarrow & \bigcirc \neg x & \text{[5, 7 Step Resolution]} \\
9. & f & \Rightarrow & \bigcirc \textbf{false} & \text{[2, 8 Step Resolution]} \\
10. & \textbf{start} & \Rightarrow & \neg f & \text{[9 Rewriting]} \\
11. & \textbf{true} & \Rightarrow & \bigcirc \neg f & \text{[9 Rewriting]} \\
12. & \textbf{start} & \Rightarrow & \textbf{false} & \text{[1, 10 (Initial) Step Resolution]}
\end{array}$$

A contradiction has been obtained meaning the negated formula is unsatisfiable, and therefore the original formula is valid.

5.2 Temporal Resolution Example (from a Set of Clauses)

Assume we wish to show that the following set of PLTL-clauses (already translated into SNF) is unsatisfiable.

$$
\begin{array}{rrcl}
1. & \textbf{start} & \Rightarrow & f \\
2. & \textbf{start} & \Rightarrow & a \\
3. & \textbf{start} & \Rightarrow & p \\
4. & f & \Rightarrow & \Diamond\neg p \\
5. & f & \Rightarrow & \bigcirc a \\
6. & a & \Rightarrow & \bigcirc(b \vee x) \\
7. & b & \Rightarrow & \bigcirc a \\
8. & b & \Rightarrow & \bigcirc p \\
9. & a & \Rightarrow & \bigcirc p \\
10. & a & \Rightarrow & \bigcirc\neg x
\end{array}
$$

As the set of PLTL-clauses contains a sometime PLTL-clause (no. 4) we augment with the following PLTL-clauses.

$$
\begin{array}{rrcll}
11. & \textbf{start} & \Rightarrow & \neg f \vee \neg p \vee w_{\neg p} & [4\ \text{Augmentation}] \\
12. & \textbf{true} & \Rightarrow & \bigcirc(\neg f \vee \neg p \vee w_{\neg p}) & [4\ \text{Augmentation}] \\
13. & w_{\neg p} & \Rightarrow & \bigcirc(\neg p \vee w_{\neg p}) & [4\ \text{Augmentation}]
\end{array}
$$

Step resolution occurs as follows.

$$
\begin{array}{rrcll}
14. & a & \Rightarrow & \bigcirc b & [6, 10\ \text{Step Resolution}]
\end{array}
$$

Note other step resolution inferences may be performed, for example, between 1 and 11, but we omit them as they play no part in the proof. By merging PLTL-clauses 9 and 14, and 7 and 8 into SNF_m using the merged-SNF rule given in Section 3.1 we obtain the following loop in p (in SNF_m)

$$
\begin{array}{rcll}
a & \Rightarrow & \bigcirc(b \wedge p) & [9, 14\ \text{SNF}_m] \\
b & \Rightarrow & \bigcirc(a \wedge p) & [7, 8\ \text{SNF}_m]
\end{array}
$$

for resolution with PLTL-clause 4. The resolvents after temporal resolution are PLTL-clauses 15–20 below:

$$
\begin{array}{rrcll}
15. & \textbf{start} & \Rightarrow & \neg f \vee \neg p \vee \neg a & [4, 7, 8, 9, 14\ \text{Temporal Resolution}] \\
16. & \textbf{true} & \Rightarrow & \bigcirc(\neg f \vee \neg p \vee \neg a) & [4, 7, 8, 9, 14\ \text{Temporal Resolution}] \\
17. & \textbf{start} & \Rightarrow & \neg f \vee \neg p \vee \neg b & [4, 7, 8, 9, 14\ \text{Temporal Resolution}] \\
18. & \textbf{true} & \Rightarrow & \bigcirc(\neg f \vee \neg p \vee \neg b) & [4, 7, 8, 9, 14\ \text{Temporal Resolution}] \\
19. & w_{\neg p} & \Rightarrow & \bigcirc(\neg p \vee \neg a) & [4, 7, 8, 9, 14\ \text{Temporal Resolution}] \\
20. & w_{\neg p} & \Rightarrow & \bigcirc(\neg p \vee \neg b) & [4, 7, 8, 9, 14\ \text{Temporal Resolution}]
\end{array}
$$

And the proof concludes as follows.

$$
\begin{array}{rrcll}
21. & \textbf{start} & \Rightarrow & \neg f \vee \neg a & [3, 15\ \text{(Initial) Step Resolution}] \\
22. & \textbf{start} & \Rightarrow & \neg f & [2, 21\ \text{(Initial) Step Resolution}] \\
23. & \textbf{start} & \Rightarrow & \textbf{false} & [1, 22\ \text{(Initial) Step Resolution}]
\end{array}
$$

A contradiction has been obtained; hence the set of PLTL-clauses is unsatisfiable.

5.3 Temporal Resolution Example (from a Formula)

Next we show that $\square a \wedge \Diamond \neg a$ is unsatisfiable. First we translate to the normal form.

$$
\begin{array}{rrcl}
1. & \textbf{start} & \Rightarrow & x \\
2. & x & \Rightarrow & \Diamond \neg a \\
3. & \textbf{start} & \Rightarrow & \neg x \vee a \\
4. & \textbf{true} & \Rightarrow & \bigcirc(\neg x \vee a) \\
5. & \textbf{start} & \Rightarrow & \neg x \vee y \\
6. & \textbf{true} & \Rightarrow & \bigcirc(\neg x \vee y) \\
7. & y & \Rightarrow & \bigcirc y \\
8. & y & \Rightarrow & \bigcirc a
\end{array}
$$

As the set of PLTL-clauses contains a sometime PLTL-clause (no. 2) we augment with the following PLTL-clauses.

$$
\begin{array}{rrcll}
9. & \textbf{start} & \Rightarrow & \neg x \vee \neg a \vee w_{\neg a} & \text{[2 Augmentation]} \\
10. & \textbf{true} & \Rightarrow & \bigcirc(\neg x \vee \neg a \vee w_{\neg a}) & \text{[2 Augmentation]} \\
11. & w_{\neg a} & \Rightarrow & \bigcirc(\neg a \vee w_{\neg a}) & \text{[2 Augmentation]}
\end{array}
$$

We can find a loop for resolution with PLTL-clause 2 by merging 7 and 8 to give

$$y \Rightarrow \bigcirc(y \wedge a).$$

One of the resolvents obtained is PLTL-clause 12 from which we can derive a contradiction.

$$
\begin{array}{rrcll}
12. & \textbf{start} & \Rightarrow & \neg x \vee \neg a \vee \neg y & \text{[2, 7, 8 Temporal Resolution]} \\
13. & \textbf{start} & \Rightarrow & \neg x \vee \neg a & \text{[5, 12 (Initial) Step Resolution]} \\
14. & \textbf{start} & \Rightarrow & \neg x & \text{[3, 13 (Initial) Step Resolution]} \\
15. & \textbf{start} & \Rightarrow & \textbf{false} & \text{[1, 14 (Initial) Step Resolution]}
\end{array}
$$

5.4 A Larger Example

Here we conclude the example introduced in Section 3.4. Recall we are trying to show that

$$(\Diamond p \wedge \square(p \Rightarrow \bigcirc p)) \Rightarrow \Diamond \square p$$

is valid. We negated and translated the formula into SNF in Section 3.4. The PLTL-clauses in normal form are repeated here although they have been renumbered sequentially. We only show the steps relevant to the refutation.

$$
\begin{array}{rrcl@{\qquad}rrcl}
1. & \textbf{start} & \Rightarrow & f & 10. & \textbf{start} & \Rightarrow & (\neg q \vee \neg p \vee s) \\
2. & f & \Rightarrow & \Diamond p & 11. & \textbf{true} & \Rightarrow & \bigcirc(\neg q \vee \neg p \vee s) \\
3. & r & \Rightarrow & \bigcirc q & 12. & t & \Rightarrow & \Diamond \neg p \\
4. & r & \Rightarrow & \bigcirc r & 13. & u & \Rightarrow & \bigcirc t \\
5. & \textbf{start} & \Rightarrow & \neg f \vee q & 14. & u & \Rightarrow & \bigcirc u \\
6. & \textbf{true} & \Rightarrow & \neg f \vee q & 15. & \textbf{start} & \Rightarrow & \neg f \vee t \\
7. & \textbf{start} & \Rightarrow & \neg f \vee r & 16. & \textbf{true} & \Rightarrow & \bigcirc(\neg f \vee t) \\
8. & \textbf{true} & \Rightarrow & \neg f \vee r & 17. & \textbf{start} & \Rightarrow & \neg f \vee u \\
9. & s & \Rightarrow & \bigcirc p & 18. & \textbf{true} & \Rightarrow & \bigcirc(\neg f \vee u)
\end{array}
$$

Next, we augment the set of PLTL-clauses to account for the two sometime PLTL-clauses 2 and 12.

19.	$\mathbf{start} \Rightarrow (\neg f \vee w_p \vee p)$		[2 Augmentation]
20.	$\mathbf{true} \Rightarrow \bigcirc(\neg f \vee w_p \vee p)$		[2 Augmentation]
21.	$w_p \Rightarrow \bigcirc(w_p \vee p)$		[2 Augmentation]
22.	$\mathbf{start} \Rightarrow (\neg t \vee w_{\neg p} \vee \neg p)$		[12 Augmentation]
23.	$\mathbf{true} \Rightarrow \bigcirc(\neg t \vee w_{\neg p} \vee \neg p)$		[12 Augmentation]
24.	$w_{\neg p} \Rightarrow \bigcirc(w_{\neg p} \vee \neg p)$		[12 Augmentation]

Step resolution then begins.

25.	$r \Rightarrow \bigcirc(\neg p \vee s)$	[3, 11 Step Resolution]
26.	$(s \wedge r) \Rightarrow \bigcirc s$	[9, 25 Step Resolution]

By merging PLTL-clauses 4, 9, and 26 into SNF$_m$ we obtain the loop

$$(s \wedge r) \Rightarrow \bigcirc(s \wedge r \wedge p)$$

for resolution with PLTL-clause 12. This generates additional PLTL-clauses (from the resolvent) as follows.

27.	$\mathbf{start} \Rightarrow (\neg t \vee \neg s \vee \neg r \vee \neg p)$	[4, 9, 26, 12 Temporal Resolution]
28.	$\mathbf{true} \Rightarrow \bigcirc(\neg t \vee \neg s \vee \neg r \vee \neg p)$	[4, 9, 26, 12 Temporal Resolution]
29.	$w_{\neg p} \Rightarrow \bigcirc(\neg s \vee \neg r \vee \neg p)$	[4, 9, 26, 12 Temporal Resolution]

Thus, the refutation continues as follows.

30.	$\mathbf{true} \Rightarrow \bigcirc(\neg t \vee \neg r \vee \neg p \vee \neg q)$	[11, 28 Step Resolution]
31.	$r \Rightarrow \bigcirc(\neg t \vee \neg p \vee \neg q)$	[4, 30 Step Resolution]
32.	$r \Rightarrow \bigcirc(\neg t \vee \neg p)$	[3, 31 Step Resolution]
33.	$(r \wedge u) \Rightarrow \bigcirc\neg p$	[13, 32 Step Resolution]

Now by merging PLTL-clauses 4, 14, and 33

$$(r \wedge u) \Rightarrow \bigcirc(r \wedge u \wedge \neg p)$$

we have a loop for resolution with PLTL-clause 2, which generates several resolvents, including PLTL-clause 34.

34.	$\mathbf{start} \Rightarrow (\neg f \vee \neg r \vee \neg u \vee p)$	[2, 4, 14, 33 Temporal Resolution]
35.	$\mathbf{start} \Rightarrow (\neg f \vee \neg r \vee \neg u \vee \neg q \vee s)$	[10, 34 (Initial) Step Resolution]
36.	$\mathbf{start} \Rightarrow (\neg f \vee \neg r \vee \neg u \vee \neg q \vee \neg t \vee \neg p)$	[27, 35 (Initial) Step Resolution]
37.	$\mathbf{start} \Rightarrow (\neg f \vee \neg r \vee \neg u \vee \neg q \vee \neg t)$	[34, 36 (Initial) Step Resolution]
38.	$\mathbf{start} \Rightarrow (\neg f \vee \neg r \vee \neg q \vee \neg t)$	[17, 37 (Initial) Step Resolution]
39.	$\mathbf{start} \Rightarrow (\neg f \vee \neg r \vee \neg q)$	[15, 38 (Initial) Step Resolution]
40.	$\mathbf{start} \Rightarrow (\neg f \vee \neg q)$	[7, 39 (Initial) Step Resolution]
41.	$\mathbf{start} \Rightarrow \neg f$	[5, 40 (Initial) Step Resolution]
42.	$\mathbf{start} \Rightarrow \mathbf{false}$	[1, 41 (Initial) Step Resolution]

6. CORRECTNESS

First we show that augmentation is satisfiability preserving. Next, a soundness result is obtained by showing that an application of the step or temporal resolution rule preserves satisfiability. Finally completeness is proved by considering the construction of a graph representing all possible models of the augmented set of PLTL-clauses. Here, deletions of parts of the graph that cannot be used to construct models are associated with step and resolution rules.

