Connection Based Strategies for Deciding Propositional Temporal Logic

Subash Shankar and James Slagle
University of Minnesota
Minneapolis, MN 55455
{sshankar, slagle}@cs.umn.edu

Abstract

Connection methods have proven their value for efficient automated theorem proving in classical logics. However, these methods have not been extended to temporal logics due to the lack of a subformula property in existing proof procedures. We show that a slightly looser generalized subformula property exists for temporal logics. We then exploit this generalized subformula property to develop a temporal notion of polarities and connections, upon which we base an efficient proof procedure for propositional temporal logic. The proof procedure is structured around semantic tableau augmented with connections, and we propose a number of connection-based strategies. The procedure achieves many of the benefits of connection methods. The method is also sufficiently general to be extensible to other temporal logics. Experimental results indicate substantial speedup resulting from this approach.

1 Introduction

Although temporal logics (TLs) have proven to be of great value for a number of AI applications, a major drawback is the lack of efficient proof procedures. Sistla shows that the satisfiability problem for even the simple subset of linear-time propositional temporal logic (PTL) with only the eventually operator is NP-complete, and the satisfiability problems for a number of other PTL subsets are PSPACE complete [Sistla and Clarke 1985]. One common solution to this problem is to restrict the temporal logic to an executable subset requiring lower complexity. However, these restricted logics are often inconvenient due to limited expressive power, and their use is thus limited. In particular, such logics often lack the power to perform one of the most important applications of temporal logic - the modeling of temporal properties of hardware and software systems. An alternate approach is to develop more efficient proof procedures for the TLs. While this approach may not result in worst case complexity improvements, it often achieves great proof size reductions for many formulae.

Most TL proof procedures are based on semantic tableaux, which were first presented by Beth [Beth 1959] and later refined by Smullyan [Smullyan 1968]. In the semantic tableau procedure, a tableau is generated for the negation of the proposed theorem, and the resulting tableau is checked for satisfiability. The proof succeeds if the corresponding tableau is unsatisfiable. If the tableau is satisfiable, satisfiable paths in the tableau constitute models for the negation of the theorem, and thus counterexamples for the proposed theorem. Most PTL proof procedures are based on the tableau procedure presented by Wolper [Wolper 1985], though there are also some techniques based on Büchi automata [Vardi and Wolper 1986], clausal resolution [Cavalli and del Cerro 1984, Fisher 1991] and non-clausal resolution [Abadi 1987]. Unfortunately, these procedures are of exponential complexity, and it is thus desirable to guide the proofs intelligently.

Wolper modifies Wolper’s procedure to generate a PTL tableau procedure with substantial node-count reduction, due largely to the identification of nodes containing formulae which are unnecessary for the proof [Gough 1989]. Similar procedures have been applied to other TLs including a PTL with both future and past time operators [Gough 1989], and a polymodal logic with temporal and belief modalities [Wooldridge and Fisher 1994]. Although such procedures do an excellent job at reducing node-counts using algorithmic methods, they are not generally amenable to strategies that may apply when only a partial tableau is needed. In practice, in most domains where TL is useful, a single model for satisfiable formulae often suffices. For example, when formally modeling systems, a satisfiable tableau indicates a bug in the system. It is very useful to be able to generate a model corresponding to the bug; however the set of all models is unnecessary. Moreover, theorem prover performance for satisfiable formulae may be more important than for unsatisfiable formulae if we expect to use the theorem prover during design stages.

Thus, it is desirable to generate a PTL tableau procedure that can intelligently guide proofs (and non-proofs). A common method for first-order logic (FOL) is to statically identify potentially contradictory subformulae, represent these potential contradictions as connections, and use these connections to generate strategies to guide the proof procedure. A FOL connection graph procedure was proposed in [Kowalski 1975]. The connection method, also known as the matrix method, was proposed by Bibel (for FOL), and a good introduction to this method is given in [Bibel 1993]. The connection method can be seen as a variant of semantic tableaux in which formulae are represented clausally as matrices, one clause per row, and each path
These operators are included for convenience. Other traditional temporal operators, such as identities: \( OP = 0 \).