6.1 Augmented PLTL-Clause Sets

We will show that an augmented PLTL-clause set has a *model* if, and only if, its underlying (nonaugmented) PLTL-clause set has a model.

Definition 6.1.1. Given a set, S, of SNF PLTL-clauses, a *normal model* for the augmented PLTL-clause set for S is a model which satisfies the formula

$$\Box(w_l \Leftrightarrow (\neg l \wedge \Diamond l)) \tag{13}$$

for each literal l which occurs as an eventuality (i.e., inside the scope of a $\Diamond$ operator) in S.

Definition 6.1.2. An augmented PLTL-clause set is said to be *well-behaved* if it is either unsatisfiable or has a normal model.

LEMMA 6.1.1 (AUGMENTATION). *If S is a set of SNF PLTL-clauses then*

(1) $Aug(S)$ is well-behaved and

(2) $Aug(S)$ has a model if and only if S has a model.

PROOF. If $Aug(S)$ has a model then ignoring the value of each w_l at each moment gives a model for S. Conversely, if S has a model M, then M can be extended to a model M' for $Aug(S)$ by giving w_l the same truth value as $\neg l \wedge \Diamond l$ in M in each state, and for each literal l. The model M' clearly satisfies the formulae (7), (8), and (9) from Section 4.3 and (13) above. The lemma follows easily from these two observations. □

6.2 Soundness

6.2.1 *Step Resolution Rules.* It is easy to see that given a satisfiable set of PLTL-clauses the application of the initial or step resolution inferences, or simplification, preserves satisfiability.

6.2.2 *Temporal Resolution Rule.* The following lemma is a soundness result for the temporal resolution rule (applied to augmented PLTL-clause sets).

LEMMA 6.2.1 (SOUNDNESS). *Let S be a well-behaved augmented PLTL-clause set. Let the PLTL-clause set T be obtained from S by application of the temporal resolution operation. Then*

(1) T is well-behaved and

(2) if S is satisfiable then T is satisfiable.

PROOF. If S is satisfiable then S has a model, and by Lemma 6.1.1 it has a normal model M. The side conditions for temporal resolution guarantee that the loop resolvents, i.e., formulae (10), (11), and (12) given in Section 4.3 hold in M, and so M is a (normal) model for T, i.e., T is satisfiable. If S is unsatisfiable then the addition of PLTL-clauses to produce T is also unsatisfiable. Hence T is well-behaved. □

6.3 Completeness

We will now prove the completeness of the temporal resolution procedure by induction on the size of a *behavior graph* of a set of SNF PLTL-clauses. Note, as we have added all the new variables required for the translation of the unless operator by augmentation in Section 6.1 and avoided renaming the conjunction that occurs from negating the loop-formula (a disjunction) as mentioned in Section 4.2 we require no new proposition symbols during the proof. Thus the graph constructed has all the propositional symbols we require and will not increase in size during the proof.

Definition 6.3.1 (Behavior Graph). Given a set S of SNF PLTL-clauses, we construct a finite directed graph G as follows. The nodes of G are all ordered pairs (V, E) where

—V is a valuation of the proposition symbols occurring in S, i.e., V contains either p or $\neg p$ for each proposition symbol p in S; and
—E is a subset of the literals occurring as eventualities in S, i.e., literals occurring on the right-hand side of the sometime PLTL-clauses in S.

For each node (V, E), let

—R be the set of step PLTL-clauses of S which are "fired" by V—that is, the set of step PLTL-clauses whose left-hand sides are satisfied by V;
—L be the set of clauses on the right-hand sides of the PLTL-clauses in R, i.e., L contains formulae that are the disjunction of literals from the right-hand side of each PLTL-clause in R having first removed the next operator;
—E' be the set of elements of E which are not satisfied by V.

For each valuation V' which satisfies L, let E'' be the set of literals occurring on the right-hand sides of the sometime PLTL-clauses fired by V'. Then for each V' construct an edge in G from (V, E) to $(V', E' \cup E'')$. These are the only edges originating from (V, E).

Let L_0 be the set of initial PLTL-clauses of S. For each valuation V which satisfies L_0, where E' is the set of literals occurring on the right-hand sides of the sometime PLTL-clauses fired by V, the node (V, E') is designated as an *initial node* of G. The behavior graph of S is the full subgraph of G given by the set of nodes reachable from the initial nodes. We regard the identification of the initial nodes as part of the structure of the behavior graph.

LEMMA 6.3.1. *Let S be a set of SNF PLTL-clauses, and let T be the set of SNF PLTL-clauses obtained from S by adding finitely many initial PLTL-clauses and finitely many step PLTL-clauses which only involve proposition symbols* *occurring in S. Then the behavior graph of T is a subgraph of the behavior graph of S.*

PROOF. This is established by induction on the length of the shortest path from an initial node to an arbitrary node in the behavior graph of T. Let len be the length of the shortest path from an initial node to a node n. To show the base case we let $len = 0$ and show that any initial node in the behavior graph of T is an initial node in the behavior graph of S. Let $I \subseteq S$ be the initial PLTL-clauses of S and $I' \subseteq T$ the initial PLTL-clauses of T. As T has been constructed by adding initial and/or step PLTL-clauses to S, $I \subseteq I'$. Take any initial node $n_0 = (V_0, E_0)$ in the behavior graph for T. From the definition of the behavior graph V_0 must satisfy the right-hand side of the initial PLTL-clauses in I'. As $I \subseteq I'$ then V_0 must also satisfy the right-hand side of the PLTL-clauses in I. As the set of sometime PLTL-clauses in S and T is unchanged, i.e., as V_0 satisfies the left-hand side of the same sometime PLTL-clauses in S and T the set E_0 will be the same in each graph for V_0, and thus the node $n_0 = (V_0, E_0)$ is also in the behavior graph for S.

Next we assume that if any node n, where the length of the shortest path from an initial node to n is m, is in the behavior graph for T, it is also in the behavior graph for S. We show that any node n' in the behavior graph for T whose shortest path length from an initial node is $m + 1$, is also in the behavior graph for S. Let $J \subseteq S$ be the step PLTL-clauses in S and $J' \subseteq T$ the step PLTL-clauses in T. By assumption we have $J \subseteq J'$. Consider some node $n' = (V', E')$ in the behavior graph of T where the shortest path from an initial node to n' is $m + 1$. Let $n = (V, E)$ be any node in the behavior graph for T such that there is an edge from n to n' and such that the shortest path from an initial node to n is of length m. By the induction hypothesis, we assume that n is also in the behavior graph for S.

Let $X' \subseteq J'$ be the set of step PLTL-clauses in T such that the left-hand sides are satisfied by V, while the right-hand sides are satisfied by V'. Let $X \subseteq J$ be the corresponding set of step PLTL-clauses in S, i.e., where the left-hand sides are satisfied by V, while the right-hand sides are satisfied by V'. As $J \subseteq J'$ we have $X \subseteq X'$. Furthermore as no change has been made to the set of sometime PLTL-clauses any eventualities outstanding from n or triggered by n' will be the same in each graph. Thus n' is also present in the behavior graph for S. □

LEMMA 6.3.2. *Any model for a set of SNF-PLTL-clauses, S, can be constructed from a path through the behavior graph for S.*

PROOF. To construct a model from a suitable path, $N_0, N_1, N_2, \ldots$ where each $N_i = (V_i, E_i)$, through the behavior graph (i.e., one which is infinite and in which all eventualities are satisfied) take the valuation V_i from each node N_i in the path (and delete any negated proposition symbols). Any proposition symbols that do not occur in S but are required in the model may be set arbitrarily. Details of how to construct models from behavior graphs are given in Lemma 6.3.5.

Take any model $\sigma = s_0, s_1, \ldots$ for S. We show that this model can be constructed from a path through the behavior graph. First delete any proposition symbols not in S from σ to give $\sigma' = s'_0, s'_1, \ldots$. As these proposition symbols do not occur in S they have no constraints on them, so by setting these proposition symbols to true and false in the correct way we can recover σ. Note that σ' is a model for S. By definition the behavior graph for S is the reachable subgraph from the set of initial nodes. The behavior graph has been constructed where the V component of each node consists of every possible valuation. Let $\text{pos}(V_i)$ be the set of nonnegated proposition symbols in V_i. As σ' is a model for S, s'_0 must satisfy the initial rules $I \subseteq S$. To construct the behavior graph for S the initial nodes are those with valuations that satisfy I, for a particular E component. As nodes are constructed with each valuation and subset of eventualities there must be a node $N_0 = (V_0, E_0)$ where $\text{pos}(V_0) = s'_0$.

Next for some s'_i in σ' assume that there is a node $N_i = (V_i, E_i)$ in the behavior graph for S such that $\text{pos}(V_i) = s'_i$. We show that $\text{pos}(s'_{i+1}) = V_{i+1}$ for some node $N_{i+1} = (V_{i+1}, E_{i+1})$ in the behavior graph for S. Let $R \subseteq S$ be the set of step PLTL-clauses in S. Take the set of step PLTL-clauses $R' \subseteq R$ such that the left-hand side of the PLTL-clauses in R' is satisfied by V_i. As $\text{pos}(V_i) = s'_i$, s'_i must satisfy the left-hand side of the PLTL-clauses in R'. As σ' is a model for S, s'_{i+1} must satisfy the right-hand side of each PLTL-clauses in R' having deleted the next operator. From the construction of the behavior graph, edges are drawn from N_i to nodes whose valuation satisfies the right-hand side of each PLTL-clauses in R' having deleted the next operator (for some E component). As nodes have been constructed for all valuation/eventuality component combinations there will be one $N_{i+1} = (V_{i+1}, E_{i+1})$ such that $\text{pos}(V_{i+1}) = s'_{i+1}$.

Hence we can construct σ' using the valuations from each node and following a path through the behavior graph for S. This can be extended to σ by setting the additional proposition symbols as required. □

LEMMA 6.3.3. *Let S be a set of PLTL-clauses and T be the set of clauses obtained from S by applying one simplification or subsumption step. The behavior graph for S is the same as the behavior graph for T.*

PROOF. First assume we have performed a simplification step. We show that any node and edge that is in the behavior graph for S is also in the behavior graph for T. The proof of the converse is similar. The proof is by induction on the length of the shortest path from an initial node. For the base case the length of the path from an initial node to n is 0, i.e., n is an initial node. If the simplification step has not been performed on an initial PLTL-clause i.e., the set of initial PLTL-clauses in S and in T are the same, then n must also be in the behavior graph for T. Otherwise we have performed a simplification step on an initial PLTL-clause, i.e., S contains $\textbf{start} \Rightarrow Y$ and T contains $\textbf{start} \Rightarrow Y'$ where $Y \equiv Y'$. Each initial node n in the behavior graph for S satisfies Y by definition of the behavior graph. As $Y \equiv Y'$ node n also satisfies Y', so n is in the behavior graph for T.

Next assume the node n in the behavior graph for S, whose shortest path distance from an initial node is m, is also in the behavior graph for T. We show that any node of shortest path length $m + 1$ from an initial node is also in the behavior graph for T. Take a node n'' in the behavior graph for S whose shortest path length from an initial node is $m+1$. Consider n' such that (n', n'') is an edge in the behavior graph from S where the shortest path length from n' to an initial node is m. From the induction hypothesis n' is also in the behavior graph for T. Assume that a simplification step has been applied to rule $X \Rightarrow \bigcirc Y \in S$ to obtain $X' \Rightarrow \bigcirc Y' \in T$ and that n' satisfies X. Thus from the definition of the behavior graph n'' must satisfy Y. As we have performed a simplification step $X \equiv X'$ and $Y \equiv Y'$, so n' also satisfies X' and n'' satisfies Y', as the sets S and T are unchanged apart from this. Hence n'' and the edge (n', n'') must also be in T. If the node n' did not satisfy X, or the simplification rule had been on an initial PLTL-clause, then n'' would again be in the behavior graph for T as the remaining rules are unchanged. The proof of the converse is similar.

To show the proof holds for a subsumption step assume S contain rules $X \Rightarrow \bigcirc Y$ and $X' \Rightarrow \bigcirc Y'$ where $X \Rightarrow X'$ and $Y' \Rightarrow Y$. Thus by a subsumption step $T = S \backslash \{X \Rightarrow \bigcirc Y\}$. The proof is similar to the above. □

We now introduce the concept of a *reduced behavior graph*, which will be used later in the completeness proof.

Definition 6.3.2 (Reduced Behavior Graph). Given a behavior graph we apply the following rules repeatedly until no more deletions are possible.

—If a node has no successors, delete that node (and all edges to the node).

—If a node $n = (V, E)$ contains an eventuality l (i.e., $l \in E$) and l is not satisfied in n, i.e., $l \notin V$, and there is no path from n to a node whose valuation satisfies l, then delete n.

The resulting graph is called the *reduced behavior graph* for S.

This terminology implies that the reduced graph does not depend on the order of deletions. The proof of this fact is straightforward, but is not necessary for the completeness proof—we only need to know that *a* reduced graph (one from which no further deletions are permitted) exists.

LEMMA 6.3.4. *During the construction of a reduced behavior graph any node reachable from a deleted node is also deleted.*

PROOF. There are two conditions for the deletions of nodes to form a reduced behavior graph. Firstly nodes with no successors are deleted. No nodes are reachable from a node with no successors; hence the lemma follows. Secondly nodes $n = (V, E)$ that are deleted where l is an outstanding eventuality, i.e., $l \in E$ but no reachable node satisfies l, i.e., $\neg l \in V$. From the construction of the behavior graph and from the conditions allowing us to delete n, any node $n' = (V', E')$ reachable from n must contain l as an outstanding eventuality, i.e., $l \in E$ and but does not satisfy l. Thus any node reachable from n must also be deleted. □

LEMMA 6.3.5. *A set of SNF PLTL-clauses is unsatisfiable if, and only if, its reduced behavior graph is empty.*

PROOF. Let S be a set of SNF PLTL-clauses. An infinite path through the (unreduced) behavior graph for S, starting at an initial node, gives a sequence of valuations for the propositional symbols—i.e., a PLTL model. By construction of the graph, this model satisfies the initial and step PLTL-clauses of S. Furthermore, by Lemma 6.3.2 any such model must arise from a path through the behavior graph. However, not all paths give models for the full set of PLTL-clauses S, since either the paths may not be infinite or they may fail to satisfy some eventualities (which occur within sometime PLTL-clauses). If a node, n, has no successors, then there are no infinite paths through that node, so any model for S must arise from a path through the graph with n deleted. Thus the first deletion criterion can be applied without removing any potential models. Also, if a node n contains an eventuality l then any path through that node which is to yield a model for S must satisfy l either at n or somewhere later in the path. Thus, if a node contains an eventuality that cannot be satisfied then this node cannot be part of a model for the set of PLTL-clauses; hence, we can apply the second deletion criterion without discarding potential models for S. The "if" part of the proposition follows.

To prove the "only if" part, suppose the reduced behavior graph for S, call it G, is nonempty. We will now use G to construct a model for S. First note that the set of initial nodes in G is nonempty, since, in the behavior graph, every node is reachable from the initial nodes and since any node reachable from a deleted node is also deleted (by Lemma 6.3.4). Now, choose an initial node $n_0 = (V_0, E_0)$. If E_0 is nonempty, choose an ordering $e_1, \ldots, e_k$ for the literals in E_0. Since n_0 has not been deleted, there is a path in G to a node $m_{0,1}$ in which the eventuality e_1 is satisfied. If the eventuality e_2 is not present in $m_{0,1}$ it must have been satisfied somewhere along the path. Otherwise, we can extend the path to a node $m_{0,2}$, which satisfies e_2. Continuing in this way we can find a path P_1 (which may consist simply of the node n_0 if all of E_0 are satisfied there) such that each element of E_0 is satisfied at some point along P_1. Let n_1 be a successor of the end point of P_1 (it must have a successor, since we have deleted all terminal nodes). Repeating our construction, we can find a path P_2 beginning at n_1 along which all the eventualities in n_1 are satisfied. Let n_2 be a successor of the end point of P_2. Repeat this construction until $n_i = n_j$ for some $i > j$, which must happen eventually, since G is finite. Let Q be the path $P_{i+1} \ldots P_j$. Then the path $P = P_1 P_2 \ldots P_i Q\, Q \ldots$ has the property that, for each node in the path, each eventuality in that node is satisfied at some node later in the path. To see this, recall that if a node contains an eventuality e but does not satisfy e, then e is in the eventuality set of all immediate successors of l. So, either e is satisfied before we reach the next n_r or e is an eventuality in n_r and so is satisfied along P_r. Furthermore P is obviously an infinite path. It follows by the construction of the behavior graph that the sequence of valuations given by P is a model for S. □

We are now ready to prove the completeness theorem for propositional clausal temporal resolution.

THEOREM 6.3.3 (COMPLETENESS). *If a well-behaved augmented PLTL-clause set, S, is unsatisfiable then the temporal resolution procedure will derive a refutation when applied to S.*

PROOF. The proof proceeds by induction on the number of nodes in the behavior graph of S.

First we consider the effect of simplification and subsumption rules on the behavior graph for a set of PLTL-clauses. Given a set of PLTL-clauses S let the application of simplification and subsumption rules to S result in the set of PLTL-clauses S'. By Lemma 6.3.3 the behavior graph of S is identical to that of S'.

If the behavior graph is empty, then the set of initial PLTL-clauses in S is unsatisfiable. By the completeness of classical resolution, we can use step resolution on the set of initial PLTL-clauses to derive the empty clause.

Now suppose the behavior graph G is nonempty. By Lemma 6.3.5, the reduced behavior graph is empty, and so there must be a node which can be deleted from G. If G has a terminal node $n = (V, E)$, let R be the set of step PLTL-clauses whose left-hand sides are satisfied by V. Then, having deleted the next operator, the right-hand side of the PLTL-clauses in R forms an unsatisfiable set L of propositional clauses. By completeness of classical resolution again, there is a refutation of L. Choosing an element of R corresponding to each element of L, we can "mimic" this classical refutation by step resolution inferences to derive a step PLTL-clause

$$l_1 \wedge \ldots \wedge l_k \Rightarrow \bigcirc \mathbf{false} \tag{14}$$

where each l_i is a literal which is satisfied by V. The temporal resolution procedure allows us to rewrite PLTL-clause (14) as

$$\mathbf{start} \;\Rightarrow\; \neg l_1 \vee \ldots \vee \neg l_k \tag{15}$$

$$\mathbf{true} \;\Rightarrow\; \bigcirc(\neg l_1 \vee \ldots \vee \neg l_k). \tag{16}$$

By Lemma 6.3.1, adding PLTL-clauses (15) and (16) (and any other resolvents derived along the way) to S produces a PLTL-clause set T whose behavior graph H is a subgraph of G. (H is in fact a proper subgraph, since H has no node whose valuation is V. If n was an initial node it does not satisfy the initial PLTL-clause (15) as $l_i \in V$ for $i = 1 \ldots k$. If n was a noninitial node, as the left-hand side **true** is satisfied by every node in G the successor of any node must also satisfy $(\neg l_1 \vee \ldots \vee \neg l_k)$. As we have $l_i \in V$ for $i = 0 \ldots k$ no edges can be drawn to n, so H does not contain n.) Furthermore, T is well-behaved, since it has exactly the same models as S. By induction, T, and hence S, has a refutation.

If G does not have a terminal node, then it must contain a node $n = (V, E)$ such that some eventuality $l \in E$ is not satisfied at any node reachable from n. Let N be the set of nodes reachable from n. For each $n_i = (V_i, E_i) \in N$, let R_i be the set of step PLTL-clauses in S whose left-hand sides are satisfied by V_i. Let

$$A_i \Rightarrow \bigcirc B_i \tag{17}$$

be an SNF_m PLTL-clause that is the result of applying the SNF_m merging operation to the PLTL-clauses in R_i. Note that A_i is the conjunction of the left-hand side of the PLTL-clauses in R_i, B_i is the conjunction of the right-hand sides of the PLTL-clauses in R_i (contained in the next operator), and V_i satisfies A_i. Note also that A_i and B_i are simply classical propositional formulae. Then each B_i logically implies $\neg l$, since none of the V_i in N satisfy l. Each $n_i \in N$ leads to a node n_j satisfying B_i for some i. Thus n_j must satisfy $B_i \wedge l$ or $B_i \wedge \neg l$. By definition each successor of a node in N is also in N (as l is unsatisfied in all nodes reachable from n_i). As l is not satisfied by any node in N we have $B_i \wedge l$ is unsatisfiable, and thus $B_i \Rightarrow \neg l$ is valid (in classical propositional logic).

Also each B_i logically implies the disjunction of the A_i's corresponding to the successors of n_i. As each node $n_i \in N$ leads to a node $n_j = (V_j, E_j)$ that satisfies B_i, by definition $n_j \in N$ and V_j satisfies A_j. Thus $B_i \wedge \neg \bigvee_k A_k$ is unsatisfiable. Hence $B_i \Rightarrow \bigvee_k A_k$. Hence, we can use SNF$_m$ PLTL-clauses of the form (17) in an application of temporal resolution. Let A be the disjunction of the A_i. Then each V_i satisfies $\neg l \wedge A$. For each node n_i in N either there is a PLTL-clause $C \Rightarrow \Diamond l$ in S and the valuation at n_i satisfies C, or for each predecessor p_i of n_i the valuation at p_i satisfies w_l.

Let T be the result of adding the loop resolvents (10), (11), and (12) from Section 4.3, and let H be the behavior graph for T. Then H has no nodes from the set N. So H is a proper subgraph of G by Lemma 6.3.1, and T is well-behaved by Lemma 6.2.1. Once again, it follows by induction that there is a refutation for S. □

6.4 Termination

THEOREM 6.4.1. *The resolution algorithm will terminate.*

PROOF. Following the translation to normal form the set of PLTL-clauses is augmented, so no new proposition symbols are required during the proof. Hence we have a finite number of proposition symbols. Further, there are a finite number of right and left-hand sides we may obtain as initial and step PLTL-clauses modulo ordering of the conjunctions or disjunctions. Simplification rules mean that the left or right-hand sides cannot grow indefinitely. Note that the number of sometime PLTL-clauses does not change. Thus step (3) of the algorithm in Section 4.4 either generates **start** $\Rightarrow$ **false** and terminates, or we have tried to resolve each PLTL-clause with every other and obtained no new PLTL-clauses, i.e., something that is not in the set already (modulo ordering of conjunctions/disjunctions).

The argument is similar for the termination of step 5. Having augmented the set of PLTL-clauses with the new proposition symbols needed to translate resolvents from temporal resolution into SNF, at some point no new resolvents will be generated, as we have a finite set of possible PLTL-clauses. □

7. COMPLEXITY

We consider the increase in number of proposition symbols and PLTL-clauses generated by the translation to SNF followed by consideration of the complexity of the resolution proof method.

7.1 Translation to the Normal Form

We consider two aspects of the complexity of translating an arbitrary formula to SNF in detail, namely the maximum number of SNF PLTL-clauses generated from a formula of a given size, and the number of new proposition symbols introduced. Note in this section we do not include the new w_l proposition symbols, as we consider this to be part of the resolution method itself.