The basic proof procedure is a variant of the one in [Wolper 1985]. There are two stages:

1. Construct the tableau as a directed graph\(^1\).
2. Test the resulting graph for satisfiability.

Tableau construction proceeds by reducing leaf nodes according to a set of tableau rules. Tableau rules are represented in standard notation and may be linear or branching.

**Example 1:**

\[
\begin{array}{c}
\text{(name)} \\
F_{11}, \ldots, F_{1p}, F_{21}, \ldots, F_{2q}
\end{array}
\]

If this rule is applied to the formula \( F \), it produces two children: one with \( F_{11}, F_{12}, \ldots, F_{1p} \), and one with \( F_{21}, F_{22}, \ldots, F_{2q} \) (the exact mechanics of node construction are discussed later). If the rule has only one child, it is said to be a linear rule.

The set of tableau rules is as follows:

\[\begin{array}{ll}
\text{(alpha)} & \frac{P \land Q}{P}, \frac{P \land Q}{Q} \\
\text{(beta)} & \frac{P \lor Q}{P}, \frac{P \lor Q}{Q} \\
\text{(nu)} & \frac{P}{P}, \frac{P}{\neg P} \\
\text{(pi)} & \frac{P}{P}, \frac{P}{\neg P} \\
\text{(tau)} & \frac{\neg P}{\neg Q}, \frac{\neg P}{\neg Q} \\
\text{(upsilon)} & \frac{P \land Q}{P \land Q} \\
\text{(sigma)} & \frac{P \\ P \lor Q}{P} \\
\end{array}\]

where \( B_1, \ldots, B_m \) are all either atoms or negations of atoms. Formulae to which a tableau rule is applicable are referred to as formulae of that type (e.g., a formula preceded by a \( \square \) is referred to as a nu-formula).

Before presenting the tableau procedure, the following definitions are needed:

**Definition 1** An **eventuality formula** is a formula of the form \( \Box Q \), \( P \cup Q \), or \( \neg \Box Q \). \( Q \) is said to be the eventuality for the first two cases, and \( \neg Q \) for the third case.

**Definition 2** An **elementary formula** is a formula of the form \( \Box P \), \( \Box P \), \( A \), or \( \neg A \) where \( A \) is atomic.

An attempted proof of a formula is performed by negating the formula and constructing the tableau for the resulting formula, \( F \), in accordance with the following algorithm:

1. Preprocess \( F \) to eliminate any \( \bot \) and \( \top \) symbols, using standard tautologies.
2. Initialize the tableau, \( \mathcal{T} \), to a single node containing \( F \).
3. Repeat until there are no unexpanded nodes:
   (a) Choose an unexpanded node \( N \in \mathcal{T} \).
   (b) If every unmarked formula in \( N \) is elementary, create a node \( N_1 \) in accordance with the sigma rule.
   (c) Otherwise,
      • Select a non-elementary formula \( F \in N \).

---

\(^1\)While the actual proof is a directed graph, we refer to them as trees in this paper since the terminology of trees is convenient.
• If the tableau rule for $F$ is linear (i.e., $\eta, \alpha, \nu$), then create a node $N_1 = N \cup \{F_1, \ldots, F_p\}$ where the $F_i$'s are the formulae generated by the appropriate tableau rule.

• Otherwise, since the tableau rule for $F$ must have 2 branches, create nodes $N_1 = N \cup \{F_{21}, \ldots, F_{2q}\}$ with each branch containing the formulae generated by the corresponding branch of the appropriate tableau rule.

• In any newly created nodes, mark the formula $F$.

(d) For each newly created node $N_i$ which is already in $T$, add an arc in $T$ from $N$ to $N_i$. Otherwise ($N_i \notin T$), add $N_i$ to $T$ along with an arc from $N$ to $N_i$.

(e) Mark $N$ expanded.