7.1.1 *Number of PLTL-Clauses Generated.* We define the length "len" of a formula A as follows.

$$\begin{aligned}
\mathsf{len}(\Diamond l) &= 1 && (l \text{ is a literal})\\
\mathsf{len}(l_1 \vee l_2 \ldots \vee l_n) &= 1 && (l_i \text{ are literals and } n \geq 1)\\
\mathsf{len}(\mathbf{const}) &= 1 && (\mathbf{const} \text{ is one of } \mathbf{true}, \neg\mathbf{true}, \mathbf{false} \text{ or } \neg\mathbf{false})\\
\mathsf{len}(\bigcirc(l_1 \vee l_2 \ldots \vee l_n)) &= 1 && (l_i \text{ are literals and } n \geq 1)
\end{aligned}$$

$$\begin{aligned}
\mathsf{len}(\Box A) &= \\
\mathsf{len}(\Diamond A) &= && (A \text{ not a literal})\\
\mathsf{len}(\bigcirc A) &= 1 + \mathsf{len}(A) && (A \text{ not a disjunction of literals})
\end{aligned}$$

$$\begin{aligned}
\mathsf{len}(\neg\Box A) &= \\
\mathsf{len}(\neg\Diamond A) &= \\
\mathsf{len}(\neg\bigcirc A) &= 1 + \mathsf{len}(\neg A)
\end{aligned}$$

$$\begin{aligned}
\mathsf{len}(A \vee B) &= && (A \text{ and } B \text{ not disjunctions of literals})\\
\mathsf{len}(A \wedge B) &= \\
\mathsf{len}(A \mathcal{U} B) &= \\
\mathsf{len}(A \mathcal{W} B) &= 1 + \mathsf{len}(A) + \mathsf{len}(B)
\end{aligned}$$

$$\begin{aligned}
\mathsf{len}(\neg(A \vee B)) &= \\
\mathsf{len}(\neg(A \wedge B)) &= \\
\mathsf{len}(\neg(A \mathcal{U} B)) &= \\
\mathsf{len}(\neg(A \mathcal{W} B)) &= 1 + \mathsf{len}(\neg A) + \mathsf{len}(\neg B)
\end{aligned}$$

$$\begin{aligned}
\mathsf{len}(\neg(A \Rightarrow B)) &= 1 + \mathsf{len}(A) + \mathsf{len}(\neg B)\\
\mathsf{len}(A \Rightarrow B) &= 1 + \mathsf{len}(\neg A) + \mathsf{len}(B)
\end{aligned}$$

LEMMA 7.1.1. *For any proposition symbol x and PLTL formula W, the maximum number of PLTL-clauses, generated from the translation of $\tau_1[\Box(x \Rightarrow W)]$, denoted by* $\mathsf{clauses}(\tau_1[\Box(x \Rightarrow W)])$*, will be at most* $11 \times \mathsf{len}(W)$*, i.e.,*

$$\mathsf{clauses}(\tau_1[\Box(x \Rightarrow W)]) \leqslant (11 \times \mathsf{len}(W)).$$

PROOF. The proof is by induction on the length of W. The base case is where W has length 1, i.e., it has the form $\Diamond l, l_1 \vee \ldots \vee l_n$, **true**, **false**, $\bigcirc(l_1 \vee \ldots \vee l_n)$. As illustrated in Section 3.2 $\tau_1[\Box(x \Rightarrow \Diamond l)]$ produces one PLTL-clause, $\tau_1[\Box(x \Rightarrow (l_1 \vee \ldots \vee l_n))]$ produces two PLTL-clauses, $\tau_1[\Box(x \Rightarrow \mathbf{const})]$ produces two PLTL-clauses (where **const** is **true**, $\neg$**true**, **false** or $\neg$**false**), and $\tau_1[\Box(x \Rightarrow \bigcirc(l_1 \vee \ldots \vee l_n))]$ produces one PLTL-clause. In each case if the number of PLTL-clauses produced is M, then

$$M \leqslant (11 \times 1).$$

For the inductive hypothesis we assume that the theorem holds for formula of length n and examine each case for length $n+1$. By considering the proofs in Lemma A.1 in the Appendix, the maximum number of PLTL-clauses from removing any operator (or negated operator) is 11 (from $\neg(A\,\mathcal{W}\,B)$).

$$\begin{aligned}
\mathsf{clauses}(\tau_1[\,\square(x \Rightarrow \neg(A\,\mathcal{W}\,B))]) &= 11 + \mathsf{clauses}(\tau_1[\,\square(y \Rightarrow \neg A)]) \\
&\quad + \mathsf{clauses}(\tau_1[\,\square(z \Rightarrow \neg B)]) \\
&\leqslant (11 + (11 \times \mathsf{len}(\neg A)) + (11 \times \mathsf{len}(\neg B))) \\
&= 11(1 + \mathsf{len}(\neg A) + \mathsf{len}(\neg B)) \\
&= 11 \times \mathsf{len}(\neg(A\,\mathcal{W}\,B))
\end{aligned}$$

$$\begin{aligned}
\mathsf{clauses}(\tau_1[\,\square(x \Rightarrow (A\,\mathcal{W}\,B))]) &= 6 + \mathsf{clauses}(\tau_1[\,\square(y \Rightarrow A)]) \\
&\quad + \mathsf{clauses}(\tau_1[\,\square(z \Rightarrow B)]) \\
&\leqslant (6 + (11 \times \mathsf{len}(A)) + (11 \times \mathsf{len}(B))) \\
&\leqslant 11(1 + \mathsf{len}(A) + \mathsf{len}(B)) \\
&= 11 \times \mathsf{len}(A\,\mathcal{W}\,B)
\end{aligned}$$

$$\begin{aligned}
\mathsf{clauses}(\tau_1[\,\square(x \Rightarrow \,\square A)]) &= 6 + \mathsf{clauses}(\tau_1[\,\square(y \Rightarrow A)]) \\
&\leqslant (6 + (11 \times \mathsf{len}(A))) \\
&\leqslant 11(1 + \mathsf{len}(\neg A)) \\
&= 11 \times \mathsf{len}(\,\square A)
\end{aligned}$$

$$\begin{aligned}
\mathsf{clauses}(\tau_1[\,\square(x \Rightarrow (\neg\,\square A))]) &= 1 + \mathsf{clauses}(\tau_1[\,\square(y \Rightarrow \neg A)]) \\
&\leqslant (1 + (11 \times \mathsf{len}(\neg A))) \\
&\leqslant 11(1 + \mathsf{len}(\neg A)) \\
&= 11 \times \mathsf{len}(\neg\,\square A)
\end{aligned}$$

The cases for the other operators are similar. □

THEOREM 7.1.1. *For any PLTL formula W, the maximum number of PLTL-clauses generated from the translation into SNF will be at most* $1 + (11 \times \mathsf{len}(W))$*, i.e.,*

$$\mathsf{clauses}(\tau_0[W]) \leqslant (1 + (11 \times \mathsf{len}(W))).$$

PROOF. Let W be a PLTL formula. To transform it into SNF we apply the τ_0 transformation, i.e.,

$$\tau_0[W] = \tau_1[\,\square(x \Rightarrow W)] \wedge \square(\textbf{start} \Rightarrow x).$$

From Lemma 7.1.1 we know the maximum number of PLTL-clauses from $\tau_1[\,\square(x \Rightarrow W)]$ is $11 \times \mathsf{len}(W)$; hence, the maximum number for the translation of W is $1 + (11 \times \mathsf{len}(W))$. □

7.1.2 *Number of New Proposition Symbols Generated*

LEMMA 7.1.2. *For any proposition symbol x and PLTL formula W, the maximum number of new proposition symbols generated from the translation of $\tau_1[\,\square(x \Rightarrow W)]$, denoted by $\mathsf{props}(\tau_1[\,\square(x \Rightarrow W)])$, will be at most* $4 \times \mathsf{len}(W)$*, i.e.,*

$$\mathsf{props}(\tau_1[\,\square(x \Rightarrow W)]) \leqslant (4 \times \mathsf{len}(W)).$$

PROOF. The proof is by induction on the length of W. The base case is where W has length 1, i.e., it has the form $\Diamond l$, $l_1 \vee \ldots \vee l_n$, **true**, **false**, $\bigcirc(l_1 \vee \ldots \vee l_n)$. Each of these produces no new proposition symbols, so as $0 \leqslant (4 \times 1)$ we are done. For the inductive hypothesis we assume that the theorem holds for formulae of length n and examine each case for length $n+1$. Again we examine some of the cases involved.

$$\begin{aligned}
\mathsf{props}(\tau_1[\Box(x \Rightarrow \neg(A \mathcal{W} B))]) &= 4 + \mathsf{props}(\tau_1[\Box(y \Rightarrow \neg A)]) + \mathsf{props}(\tau_1[\Box(z \Rightarrow \neg B)]) \\
&\leqslant (4 + (4 \times \mathsf{len}(\neg A)) + (4 \times \mathsf{len}(\neg B))) \\
&= 4(1 + \mathsf{len}(\neg A) + \mathsf{len}(\neg B)) \\
&= 4 \times \mathsf{len}(\neg(A \mathcal{W} B))
\end{aligned}$$

$$\begin{aligned}
\mathsf{props}(\tau_1[\Box(x \Rightarrow (A \mathcal{W} B))]) &= 3 + \mathsf{props}(\tau_1[\Box(y \Rightarrow A)]) + \mathsf{props}(\tau_1[\Box(z \Rightarrow B)]) \\
&\leqslant (3 + (4 \times \mathsf{len}(A)) + (4 \times \mathsf{len}(B))) \\
&\leqslant 4(1 + \mathsf{len}(A) + \mathsf{len}(B)) \\
&= 4 \times \mathsf{len}(A \mathcal{W} B)
\end{aligned}$$

$$\begin{aligned}
\mathsf{props}(\tau_1[\Box(x \Rightarrow (\Box A))]) &= 2 + \mathsf{props}(\tau_1[\Box(y \Rightarrow A)]) \\
&\leqslant (2 + (4 \times \mathsf{len}(A))) \\
&\leqslant 4(1 + \mathsf{len}(A)) \\
&= 4 \times \mathsf{len}(\Box A)
\end{aligned}$$

$$\begin{aligned}
\mathsf{props}(\tau_1[\Box(x \Rightarrow (\neg \Box A))]) &= 1 + \mathsf{props}(\tau_1[\Box(y \Rightarrow \neg A)]) \\
&\leqslant (1 + (4 \times \mathsf{len}(\neg A))) \\
&\leqslant 4(1 + \mathsf{len}(A)) \\
&= 4 \times \mathsf{len}(\neg \Box A)
\end{aligned}$$

The cases for the other operators are similar. □

THEOREM 7.1.2. *For any PLTL formula W, the maximum number of new proposition symbols, N, generated from the translation into SNF will be at most* $1 + (4 \times \mathsf{len}(W))$*, i.e.,*

$$N \leqslant 1 + (4 \times \mathsf{len}(W)).$$

PROOF. Let W be a PLTL formula. To transform it into SNF we apply the τ_0 transformation, i.e.,

$$\tau_0[W] = \tau_1[\Box(x \Rightarrow W)] \wedge \Box(\textbf{start} \Rightarrow x).$$

From Lemma 7.1.2 we know the maximum number of new proposition symbols from $\tau_1[\Box(x \Rightarrow W)]$ is $4 \times \mathsf{len}(W)$. Hence the maximum number for the translation of W is $1 + (4 \times \mathsf{len}(W))$. □

7.2 Step Resolution

Both forms of step resolution are essentially equivalent to classical resolution, for example the derivation of $\bigcirc$**false** on the right-hand side of a step PLTL-clause is essentially a classical resolution proof on the clauses of the right-hand side of (a subset of) the step PLTL-clauses. The complexity of this phase of the method is equivalent to the complexity of carrying out several classical resolution proofs on (simple translations of) the SNF PLTL-clauses. Indeed, one approach to the practical mechanization of step resolution has been to translate the SNF PLTL-clauses in to a form suitable for a classical resolution theorem prover [Dixon 2000].

7.3 Temporal Resolution

In order to consider the complexity of the temporal resolution phase, we describe a (naive) algorithm to find PLTL-clauses with which to apply the temporal resolution operation.

7.3.1 *A Naive Algorithm for Loop Detection.* Given a set of m step PLTL-clauses, R, and an eventuality $\Diamond l$ from the right-hand side of a sometime PLTL-clause, we carry out the following.

(1) Construct the set of merged-SNF PLTL-clauses for the SNF PLTL-clauses in R, i.e., apply the merged-SNF operation in Section 3.1 to each set of PLTL-clauses in each member of the powerset of R obtaining the set of (SNF_m) PLTL-clauses, R^*.
(2) Delete any PLTL-clause $X_i \Rightarrow \bigcirc Y_i$ in R^* such that it is not the case that $Y_i \Rightarrow \neg l$.
(3) Delete any SNF_m PLTL-clauses $X_i \Rightarrow \bigcirc Y_i$ in R^* such that it is not the case that

$$Y_i \Rightarrow \bigvee_j X_j$$

where X_j is the left-hand side of PLTL-clause j in R^*.
(4) Repeat 3 until no more SNF_m PLTL-clauses can be deleted.

7.3.2 *Correctness of Naive Algorithm*

THEOREM 7.3.1. *Given a set of step PLTL-clauses R and an eventuality $\Diamond l$, there is a loop in $\neg l$ within R if, and only if, the above algorithm outputs a nonempty set of PLTL-clauses L'.*

PROOF. Consider a loop L in $\neg l$ formed from the set of PLTL-clauses R. Let the disjunction of the left-hand side of the SNF_m PLTL-clauses in L be X. As L is a loop the right-hand side of each SNF_m PLTL-clause in L implies both $\neg l$ and X. Assume there are n SNF_m PLTL-clauses in L. Each SNF_m PLTL-clause (or an equivalent SNF_m PLTL-clause) in L must be in the set R^* before deletions, as L has been made by combining PLTL-clauses in R.

We next consider the deletion of any SNF_m PLTL-clause in L from R^*. Step 2 of the algorithm will not remove any of the SNF_m PLTL-clauses in L from R^*, as it removes SNF_m PLTL-clauses whose right-hand side do not imply $\neg l$, but, by assumption, each SNF_m PLTL-clause in L has a right-hand side that implies $\neg l$. Assume we are about to remove an SNF_m PLTL-clause $P \Rightarrow \bigcirc Q$, contained in L from the set R^* using step 3 of the algorithm. Let Y be the disjunction of the left-hand sides of the SNF_m PLTL-clauses remaining undeleted in R^* that are not in L. Thus $P \Rightarrow \bigcirc Q$ is being deleted, as it is not the case that $Q \Rightarrow X \vee Y$. However we know that $Q \Rightarrow X$, as L is a loop, so $Q \Rightarrow X \vee Y$ must also hold giving a contradiction. Hence none of the SNF_m PLTL-clauses in L can be deleted from R^*, so the algorithm must return a set of SNF_m PLTL-clauses containing L.

Consider any set of SNF_m PLTL-clauses L' output by the algorithm. Each SNF_m PLTL-clause has been made by combining PLTL-clauses in R. Each right-hand side implies $\neg l$; otherwise it would have been deleted by step 2 of the algorithm. Each right-hand side implies the disjunction of the left-hand side of the set of SNF_m PLTL-clauses; otherwise it would have been deleted by step 3 of the algorithm. The set of SNF_m PLTL-clauses satisfies the side conditions for being a loop; hence this loop can be constructed by combining the relevant PLTL-clauses in R. □

7.3.3 *Complexity of the Naive Algorithm.* Next we consider the complexity of detecting a set of PLTL-clauses in the way outlined above. We assume a set of m step PLTL-clauses containing n proposition symbols. The cost of combining the set of PLTL-clauses R is 2^m. To check that the right-hand side of each PLTL-clause implies $\neg l$ we must check a truth table with 2^{n-1} lines. Thus for 2^m PLTL-clauses we must check in total $2^{n-1} \times 2^m = 2^{m+n-1}$ lines. For step 3 the worst case is if one PLTL-clause is deleted from the set during each cycle of deletions until all the PLTL-clauses are deleted. We must check that each PLTL-clause implies the disjunction of the remaining left-hand sides, i.e., for each right-hand side checked we must consider a truth table with 2^n lines. Thus, to check each PLTL-clause once has complexity of order $2^m \times 2^n = 2^{m+n}$, and to carry out 2^m rounds of checking we require 2^{2m+n}. Hence, the complexity of applying the resolution rule once is of order 2^{2m+n}.

This gives the worst-case bound for any loop checking algorithm. Refined approaches to finding loops only improve the average performance [Dixon 1996; 1998].

7.4 Complexity of the Temporal Resolution Method

We consider the complexity of the whole method by looking at the behavior graph used in the proof for completeness of temporal resolution. Assume we have n proposition symbols (including those added for augmentation see Section 6.1) and r eventualities. Deletions in the behavior graph represent either a series of step resolution inferences or a temporal resolution inference.

The deletion of a terminal node (and edges into it) corresponds to construction of a PLTL-clause $A \Rightarrow \bigcirc\mathbf{false}$, i.e., complexity of a classical resolution proof. The deletion of a terminal subgraph (one or more nodes) with p an unsatisfied eventuality corresponds to temporal resolution (with complexity 2^{2m+n} for m PLTL-clauses). The worst case is if we have to delete each node separately, i.e., the worst-case complexity is the number of nodes multiplied by the maximum of the complexity of a temporal resolution step and the complexity of classical resolution, plus the complexity of classical resolution (i.e., resolution between initial PLTL-clauses to finish the proof). Although the number of PLTL-clauses we have may change at each step, the worst-case number of PLTL-clauses is 2^{2n}, i.e., 2^n possible left-hand sides and 2^n possible right-hand sides. Recall that nodes in the behavior graph are pairs (V, E) where V is a valuation of the proposition symbols in the PLTL-clause set and E is a subset of the eventualities.

Thus the number of nodes in the behavior graph is $2^n \times 2^r$ (where $r \leqslant 2n$), i.e., at worst 2^{3n}. Thus complexity is of the order $2^{3n} \times 2^{2^{2n+1}+n} = 2^{2^{2n+1}+4n}$.

We note that the complexity of satisfiability for PLTL is PSPACE complete [Sistla and Clarke 1985]. The complexity for the resolution methods in Abadi and Manna [1985], Cavalli and Fariñas del Cerro [1984], and Venkatesh [1986] and the tableau method in Gough [1984] is not discussed in the relevant papers, but the complexity for Wolper's tableau [Wolper 1983] is given as exponential in the length of the initial formula.

8. RELATED WORK

We consider three resolution-based approaches for PLTL (or similar languages) and then several implemented methods for PLTL.

8.1 Resolution Methods for PLTL

8.1.1 *Venkatesh.* Venkatesh [1986] describes a clausal resolution method for PLTL for future-time operators including $\mathcal{U}$. First, formulae are translated into a normal form containing a restricted nesting of temporal operators. The normal form is

$$\bigwedge_{i=1}^{n} c_i \wedge \bigwedge_{j=1}^{m} \Box c'_j,$$

where each c_i and c'_j (known as *clauses*) is a disjunction of formulae of the form $\bigcirc^k l$, $\bigcirc^k \Box l$, $\bigcirc^k \Diamond l$, or $\bigcirc^k (l' \,\mathcal{U}\, l)$ (known as *principal terms*) for l and l' literals, $k \geqslant 0$ and $\bigcirc^k$ denoting a series of k $\bigcirc$-operators.

The clauses in the normal form therefore either apply to the first moment in time or to every moment in time (those enclosed in a $\Box$-operator). Resolution proofs are displayed in columns separating the clauses that hold in each state. To determine unsatisfiability, the principal terms (except $\bigcirc^k l$) in each clause are *unwound* to split them into present and future parts. For example the clause $F \vee \Diamond l$ is replaced by $F \vee l \vee \bigcirc\Diamond l$ and similarly for $\Box$ and $\mathcal{U}$. Next, classical-style resolution is carried out between complementary literals relating to the present parts of the clauses in each column or state. Then, any clauses in a state that contain only principal terms with one or more next operators are transferred to the next state, and the number of next operators attached to each term is reduced by one. This process is shown to be complete for clauses that contain no eventualities. Formulae that contain eventualities that are delayed indefinitely due to unwinding are eliminated, and this process is shown to be complete.

This system makes use of a normal form which at the top level is similar to ours, i.e., there are clauses that relate to to first moment in time (as do our initial PLTL clauses) and to every moment in time (as our step and eventuality PLTL-clauses). Venkatesh uses renaming to remove any nesting of operators, as we do here, to rewrite into the normal form. Thus, as with our system, new propositions are introduced into the normal form. The main difference is that Ventatesh does not remove the temporal operators $\Box$ and $\mathcal{U}$.

Our initial step resolution can be compared with the resolution of complementary literals in the first state, and step resolution is comparable to resolution of complementary literals in other states.

The main difference is the treatment of eventualities. The system described in this article looks for sets of formulae with which to apply the temporal resolution rule to generate additional constraints that must be fulfilled. Venkatesh looks for persistent unfulfilled eventualities. In many ways the Venkatesh system behaves like a temporal tableau system [Wolper 1983; Gough 1984], but classical resolution inferences are applied within states. Repeated states containing persistent eventualities are identified and the unresolved eventualities eliminated, similar to the check for unsatisfied eventualities in temporal tableau.

The overall approach to the system described in this article generates constraints until we obtain a contradiction in the initial state **start** $\Rightarrow$ **false**. Venkatesh's approach reasons forward carrying clauses that are disjunctions of terms involving one or more next operator to the next moment, having deleted a next operator. This forward reasoning approach seems similar to the work on the executable temporal logics METATEM [Barringer et al. 1996].

8.1.2 *Cavalli and Fariñas del Cerro.* A clausal resolution method for PLTL is outlined in Cavalli and Fariñas del Cerro [1984]. The temporal operators defined in the logic include $\bigcirc$, $\square$, and $\diamondsuit$ but do not include $\mathcal{U}$. The method described rewrites formulae to a complicated normal form and then applies a series of temporal resolution rules.

A formula, F, is said to be in Conjunctive Normal Form (CNF), if it is of the form

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where each C_j is called a *clause* and is of the following form.

$$\begin{aligned} C_j = {} & L_1 \vee L_2 \vee \ldots \vee L_n \vee \square D_1 \vee \square D_2 \vee \ldots \vee \square D_p \\ & \vee \diamondsuit A_1 \vee \diamondsuit A_2 \vee \ldots \vee \diamondsuit A_q \end{aligned}$$

Here each L_i is a literal preceded by a string of zero or more $\bigcirc$-operators; each D_i is a disjunction of the same general form as the clauses; and each A_i is a conjunction where each conjunct possesses the same general form as the clauses. The resolution operations are split into three types, classical operations, temporal operations, and transformation operations. The former apply the classical resolution rule and classical logic rewrites, the latter two are required for manipulations of temporal operators. For example a temporal operation is of the form that $\square x$ and $\diamondsuit y$ can be resolved if x and y are resolvable, and the resolvent will be the resolvent of x and y with a $\diamondsuit$-operator in front.

Formulae are refuted by translation to normal form and repeated application of the inference rules. Resolution only takes place between clauses in the context of certain operators outlined in the resolution rules.

The method is only similar to our method as it uses translation to a clause form, although the normal form is much more complicated. The rules required to rewrite formulae into the normal form depend on temporal theorems and classical methods. Renaming and the introduction of new proposition symbols are not required.

The temporal and transformation operations take account of the temporal operators to make sure that contradictory formulae occur at the same moment in time. In our system this is done by translating to the normal form followed by initial and step resolution. Several operations are defined to deal with eventualities; for example the temporal operation given above, whereas we have just the one temporal resolution rule. The following complex transformation operation can be applied to an eventuality and is required to deal with the induction between $\Box$ and $\bigcirc$

$$\begin{aligned}\Sigma_3(\Diamond E, F) \;=\; & E \vee \bigcirc E \vee \ldots \bigcirc^{n-1} E \vee \\ & \Sigma_i(\Diamond(\neg E \wedge \bigcirc\neg E \wedge \ldots \wedge \bigcirc^{n-1}\neg E \wedge \bigcirc^{n} E), F)\end{aligned}$$

$$\text{And if } E \vee \bigcirc E \vee \ldots \bigcirc^{n-1} E \text{ or } (\Diamond(\neg E \wedge \bigcirc\neg E \wedge \ldots \wedge \bigcirc^{n-1}\neg E \wedge \bigcirc^{n} E), F)$$
$$\text{is resolvable then } (\Diamond E, F) \text{ is resolvable}$$

where Σ_i denotes the further application of a classical, temporal, or transformation operation and where $\bigcirc^{n-1}$ denotes a string of $n-1$ next operators. The method is only described for a subset of the operators that we use, i.e., a less expressive logic. Further, the completeness proof is only given for the $\Box$, $\Diamond$, and $\bigcirc$ operators. An implementation of the method has been developed; however it is not clear when to apply each operation to lead toward a proof.