(f) For each newly added node, if it contains both $F$ and $\neg F$ for some formula $F$, mark that node expanded.

A few details in the algorithm require further discussion. First, this procedure differs from traditional FOL tableaux since tableau rules produce new nodes instead of just new formulae. Second, formulae are marked instead of being deleted after they are expanded. The reasons for this will become clear later, since these formulae will be needed when checking whether the tableau is satisfiable. Finally, the sigma rule is applicable only to nodes for which other rules do not apply — i.e., nodes for which the only unmarked formulae are elementary formulae. Such nodes are referred to as states. As with traditional tableau, models can be read off tableau paths, with sigma rule decompositions corresponding to a forward movement of one time unit.

Since duplicate nodes are not created, the procedure results in a directed cyclic graph. To determine satisfiability, the following elimination rules are repetitively applied:

S1: If a node contains both a formula and its negation, eliminate that node.

S2: If a node which is either the root node or the child of a state contains an unsatisfiable eventuality formula (as defined below), eliminate that node.

S3: If all the successors of a node have been eliminated, eliminate that node.

The second rule is needed so that an eventuality formula which is never satisfied isn't found satisfiable. Although both the eventually and (strong) until operators have semantics requiring an eventuality to be satisfied, the nature of the pi and upsilon rules allows the satisfaction of the eventualities to be infinitely delayed. Thus, the following rule is used to determine whether an eventuality is satisfiable:

An eventuality formula in a node $N$ is satisfiable if there is a path in the tableau leading from $N$ to a node $N_1$ containing the corresponding eventuality.

If the root node is eliminable, then the tableau is unsatisfiable and the original formula is proven; otherwise, a counterexample may be read off the tableau with the understanding that sigma rule applications correspond to a movement in time.

Elimination rule S2 differs somewhat from the other elimination rules in that it can't be applied dynamically since it requires the generation of the relevant children nodes. This distinction becomes important later in our proof procedure strategies.

Example 2:
A naive tableau for the formula $(QUP) \land (\Box \neg P)$ is illustrated above. Each node in the tableau is drawn with three components: an identification number on the left, a marked part on the top, and an unmarked part on the bottom. Nodes 4 and 8 are unsatisfiable by S1 and are immediately eliminated. Node 6 is the child of a state and is unsatisfiable by rule S2 since it contains QUP which is unsatisfiable since no remaining descendant of node 6 contains $P$. Repeatedly applying rule S3, the root becomes unsatisfiable, and the proof is complete.

Theorem 1 A formula is unsatisfiable if its tableau is eliminable.

Proof:
As with other proofs, we provide a brief sketch here and a complete proof in [Shankar 1997]. Soundness of non-temporal rules is as usual, while soundness of temporal rules follows from the PTL identities: $\Box P \equiv P \land \square P$ and $PUQ \equiv Q \lor (P \land \diamond (PUQ))$. Soundness of the sigma rule follows from our interpretation of the state decompositions as movements in time in the resulting model. If the root is not eliminable, there must be a path through the tableau which contains no contradictions and satisfies all eventualities. This path constitutes a model. If the root is eliminable, a case analysis of elimination rules S1-S3 shows that the formula is unsatisfiable. 

Proof of termination relies on a generalized subformula property:
Definition 3 Given an input formula, $F$, the set of (generalized) subformulae of $F$ is the set of all subterms of $F$, possibly preceded with a $\neg$ and/or a $\diamond$ symbol. The definition implicitly assumes that the $\neg$ and $\diamond$ operators commute (as is the case). For the rest of this paper, the term subformula is used to refer to generalized subformulae.

Example 3:
Consider the formula $\Box P$. The set of subformulae is $\{P, \neg P, \diamond P, \neg \diamond P, \Box P, \neg \Box P, \diamond \Box P, \neg \diamond \Box P\}$.
Theorem 2 Any formula generated during a proof is a subformula of the input formula.

Theorem 3 The above proof procedure terminates.

Proof:
Let n be number of subterms of the input. Then, the number of subformulas of the input is at most 4n. Since the tableau procedure disallows identical nodes, there are at most 2^n nodes in the proof. Thus, the tableau construction terminates. □

The above is a very loose bound on the number of nodes, and is of no practical interest.

4 Polaris and Connections

The generalized subformula property can be exploited to provide representation schemes that eliminate dynamic for-

Both theorems:
Let n be number of subterms of the input. Then, the number of subformulas of the input is at most 4n. Since the tableau procedure disallows identical nodes, there are at most 2^n nodes in the proof. Thus, the tableau construction terminates. □

The above is a very loose bound on the number of nodes, and is of no practical interest.

4 Polaris and Connections

The generalized subformula property can be exploited to provide representation schemes that eliminate dynamic for-

Both theorems:
Let n be number of subterms of the input. Then, the number of subformulas of the input is at most 4n. Since the tableau procedure disallows identical nodes, there are at most 2^n nodes in the proof. Thus, the tableau construction terminates. □

The above is a very loose bound on the number of nodes, and is of no practical interest.
other part of the tableau and \( R' \) is also not eliminable for the same reason. If there is some path from \( R \) to \( N \) which is not eliminable in \( T \), then \( N' \) must also not be eliminable in \( T' \). Moreover, any eventuality on that path in \( N' \) remains satisfiable in \( T' \) since \( F \) has no connections. Thus, \( R' \) is also not eliminable.

Thus, \( R \) is eliminable iff \( R' \) is eliminable. \( \square \)

Of course, \( F \) must be kept if a model is desired for satisfiable tableaux. In our theorem prover, we keep \( F \) as a deleted formula, and output it as part of the model. We also avoid the creation of node \( N' \) by simply deleting \( F \) after declaring any eventuality formulae it satisfies as satisfiable.

The next strategy prunes trees once they can no longer be found unsatisfiable:

**Strategy 2** If a node contains no connections, declare the node satisfiable and terminate the tree at that node.

**Proof** (of correctness):
This strategy is equal to a finite number of applications of Strategy 1, and is thus sound. \( \square \)

Strategies 1 and 2 are related to the PURE principle in connection graph procedures. In connection graphs, the PURE principle allows for deletion of clauses which contain literals with no complementary literals (i.e. pure literals). Our strategies are, however, somewhat more general even in the non-temporal case since they apply to arbitrary formulae instead of just literals.

The next strategy is one of the most powerful strategies afforded by our procedure, and does not seem to be achievable in many other PTL decision procedures. First, note that we are not interested in the generation of all possible models for formulae which are found to be satisfiable; one model suffices.

Under the above assumption, the following strategy eliminates the expansion of many nodes whose satisfiability can be deduced from the satisfiability of other nodes:

**Strategy 3** Suppose a node \( N_1 \) is a subnode of node \( N_2 \), and \( N_2 \) is not reachable from \( N_1 \). Then, expand only node \( N_1 \) and declare \( N_2 \) eliminable if \( N_1 \) is eliminable.

Note that if \( N_1 \) is not eliminable, \( N_2 \) may or may not be eliminable. However, this is not of concern since we are not interested in finding all possible models for satisfiable formulae.

**Proof** (of correctness):
Consider the pruned tableau.

First, suppose that \( N_1 \) is eliminable. Then, the set of formulae in \( N_1 \) is satisfiable. But, since \( N_1 \subseteq N_2 \), this implies that the set of formulae in \( N_2 \) is also unsatisfiable and \( N_2 \) is thus eliminable in the original tableau. Thus the root in the pruned tableau is eliminable iff the root of the original tableau is.

Now, suppose that neither \( N_1 \) nor the root is eliminable. Then, any path from the root going through \( N_1 \) constitutes a model for the tableau.