8.1.3 *Abadi.* Nonclausal temporal resolution systems are developed for propositional [Abadi and Manna 1985] and then first-order temporal logics [Abadi and Manna 1990] that are discrete and linear and have finite past and infinite future. The systems are developed first for fragments of the logic including the temporal operators $\bigcirc$, $\Box$, and $\Diamond$ and then extended for $\bigcirc$, $\Box$, $\Diamond$, $\mathcal{W}$[2], and $\mathcal{P}$. The binary operator $\mathcal{P}$ is known as *precedes* where $u\mathcal{P}v = \neg((\neg u)\,\mathcal{W}\,v)$.

Because the system is nonclausal many simplification and inference rules need to be defined. The resolution rule is of the form

$$A\langle u, \ldots, u\rangle, B\langle u, \ldots, u\rangle \longrightarrow A\langle \mathbf{true}\rangle \vee B\langle \mathbf{false}\rangle$$

where $A\langle u, \ldots, u\rangle$ denotes that u occurs one or more times in A. Here occurrences of u in A and B are replaced with **true** and **false** respectively. To ensure the rule is sound each u that is replaced must be in the scope of the same number of $\bigcirc$-operators, and must not be in the scope of any other modal operator in A or B, i.e., they must apply to the same moment in time. Other rules such as distribution and modality rules allow the format of the expression to be changed, for example the $\Box$-modality rule allows any formula $\Box u$ to be rewritten as $u \wedge \bigcirc\Box u$.

The induction rule deals with the interaction between $\bigcirc$ and $\Box$ and is of the form

$$w, \Diamond u \longrightarrow \Diamond(\neg u \wedge \bigcirc(u \wedge \neg w)) \text{ if } \vdash \neg(w \wedge u).$$

[2]Abadi denotes $\mathcal{W}$, *unless* (or *weak until*), as $\mathcal{U}$.

Informally this means that if w and u cannot both hold at the same time and if w and $\Diamond u$ hold now then there must be a moment in time (now or) in the future when u does not hold and at the next moment in time u holds and w does not. Both systems are shown complete. A proof editor has been developed for the propositional system with the $\bigcirc$, $\Box$, and $\Diamond$ operators.

As there is no translation to a normal form many rules need to be specified to allow for every different combination of operators. The resolution rule only allows resolution of formulae within the same number of next operators and can perhaps be compared with our step resolution rule except, due to our uniform normal form, our step resolution rule is much easier to apply. Finally the rule that corresponds with our temporal resolution rule is the induction rule. This rule can only be applied if a complex side condition is checked.

Although a proof editor has been developed for the restricted propositional system it seems unlikely that Abadi's system lends itself to a fully automatic implementation. This is because of the large number of rules that may be applied. Further, the induction rule requires a proof as a side condition to its usage which will make automatic proofs difficult. The implementation of the induction rule is not discussed. The temporal resolution rule we have described in this article is also complex; however we have considered its implementation in Dixon[1996; 1998] and developed a fully automatic prototype theorem prover based on this.

8.2 Implementations

We now briefly mention several implementations available for linear time temporal logics. The Logics Workbench [Jaeger et al. 2000], a theorem-proving system for various modal logics available over the Web, has a module for dealing with logics such as PLTL [Schwendimann 1998]. The implementation of this module is based on tableau with an analysis of strongly connected components to deal with eventualities. A tableau-based theorem prover for PLTL, called DP, has also been developed [Gough 1984]. Finally, the STeP system [Bjorner et al. 1995], based on ideas presented in Manna and Pnueli [1992; 1995], and providing both model checking and deductive methods for PLTL-like logics, has been used in order to assist the verification of concurrent and reactive systems based on temporal specifications.

9. SUMMARY

In this article we have described, in detail, a clausal resolution method for propositional linear temporal logic (PLTL), and have considered its soundness, completeness, termination, and complexity. The method is based on the translation to a concise normal form, and the application of both step resolution (essentially classical resolution) and temporal resolution operations. Since temporal logics such as PLTL are useful for describing reactive systems, the resolution method has a variety of applications in verifying properties of complex systems. We believe that this resolution system can form the basis of an efficient temporal theorem proving system that can outperform other systems developed for such logics. However, there is still work to be done in order to realize this.

9.1 Future Work

A prototype version of this system has been implemented in Prolog, primarily to test the *loop search* algorithms required for the temporal resolution rule [Dixon 1996]. A more refined C++ version, known as CLATTER, is currently under development. Both these systems utilize the fact that step resolution is very similar to classical resolution and consequently use a resolution theorem prover for classical logic, namely OTTER, to implement this part of the system [Dixon 2000].

The normal form used in this article (SNF) has been extended to apply to other logics such as branching-time temporal logics [Bolotov and Fisher 1997] and multimodal logics involving both a temporal and a modal dimension [Dixon et al. 1998]. Much of our current work involves extending the clausal resolution approach to a wider variety of temporal and modal logics. In each of these logics, not only must a version of SNF be defined, but specialized resolution operations must be developed dependent on the properties of the logic in question.

Just as strategies for classical resolution have been successful in improving efficiency, we aim to develop similar strategies for temporal resolution. In particular, we are interested in the most efficient way to apply the resolution operations in order to reduce the number of resolution inferences that are made that do not contribute toward finding a proof. The work described in Dixon and Fisher [1998] outlines preliminary steps in the definition of a temporal set of support. The set of support strategy for classical resolution restricts the number of resolution inferences that can be made. Inferences can only be made where one of the clauses being resolved is from a subset of the full clause set known as the set of support. Thus if we are asked to prove that B is a logical consequence of A (or $A \vdash B$) in resolution we would try show that $A \wedge \neg B$ is unsatisfiable. To use the set of support strategy the clauses derived from A are separated from those derived from $\neg B$, the latter being put into the set of support. Thus resolution inferences between two clauses derived from A are avoided. We are also developing and applying a modified resolution operation that can be used in a more flexible way, and can be used with strategies such as set of support. Initial results can be found in Fisher and Dixon [2000].

Finally as efficient subsets of classical logic such as Horn clauses, have been investigated we hope to define restrictions on the normal form that allow temporal resolution to be carried out more efficiently and investigate the classes of problem these subsets correspond to.

APPENDIX

A. PROOFS FOR TRANSFORMATION INTO SNF

Here we present several lemmas required for the proof of Theorem 3.3.1 showing the translation to SNF is satisfiability preserving. The transformations τ_0 and τ_1 are defined in Section 3.2, and new proposition symbols generated are shown in bold.

Firstly, we show that

$$\models \tau_0[W] \Rightarrow W,$$

i.e., any model for the transformed formula is a model for the original. However before we show this we first prove a lemma.

LEMMA A.1. *For all PLTL formulae W*

$$\models \tau_1[\square(x \Rightarrow W)] \Rightarrow \square(x \Rightarrow W)$$

where x is a proposition symbol.

PROOF. The proof is carried out by induction on the structure of W. For the base cases we have the following.

$$\begin{array}{lrcl}
1. & \tau_1[\square(x \Rightarrow \Diamond l)] & = & \square(x \Rightarrow \Diamond l) \\
2. & \tau_1[\square(x \Rightarrow l_1 \vee \ldots \vee l_n)] & = & \square(\mathbf{start} \Rightarrow \neg x \vee l_1 \vee \ldots \vee l_n) \wedge \\
 & & & \square(\mathbf{true} \Rightarrow \bigcirc(\neg x \vee l_1 \vee \ldots \vee l_n)) \\
 & & \Rightarrow & \square(x \Rightarrow (l_1 \vee \ldots \vee l_n)) \\
3. & \tau_1[\square(x \Rightarrow \mathbf{true})] & = & \square(\mathbf{start} \Rightarrow \mathbf{true}) \wedge \\
 & & & \square(\mathbf{true} \Rightarrow \bigcirc\mathbf{true}) \\
 & & \Rightarrow & \square(x \Rightarrow \mathbf{true}) \\
4. & \tau_1[\square(x \Rightarrow \mathbf{false})] & = & \square(\mathbf{start} \Rightarrow \neg x) \wedge \\
 & & & \square(\mathbf{true} \Rightarrow \bigcirc\neg x) \\
 & & \Rightarrow & \square(x \Rightarrow \mathbf{false}) \\
5. & \tau_1[\square(x \Rightarrow \bigcirc(l_1 \vee \ldots \vee l_n))] & = & \square(x \Rightarrow \bigcirc(l_1 \vee \ldots \vee l_n))
\end{array}$$

Now, we assume that the lemma holds for A, B, $\neg A$, and $\neg B$, e.g., $\tau_1[\square(x \Rightarrow A)] \Rightarrow \square(x \Rightarrow A)$, and show it holds for all combinations of operators or negated operators, e.g., $A \wedge B$, $\neg(A \wedge B)$, $\square A$, $\neg\square A$. We consider the cases for $\square A$, $\neg\square A$, $A \mathcal{W} B$, and $\neg(A \mathcal{W} B)$ and note that proofs for the other operators are similar (where $\mathbf{v}$, $\mathbf{w}$, $\mathbf{y}$, and $\mathbf{z}$ are new proposition symbols):

$$\begin{array}{rcl}
\tau_1[\square(x \Rightarrow \square A)] & = & \tau_1[\square(x \Rightarrow \square\mathbf{y})] \wedge \tau_1[\square(\mathbf{y} \Rightarrow A)] \\
 & = & \tau_1[\square(x \Rightarrow \mathbf{y})] \wedge \tau_1[\square(x \Rightarrow \mathbf{z})] \wedge \square(\mathbf{z} \Rightarrow \bigcirc\mathbf{y}) \wedge \\
 & & \quad \square(\mathbf{z} \Rightarrow \bigcirc\mathbf{z}) \wedge \tau_1[\square(\mathbf{y} \Rightarrow A)] \\
 & \Rightarrow & \square(\mathbf{start} \Rightarrow \neg x \vee \mathbf{y}) \wedge \square(\mathbf{true} \Rightarrow \bigcirc(\neg x \vee \mathbf{y})) \wedge \\
 & & \square(\mathbf{start} \Rightarrow \neg x \vee \mathbf{z}) \wedge \square(\mathbf{true} \Rightarrow \bigcirc(\neg x \vee \mathbf{z})) \wedge \\
 & & \square(\mathbf{z} \Rightarrow \bigcirc\mathbf{y}) \wedge \square(\mathbf{z} \Rightarrow \bigcirc\mathbf{z}) \wedge \square(\mathbf{y} \Rightarrow A) \\
 & \Rightarrow & \square(x \Rightarrow \square A)
\end{array}$$

where $\tau_1[\square(\mathbf{y} \Rightarrow A)] \Rightarrow \square(\mathbf{y} \Rightarrow A)$ from the induction hypothesis:

$$\begin{array}{rcl}
\tau_1[\square(x \Rightarrow \neg\square A)] & = & \square(x \Rightarrow \Diamond\mathbf{y}) \wedge \tau_1[\square(\mathbf{y} \Rightarrow \neg A)] \\
 & \Rightarrow & \square(x \Rightarrow \Diamond\mathbf{y}) \wedge \square(\mathbf{y} \Rightarrow \neg A) \\
 & \Rightarrow & \square(x \Rightarrow \Diamond\neg A) \\
 & \Rightarrow & \square(x \Rightarrow \neg\square A)
\end{array}$$

where $\tau_1[\square(\mathbf{y} \Rightarrow \neg A)] \Rightarrow \square(\mathbf{y} \Rightarrow \neg A)$ from the induction hypothesis:

$$\begin{array}{rcl}
\tau_1[\square(x \Rightarrow (A \mathcal{W} B))] & = & \tau_1[\square(x \Rightarrow \mathbf{y} \mathcal{W} \mathbf{z})] \wedge \tau_1[\square(\mathbf{y} \Rightarrow A)] \wedge \tau_1[\square(\mathbf{z} \Rightarrow B)] \\
 & = & \tau_1[\square(x \Rightarrow \mathbf{y} \vee \mathbf{z})] \wedge \tau_1[\square(x \Rightarrow \mathbf{w} \vee \mathbf{z})] \wedge \\
 & & \quad \square(\mathbf{w} \Rightarrow \bigcirc(\mathbf{y} \vee \mathbf{z})) \wedge \square(\mathbf{w} \Rightarrow \bigcirc(\mathbf{w} \vee \mathbf{z})) \wedge \\
 & & \quad \tau_1[\square(\mathbf{y} \Rightarrow A)] \wedge \tau_1[\square(\mathbf{z} \Rightarrow B)] \\
 & \Rightarrow & \square(\mathbf{start} \Rightarrow \neg x \vee \mathbf{y} \vee \mathbf{z}) \wedge \square(\mathbf{true} \Rightarrow \bigcirc(\neg x \vee \mathbf{y} \vee \mathbf{z})) \wedge \\
 & & \square(\mathbf{start} \Rightarrow \neg x \vee \mathbf{w} \vee \mathbf{z}) \wedge \square(\mathbf{true} \Rightarrow \bigcirc(\neg x \vee \mathbf{w} \vee \mathbf{z})) \\
 & & \wedge \square(\mathbf{w} \Rightarrow \bigcirc(\mathbf{y} \vee \mathbf{z})) \wedge \square(\mathbf{w} \Rightarrow \bigcirc(\mathbf{w} \vee \mathbf{z})) \wedge \\
 & & \square(\mathbf{y} \Rightarrow A) \wedge \square(\mathbf{z} \Rightarrow B) \\
 & \Rightarrow & \square(x \Rightarrow (A \mathcal{W} B))
\end{array}$$

$$
\begin{aligned}
\tau_1[\square(x \Rightarrow \neg(A \mathcal{W} B))] &= \tau_1[\square(x \Rightarrow (\mathbf{y}\,\mathcal{U}\,\mathbf{v})] \wedge \tau_1[\square(\mathbf{v} \Rightarrow (\mathbf{y} \wedge \mathbf{z}))]\wedge \\
&\quad \tau_1[\square(\mathbf{y} \Rightarrow \neg B)] \wedge \tau_1[\square(\mathbf{z} \Rightarrow \neg A)] \\
&= \tau_1[\square(x \Rightarrow \mathbf{v} \vee \mathbf{y})] \wedge \tau_1[\square(x \Rightarrow \mathbf{v} \vee \mathbf{w})]\wedge \\
&\quad \square(x \Rightarrow \Diamond \mathbf{v}) \wedge \square(\mathbf{w} \Rightarrow \bigcirc(\mathbf{v} \vee \mathbf{y}))\wedge \\
&\quad \square(\mathbf{w} \Rightarrow \bigcirc(\mathbf{v} \vee \mathbf{w})) \wedge \tau_1[\square(\mathbf{v} \Rightarrow (\mathbf{y} \wedge \mathbf{z}))]\wedge \\
&\quad \tau_1[\square(\mathbf{y} \Rightarrow \neg B)] \wedge \tau_1[\square(\mathbf{z} \Rightarrow (\neg A))] \\
&\Rightarrow \square(\mathbf{start} \Rightarrow \neg x \vee \mathbf{v} \vee \mathbf{y}) \wedge \square(\mathbf{true} \Rightarrow \bigcirc(\neg x \vee \mathbf{v} \vee \mathbf{y}))\wedge \\
&\quad \square(\mathbf{start} \Rightarrow \neg x \vee \mathbf{v} \vee \mathbf{w}) \wedge \square(\mathbf{true} \Rightarrow \bigcirc(\neg x \vee \mathbf{v} \vee \mathbf{w})) \\
&\quad \wedge \square(x \Rightarrow \Diamond \mathbf{v})\wedge \\
&\quad \square(\mathbf{w} \Rightarrow \bigcirc(\mathbf{v} \vee \mathbf{y})) \wedge \square(\mathbf{w} \Rightarrow \bigcirc(\mathbf{v} \vee \mathbf{w}))\wedge \\
&\quad \square(\mathbf{start} \Rightarrow \neg \mathbf{v} \vee \mathbf{y}) \wedge \square(\mathbf{start} \Rightarrow \neg \mathbf{v} \vee \mathbf{z})\wedge \\
&\quad \square(\mathbf{true} \Rightarrow \bigcirc(\neg \mathbf{v} \vee \mathbf{y})) \wedge \square(\mathbf{true} \Rightarrow \bigcirc(\neg \mathbf{v} \vee \mathbf{z}))\wedge \\
&\quad \square(\mathbf{y} \Rightarrow \neg B) \wedge \square(\mathbf{z} \Rightarrow (\neg A)) \\
&\Rightarrow \square(x \Rightarrow ((\neg B)\,\mathcal{W}\,(\neg A \wedge \neg B))) \wedge \square(x \Rightarrow \Diamond(\neg A \wedge \neg B)) \\
&\Rightarrow \square(x \Rightarrow ((\neg B)\,\mathcal{U}\,(\neg A \wedge \neg B))) \\
&\Rightarrow \square(x \Rightarrow \neg(A \mathcal{W} B)) \qquad \square
\end{aligned}
$$

LEMMA A.2. *For all PLTL formulae W*

$$\models \tau_0[W] \Rightarrow W.$$

PROOF. For any PLTL formula W, the first step in the transformation is to anchor W to the first moment in time, i.e., $\tau_0[W] \longrightarrow \square(\mathbf{start} \Rightarrow x) \wedge \tau_1[\square(x \Rightarrow W)]$. From Lemma A.1 we have shown that $\tau_1[\square(x \Rightarrow W)] \Rightarrow \square(x \Rightarrow W)$. Thus, as x holds at the first moment in time and the transformation implies that $(x \Rightarrow W)$ holds at every moment in time, then W also holds now. $\square$

Next we show that for any satisfiable formula its translation is also satisfiable, i.e., for any PLTL formula W, if W is satisfiable then $\tau_0[W]$ is satisfiable. This is established by showing that given a model for a formula at some stage in the transformation process for each step carried out in the transformation we can find a model for the transformed formula.

Definition A.1 (Pre-PLTL-Clause Form). A PLTL formula is said to be in *pre-PLTL-clause from* if, and only if, it has the structure

$$(x_i \Rightarrow W_i)$$

where x_i is a proposition symbol (or **start**) and W_i is a PLTL formula.

LEMMA A.3. *Let σ be a model such that*

$$(\sigma, 0) \models \left[\bigwedge_h \square R_h\right] \wedge \square(x \Rightarrow W)$$

where each R_h is in pre-PLTL-clause form (i.e., an implication where the proposition symbol on the left-hand side of each implication may be different). Then, there exists a model σ' such that

$$(\sigma', 0) \models \left[\bigwedge_h \square R_h\right] \wedge \bigwedge_j \square S_j \wedge \bigwedge_k \square T_k$$

where R_h is in pre-PLTL-clause form, S_j is in pre-PLTL-clause form, and T_k is in PLTL-clause form resulting from one step of the τ_1 transformation, i.e.,

$$\tau_1[\,\square(x \Rightarrow W)] \longrightarrow \left[\bigwedge_j \tau_1[\,\square S_j]\right] \wedge \bigwedge_k \square T_k.$$

Proof. We examine the structure of W. There are three main types of transformation that can be applied: the removal of classical operators, the renaming of complex subformulae, and the rewriting of temporal operators applied to literals. We begin by considering the removal of classical operators.

First, assume W is a conjunction $A \wedge B$, i.e.,

$$(\sigma, 0) \models \left[\bigwedge_h \square R_h\right] \wedge \square(x \Rightarrow (A \wedge B)).$$

Applying the τ_1 translation we have

$$\tau_1[\,\square(x \Rightarrow (A \wedge B))] \longrightarrow \tau_1[\,\square(x \Rightarrow A)] \wedge \tau_1[\,\square(x \Rightarrow B)],$$

so we must show there is a model σ' such that

$$(\sigma', 0) \models \left[\bigwedge_h \square R_h\right] \wedge \square(x \Rightarrow A) \wedge \square(x \Rightarrow B).$$

Now, as $(\sigma, 0) \models \square(x \Rightarrow (A \wedge B))$ for all $i \in \mathbb{N}$, then if $(\sigma, i) \models x$ both $(\sigma, i) \models A$ and $(\sigma, i) \models B$. That is

$$(\sigma, 0) \models \left[\bigwedge_h \square R_h\right] \wedge \square(x \Rightarrow A) \wedge \square(x \Rightarrow B).$$

So, by setting σ' equal to σ we have such a model. The proofs are similar for the other classical logic operators.

Next, we consider renaming transformations and assume W is of the form $\square A$ where A is not a literal. Now, assume that there exists a σ such that

$$(\sigma, 0) \models \left[\bigwedge_h \square R_h\right] \wedge \square(x \Rightarrow \square A).$$

By applying the τ_1 transformation, we have

$$\tau_1[\,\square(x \Rightarrow \square A)] \longrightarrow \tau_1[\,\square(x \Rightarrow \square \mathbf{y})] \wedge \tau_1[\,\square(\mathbf{y} \Rightarrow A)]$$

where $\mathbf{y}$ is a new proposition symbol. Thus, we must show that there exists a model σ' such that

$$(\sigma', 0) \models \left[\bigwedge_h \square R_h\right] \wedge \square(x \Rightarrow \square \mathbf{y}) \wedge \square(\mathbf{y} \Rightarrow A).$$

First assume that x is never satisfied in σ. A model σ' identical to σ except it contains the variable $\mathbf{y}$ such that $\mathbf{y}$ is false everywhere will suffice. Otherwise let j be the first place that x is satisfied in σ. As $(\sigma, 0) \models \square(x \Rightarrow \square A)$ for all $i \geq j$ then $(\sigma, i) \models A$. Let σ' be the same as σ except it contains a new proposition symbol $\mathbf{y}$ that is satisfied in all $i \geq j$ and unsatisfied elsewhere, i.e., $0 \leq i < j$. Thus, as σ' is identical to σ, except for $\mathbf{y}$, we have $(\sigma', i) \models A$ for all $i \geq j$, and from the definition of σ' we have for all $i \geq j$, $(\sigma', i) \models \mathbf{y}$ and, for all $i < j$, $(\sigma', i) \models \neg\mathbf{y}$. Thus, from the semantics of PLTL, $(\sigma', 0) \models \Box(\mathbf{y} \Rightarrow A)$. Now, as $(\sigma', i) \models \mathbf{y}$ for all $i \geq j$ then $(\sigma', j) \models \Box\mathbf{y}$ from the semantics of $\Box$. Also, as $(\sigma', j) \models x$ and by assumption j is the first place x is satisfied in σ and therefore σ', $(\sigma', 0) \models \Box(x \Rightarrow \Box\mathbf{y})$. Further

$$(\sigma', 0) \models \bigwedge_h \Box R_h$$

as

$$(\sigma, 0) \models \bigwedge_h \Box R_h$$

from our choice of σ'. Hence

$$(\sigma', 0) \models \left[\bigwedge_h \Box R_h\right] \wedge \Box(x \Rightarrow \Box\mathbf{y}) \wedge \Box(\mathbf{y} \Rightarrow A)$$

as desired. The proof of other renaming operations is similar.