Finally, suppose that \( N_1 \) is not eliminable and the root is eliminable. Then, there must be some ancestor of \( N_1 \), \( A \) which contains an unsatisfiable eventuality formula \( F \). Then, by the nature of the tableau rules, \( N_1 \) must contain a formula corresponding to the next-time branch of \( F \) along with formulae that can be used to contradict \( F \)'s eventuality at every future timepoint. But since \( N_1 \subseteq N_2 \), the root in the original tableau is also eliminable. \( \square \)

This strategy is a temporal generalization of the SUBS rule in FOL theorem proving. The SUBS rule allows for deletion of clauses which are supersets of other clauses, since they are subsumed by the smaller clauses. If our tableau procedure is applied only to sets of clauses where the clauses are conjunctive and connected by disjunctions, Strategy 3 is identical. Of course, the strategy is more general, since it applies to arbitrary non-clausal formulæ.

Branching can exponentially increase a tree's size, and it is desirable to reduce the number of branches. To reduce the number of branches, the following strategy is used:

**Strategy 4** If node \( N \) contains multiple unmarked branching formulae, and expansion of one of these formulæ, \( F \), would result in no connections to the newly generated formulæ in one branch, select \( F \) for expansion.

**Proof** (of correctness):
Since the correctness of the procedure is independent of the order in which rules are applied, the proof is trivial. \( \square \)

In the implementation, this strategy is actually used in conjunction with Strategy 1 and possibly with Strategy 3 to eliminate branches completely.

**Example 5:**
Using set notation, let \( N = \Gamma \cup \{\Diamond \rho, \Box \neg \rho \} \) where \( \Gamma \) is a set of branching formulæ with no connections to \( \rho \). Then Strategy 4 selects \( \Diamond \rho \) for expansion, generating the two children node \( N_1 = \Gamma \cup \{\ast \Diamond \rho, \ast \Box \neg \rho \} \) and \( N_2 = \Gamma \cup \{\ast \Diamond \rho, \ast \Diamond \rho, \ast \Box \neg \rho \} \) (using \( \ast \) to denote marked formulæ). Strategy 1 is then applied to delete \( \rho \) from \( N_1 \), and Strategy 3 can then be applied to prevent expansion of \( N_2 \). Thus, we only need to expand \( \Gamma \cup \{\ast \Diamond \rho, \ast \Box \neg \rho \} \).

When used this way, this strategy is actually a technique used to allow for earlier application of Strategy 1. If it is applied to a beta formula, it serves as a generalization of the disjunctive subformula deletion and UNIT rules used in connection methods. Otherwise, it is applied to a pi, tau, or upsilon formulæ, and it becomes a temporal generalization of the disjunctive subformula deletion rule. Note that a similar strategy can be added for conjunctive subformula deletion; however, we treat that case through other means.

The above strategies apply to traditional as well as temporal logics; however our notion of temporal polarities and the resulting connections allows the rules to apply to PTL. In practice, many PTL proofs repeat proofs of properties for future timepoints as well as the current one, though the same proof might apply. The following strategy alleviates this problem in many cases.

**Strategy 5** Suppose a node, \( N \) contains an eventuality formula, \( F \), with corresponding eventuality \( E \). Moreover, suppose every connection from \( E \) is to a term which is in the scope of only nu or alpha formulæ, and in the scope of at least one nu formulæ. Then, expand \( F \) without branching, by ignoring the next-time branch.

**Proof** (of correctness):
Expansion of \( F \) generates \( E \) in the current-time branch. By the nature of the nu, alpha, and sigma rules, the conditions on the connection ensure that every contradiction to \( E \) at the current time point also occurs at all future time points and vice versa. Thus, if the current-time branch is eliminable due to a contradiction to \( E \), then the next-time branch is also eliminable. The next-time branch may thus be ignored. \( \square \)

The conditions for Strategy 5 need to be checked before
node expansion, since expanding other formulae may make
the strategy inapplicable.

Strategy 5 is similar to standard theorem proving rules
which eliminate tautologies from further consideration.
Traditional propositional tautologies all apply, and we
present several other similar temporal strategies [Shankar
1997].

Example 6:
Consider the tableau of Example 2. This strategy applies
to node 2, from which the $Q(P)\Box$ is expanded on to create a
new node 3. Expanding on $\Box\neg P$ in node 3 results in node
4 which is unsatisfiable. Thus, the original 9 node tableau
is pruned to 4 nodes.

6 Results
In this paper we have described a PTL proof procedure
which achieves many of the benefits of connection methods
within the framework of semantic tableaux. Our strategies
are surprisingly simple; however, they achieve most of the
benefits of the major connection method reduction rules,
including the PURE, SUBS, TAUT, and UNIT; moreover,
we generalize these rules substantially in some cases. There
are several other connection method rules that apply only
to clausal methods.

Gough [Gough 1989] provides another PTL tableau pro-
cedure which accomplishes substantial node-count reduc-
tion. The reduction occurs mostly by identifying states
which contain unnecessary formulae. Strategy 3 performs a
similar function, though it is somewhat less general. While
Gough's procedure does an excellent job when the com-
plete tableau needs to be constructed, it may not be needed
on satisfiable tableaux, since partial tableaux often suf-
fice. For example, [Gough 1989] presents a tableau for
$\Diamond P \land \Box Q \land (P \land Q)$ which achieves a reduction from 26 to
8 states. However, since there are no connections in this
formula, Strategy 2 trivially determines that the formula is
satisfiable with no expansion.

We have implemented a theorem prover based on the
techniques presented here. Our results, based on a collec-
tion of 38 textbook theorems and 66 small but 'hard' prob-
lems are promising. The pruning strategies have shown an
average reduction in node count of 20-50% for unsatisfiable
tableaux. For satisfiable tableaux, the prover achieves arbi-
trarily high node reductions, though the average reduction
is closer to 50%. We expect that the actual time saved will
be greater than node count reductions due to the non-linear
complexity of certain critical graph algorithms.

7 Extensions
One of the most important advantages of our approach
is its generality. Since our notion of polarities (and thus,
connections) relies on the presence of a temporal context,
it is applicable to any discrete linear-time temporal logic
with a subformula property, though strategies may differ.
Moreover, since it does not change the basic nature of the
tableau, strategies are easily generatable, though all tra-
ditional reduction rules may not apply and many require
generalization. In many cases, the strategies we have pre-
ented here apply unchanged.

Some useful logics that our procedure may be applied
to include temporal logic with both past and future time
operators and polymodal logics with time as a modality.

Wolper [Wolper 1983] provides a tableau procedure for an
extended TL which augments PTL with temporal operators
definable by right-linear grammars, and our procedure is
also applicable to this. Our procedures may be extended
to the first-order versions of the above logics, using the tech-
niques presented in [Wrightson 1987] for first order logic.

References
diss., Stanford University.
for Linear Temporal Logic. In 4th Conference on
Automated Deduction, 113-127. Berlin: Springer-
Verlag.
Logic. In International Joint Conference on Artificial In-
telligence, 99-104. Menlo Park, Calif.: International Joint
Conference on Artificial Intelligence, Inc.
Gough, G. 1989. Decision Procedures for Temporal Logic,
Technical Report, UMCS-89-10-1, University of Manch-
ester.
Shankar, S. 1997. A Connection Based Proof Proce-
dure for Propositional Temporal Logic, Technical Report,
Dept. of Computer Science, University of Minnesota.
of Propositional Linear Temporal Logics. Journal of the
ACM 32:733-749.
Smullyan, R.M. 1968. First Order Logic. Berlin: Springer-
Verlag.
Techniques for Modal Logics of Programs Journal of
Wallen, L.A. 1990. Automated Deduction in Non-
dure for a Temporal Belief Logic. In International Confer-
Wolper, P. June-September 1985. The Tableau Method for
Temporal Logic: an Overview. Logique Et Analyse 110-
Information and Control 56:72-99.
Proceedings of the Australian Joint Artificial Intelligence
Conference, 553-566.

TEMPORAL REASONING 177