Finally we consider the removal of unwanted temporal operators. Again, we let W be $\Box A$, but this time assume that A is a literal. Assume that there exists a σ such that

$$(\sigma, 0) \models \left[\bigwedge_h \Box R_h\right] \wedge \Box(x \Rightarrow \Box A).$$

By applying the τ_1 transformation we obtain

$$\begin{aligned}\tau_1[\Box(x \Rightarrow \Box A)] \longrightarrow\ & \tau_1[\Box(x \Rightarrow A)] \wedge \tau_1[\Box(x \Rightarrow \mathbf{y})] \wedge \Box(\mathbf{y} \Rightarrow \bigcirc A) \wedge \\ & \Box(\mathbf{y} \Rightarrow \bigcirc\mathbf{y})\end{aligned}$$

where $\mathbf{y}$ is a new proposition symbol. Thus, we must show that there exists a model σ' such that

$$(\sigma', 0) \models \left[\bigwedge_h \Box R_h\right] \wedge \Box(x \Rightarrow A) \wedge \Box(x \Rightarrow \mathbf{y}) \wedge \Box(\mathbf{y} \Rightarrow \bigcirc A) \wedge \Box(\mathbf{y} \Rightarrow \bigcirc\mathbf{y}).$$

First assume that x is never satisfied in σ. Similarly to the above, a model σ' identical to σ except containing the variable $\mathbf{y}$ such that $\mathbf{y}$ is false everywhere will suffice. Otherwise let j be the first place that x is satisfied in σ. Let σ' be the model that is identical to σ except it contains the variable $\mathbf{y}$ such that for all $i \geq j$, $(\sigma', i) \models \mathbf{y}$ and for all $0 \leq i < j$, $(\sigma', i) \models \neg\mathbf{y}$. Thus, as σ is the same as σ' except for the valuation of $\mathbf{y}$, and

$$(\sigma, 0) \models \bigwedge_h \Box R_h,$$

then we have

$$(\sigma', 0) \models \bigwedge_h \Box R_h.$$

We have assumed that $(\sigma, 0) \models \square(x \Rightarrow \square A)$, so for all $i \geq j$, $(\sigma, i) \models A$; hence for all $i \geq j$, $(\sigma', i) \models A$. Thus, as $(\sigma', j) \models x$, where j is the first place that x holds and for all $i \geq j$, $(\sigma', i) \models A$ we have $(\sigma', 0) \models \square(x \Rightarrow A)$. Now as j is the first place that x holds and $(\sigma', i) \models \mathbf{y}$ for all $i \geq j$ we have $(\sigma', 0) \models \square(x \Rightarrow \mathbf{y})$ and $(\sigma', 0) \models \square(\mathbf{y} \Rightarrow \bigcirc\mathbf{y})$. Also, as $i \geq j$, $(\sigma, i) \models A$ then, due to our choice of σ', for all $i \geq j$, $(\sigma', i) \models A$ and so $(\sigma', 0) \models \square(\mathbf{y} \Rightarrow \bigcirc A)$. Hence

$$(\sigma', 0) \models \left[\bigwedge_h \square R_h\right] \wedge \square(x \Rightarrow A) \wedge \square(x \Rightarrow \mathbf{y}) \wedge \square(\mathbf{y} \Rightarrow \bigcirc A) \wedge \square(\mathbf{y} \Rightarrow \bigcirc\mathbf{y})$$

as required. □

LEMMA A.4. *Given a model σ, and a PLTL formula W, such that $(\sigma, 0) \models W$, there exists a model σ' such that $(\sigma', 0) \models \tau_0[W]$.*

PROOF. Firstly note that if $(\sigma, 0) \models W$ then there is a model σ'' such that

$$(\sigma'', 0) \models (\mathbf{start} \Rightarrow \mathbf{y}) \wedge \square(\mathbf{y} \Rightarrow W).$$

The model σ'' is identical to σ except it includes the new proposition symbol $\mathbf{y}$ which is set to true where $i = 0$ and false everywhere else. Applying τ_0 to W, we obtain

$$(\mathbf{start} \Rightarrow \mathbf{y}) \wedge \tau_1[\square(\mathbf{y} \Rightarrow W)].$$

Now, from Lemma A.3, and given that $(\mathbf{start} \Rightarrow \mathbf{y}) \wedge \square(\mathbf{y} \Rightarrow W)$ has a model σ'' every application of the τ_1 transformation can be satisfied in some new model. Hence, if W has a model then there exists a model that satisfies $\tau_0[W]$. □

ACKNOWLEDGMENTS

We would like to thank Howard Barringer, Graham Gough, Alexander Bolotov, Ullrich Hustadt, and Anatoli Degtiarev for their helpful comments and suggestions about this work. The authors would also like to extend their gratitude to all the anonymous referees for their hard work. Their dedication and time spent reviewing this article is much appreciated and has led to a greatly improved article.

REFERENCES

ABADI, M. AND MANNA, Z. 1985. Nonclausal Temporal Deduction. *Lecture Notes in Computer Science 193*, 1–15.

ABADI, M. AND MANNA, Z. 1990. Nonclausal Deduction in First-Order Temporal Logic. *ACM Journal 37,* 2 (Apr.), 279–317.

ARTALE, A. AND FRANCONI, E. 1999. Introducing Temporal Description Logics. In *Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning (TIME-99)*, C. Dixon and M. Fisher, Eds. IEEE Computer Society Press, Orlando, Florida. ISBN 0-7695-0173-7.

BARRINGER, H., FISHER, M., GABBAY, D., OWENS, R., AND REYNOLDS, M., Eds. 1996. *The Imperative Future: Principles of Executable Temporal Logic.* Research Studies Press.

BJORNER, N., BROWNE, A., CHANG, E., COLÓN, M., KAPUR, A., MANNA, Z., SIPMA, H. B., AND URIBE, T. E. 1995. *STeP The Stanford Temporal Prover Educational Release Version 1.0 User's Manual*. Computer Science Department, Stanford University, California 94305.

BOLOTOV, A. AND FISHER, M. 1997. A Resolution Method for CTL Branching-Time Temporal Logic. In *Proceedings of TIME-97 the Fourth International Workshop on Temporal Representation and Reasoning*. IEEE Computer Society Press, Daytona Beach, Florida, 20–27.

BURGESS, J. P. AND GUREVICH, Y. 1985. The Decision Problem for Linear Temporal Logics. *Notre Dame Journal of Formal Logic 26,* 2 (Apr.), 115–128.

CAVALLI, A. AND FARIÑAS DEL CERRO, L. 1984. A Decision Method for Linear Temporal Logic. In *Proceedings of the 7th International Conference on Automated Deduction*, R. E. Shostak, Ed. Lecture Notes in Computer Science, vol. 170. Springer-Verlag, 113–127.

CHANG, C. L. AND LEE, R. C. T. 1973. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

DIXON, C. 1996. Search Strategies for Resolution in Temporal Logics. In *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE)*, M. A. McRobbie and J. K. Slaney, Eds. Lecture Notes in Artificial Intelligence, vol. 1104. Springer-Verlag, New Brunswick, New Jersey, 672–687.

DIXON, C. 1998. Temporal Resolution using a Breadth-First Search Algorithm. *Annals of Mathematics and Artificial Intelligence 22*, 87–115.

DIXON, C. 2000. Using Otter for Temporal Resolution. In *Advances in Temporal Logic*. Applied Logic Series, vol. 16. Kluwer, 149–166. Proceedings the Second International Conference on Temporal Logic (ICTL). ISBN 0-7923-6149-0.

DIXON, C. AND FISHER, M. 1998. The Set of Support Strategy in Temporal Resolution. In *Proceedings of TIME-98 the Fifth International Workshop on Temporal Representation and Reasoning*. IEEE Computer Society Press, Sanibel Island, Florida.

DIXON, C., FISHER, M., AND JOHNSON, R. 1995. Parallel Temporal Resolution. In *Proceedings of TIME-95 the Second International Workshop on Temporal Representation and Reasoning*. Melbourne Beach, Florida.

DIXON, C., FISHER, M., AND WOOLDRIDGE, M. 1998. Resolution for Temporal Logics of Knowledge. *Journal of Logic and Computation 8,* 3, 345–372.

EMERSON, E. A. 1990. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 996–1072.

EMERSON, E. A. AND SRINIVASAN, J. 1988. Branching Time Temporal Logic. *Lecture Notes in Computer Science 354*, 123–172.

FISHER, M. 1991. A Resolution Method for Temporal Logic. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufman, Sydney, Australia, 99–104.

FISHER, M. 1992. A Normal Form for First-Order Temporal Formulae. In *Proceedings of Eleventh International Conference on Automated Deduction (CADE)*. Lecture Notes in Computer Science, vol. 607. Springer-Verlag, Saratoga Springs, New York, 370–384.

FISHER, M. 1997. A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. *Journal of Logic and Computation 7,* 4 (Aug.), 429–456.

FISHER, M. AND DIXON, C. 2000. Guiding Clausal Temporal Resolution. In *Advances in Temporal Logic*. Applied Logic Series, vol. 16. Kluwer, 167–184. Proceedings the Second International Conference on Temporal Logic (ICTL). ISBN 0-7923-6149-0.

FISHER, M. AND NOËL, P. 1992. Transformation and Synthesis in METATEM—Part I: Propositional METATEM. Technical Report UMCS-92-2-1, Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, U.K. Feb.

GABBAY, D., PNUELI, A., SHELAH, S., AND STAVI, J. 1980. The Temporal Analysis of Fairness. In *Proceedings of the Seventh ACM Symposium on the Principles of Programming Languages*. Las Vegas, Nevada, 163–173.

GABBAY, D. M. 1987. Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In *Proceedings of Colloquium on Temporal Logic in Specification*, B. Banieqbal, H. Barringer, and A. Pnueli, Eds. Lecture Notes in Computer Science, vol. 398. Springer-Verlag, Altrincham, U.K., 402–450.

GOUGH, G. D. 1984. Decision Procedures for Temporal Logic. M.S. thesis, Department of Computer Science, University of Manchester. Also University of Manchester, Department of Computer Science, Technical Report UMCS-89-10-1.

HAREL, D. AND PNUELI, A. 1985. On the Development of Reactive Systems. Tech. Rep. CS85-02, Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel. Jan.

HOLZMANN, G. J. 1997. The Model Checker Spin. *IEEE Trans. on Software Engineering 23,* 5 (May), 279–295. Special Issue on Formal Methods in Software Practice.

HORROCKS, I. 1998. The FaCT System. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference (Tableaux'98)*. LNCS, vol. 1397. Springer-Verlag, 307–312.

HUSTADT, U. AND SCHMIDT, R. 1999. An Empirical Analysis of Modal Theorem Provers. *Journal of Applied Non-Classical Logics 9,* 4, 479–522.

JAEGER, G., BALSIGER, P., HEUERDING, A., SCHWENDIMANN, S., BIANCHI, M., GUGGISBERG, K., JANSSEN, G., HEINLE, W., ACHERMANN, F., BOROUMAND, A. D., BRAMBILLA, P., BUCHER, I., AND ZIMMERMANN, H. 2000. LWB–The Logics Workbench 1.1. http://www.lwb.unibe.ch/. University of Berne, Switzerland.

LAMPORT, L. 1983. Specifying Concurrent Program Modules. *ACM Transactions on Programming Languages and Systems 5,* 2 (Apr.), 190–222.

LICHTENSTEIN, O., PNUELI, A., AND ZUCK, L. 1985. The Glory of the Past. In *Logics of Programs* (*Proc. Conf. Brooklyn USA 1985*), R. Parikh, Ed. Lecture Notes in Computer Science, vol. 193. Springer-Verlag, Berlin, 196–218.

MANNA, Z. AND PNUELI, A. 1992. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York.

MANNA, Z. AND PNUELI, A. 1995. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York.

MCCUNE, W. 1994. OTTER *3.0 Reference Manual and Guide*. Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439-4801.

PLAISTED, D. A. AND GREENBAUM, S. A. 1986. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Computation 2,* 3 (Sept.), 293–304.

PNUELI, A. 1977. The Temporal Logic of Programs. In *Proceedings of the Eighteenth Symposium on the Foundations of Computer Science*. Providence.

PRIOR, A. 1967. *Past, Present and Future*. Oxford University Press.

ROBINSON, J. A. 1965. A Machine–Oriented Logic Based on the Resolution Principle. *ACM Journal 12,* 1 (Jan.), 23–41.

SCHWENDIMANN, S. 1998. A New One-Pass Tableau Calculus for PLTL. In *Proceedings of Tableaux 98*, H. de Swart, Ed. Lecture Notes in Artificial Intelligence, vol. 1397. Springer-Verlag, 277–291.

SISTLA, A. P. AND CLARKE, E. M. 1985. Complexity of Propositional Linear Temporal Logics. *ACM Journal 32,* 3 (July), 733–749.

SISTLA, A. P., VARDI, M., AND WOLPER, P. 1987. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science 49*, 217–237.

STEEDMAN, M. 1997. Temporality. In *Handbook of Logic and Language*. Elsevier North Holland, 895–935.

STIRLING, C. 1992. Modal and Temporal Logics. In *Handbook of Logic in Computer Science*. Oxford University Press.

TANSEL, A., Ed. 1993. *Temporal Databases: theory, design, and implementation*. Benjamin/Cummings.

VENKATESH, G. 1986. A Decision Method for Temporal Logic Based on Resolution. *Lecture Notes in Computer Science 206*, 272–289.

WEIDENBACH, C. 1997. Spass: Version 0.49. *Journal of Automated Reasoning 18*, 247–252.

WOLPER, P. 1983. Temporal Logic Can Be More Expressive. *Information and Control 56*, 72–99.

WOS, L., OVERBEEK, R., LUSK, E., AND BOYLE, J. 1984. *Automated Reasoning—Introduction and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey.

Received July 1999; revised December 1999 and June 2000; accepted June 2000