A Graph-Based Approach To Resolution In Temporal Logic*

Clare Dixon[1], Michael Fisher[2] and Howard Barringer[1]

[1] Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK.
{dixonc,howard}@cs.man.ac.uk

[2] Department of Computing, Manchester Metropolitan University, Chester Street, Manchester M1 5GD, UK.
michael@sun.com.mmu.ac.uk

Abstract. In this paper, we present algorithms developed in order to implement a clausal resolution method for discrete, linear temporal logics, presented in [Fis91]. As part of this method, temporal formulae are rewritten into a normal form and both 'non-temporal' and 'temporal' inference rules are applied. Through the use of a graph-based representation for the normal form, "efficient" search algorithms can be applied to detect sets of formulae for which temporal resolution is applicable. Further, rather than constructing the full graph structure, our algorithms only explore and construct as little of the graph as possible. These algorithms have been implemented and have been combined with sub-programs performing translation to normal form and non-temporal resolution to produce an integrated resolution based temporal theorem-prover.

1 Introduction

Although resolution has been widely used as a decision procedure in classical logics, decision procedures in temporal logic have usually been tableau or automata based, for example [Wol83, Gou84, VW86, SV89]. However, some resolution proof procedures for temporal logics have been developed, for example [CdC84, Ven86, AM90].

A naive application of the classical resolution rules to two complementary literals fails as the literals may occur in different moments in time. An obvious extension of the classical resolution rule to temporal logics is to try to resolve complementary formulae appearing within the context of the '$\Box$' (*always in the future*) and '$\Diamond$' (*sometimes in the future*) operators using the rule

$$\frac{\begin{array}{l} A \vee \Box p \\ B \vee \Diamond \neg p \end{array}}{A \vee B}$$

where A and B are formulae and p is a proposition. This rule is sound since, while '$\Box p$' states that p will always be true, '$\Diamond \neg p$' states that p will be false at some point in the

* The work of the first author was supported by SERC under a PhD Studentship.

future. However, due to the interaction (in discrete temporal logics) between '$\Box$' and '$\bigcirc$' (meaning *at the next moment in time*) operators, a formula implying $\Box p$ may be *hidden* within a set of other formulae containing '$\bigcirc$' operators. For example, the formula

$$\Box(a \Rightarrow \bigcirc a) \wedge a \wedge l \wedge \Box(a \Rightarrow \bigcirc l)$$

implies $\Box l$ although this is not immediately obvious, and so this formula should be resolvable with $\Diamond \neg l$.

To address these issues, a normal form for temporal formulae [Fis91, Fis92], together with a resolution method relying upon the structure of the formulae within the normal form [Fis91], have been developed. The resolution method uses both the classical resolution rule, in order to derive contradictions that occur solely within temporal contexts (termed *non-temporal resolution*), together with a new resolution rule, which derives contradictions *across* temporal contexts (termed *temporal resolution*).

In this paper, an implementation of the temporal resolution proof method is described. In particular, a graph-based approach to the detection of sets of formulae to which the temporal resolution rule can be applied is developed.

This paper is structured as follows. In §2, a description of the propositional temporal logic used in this paper is given, while, in §3, an outline of the temporal resolution method and an overview of the graph-based implementation of this method is presented. Details of the algorithms used are given in §4. The soundness and completeness of these algorithms are considered in §5. In §6, both related work and future work of the authors is outlined.

2 A Linear Temporal Logic

2.1 Syntax and Semantics

The logic used in this report is Propositional Temporal Logic (PTL), which is based on a linear, discrete model of time with finite past and infinite future. PTL may be viewed as a classical propositional logic augmented with both future-time and past-time temporal operators. Future-time temporal operators include '$\Diamond$' (*sometime in the future*), '$\Box$' (*always in the future*), '$\bigcirc$' (*in the next moment in time*), '$\mathcal{U}$' (*until*), '$\mathcal{W}$' (*unless* or *weak until*), each with a corresponding past-time operator. Since our temporal models assume a finite past, for convenience, two last-time operators are used namely '$\bullet$' (*weak last*) and '$\circledcirc$' (*strong last*). Thus for any formula A, $\circledcirc A$ is false, when interpreted at the beginning of time, while $\bullet A$ is true at that point. In particular, $\bullet$ **false** can only be satisfied when interpreted at the beginning of time.[3]

Models for PTL consist of a sequence of *states*, representing moments in time, i.e.,

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

Here, each state, s_i, contains those propositions satisfied in the i^{th} moment in time. As formulae in PTL are interpreted at a particular moment, the satisfaction of a formula f is denoted by

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

[3] Although this is not necesary it avoids having to deal with negated past-time formulae like $\neg\bullet$ **false**.

where σ is the model and i is the state index at which the temporal statement is to be interpreted. For any well-formed formula f, model σ and state index i, then either $(\sigma, i) \models f$ or $(\sigma, i) \not\models f$. For example, a proposition symbol, 'p', is satisfied in model σ and at state index i if, and only if, p is one of the propositions in state s_i, i.e.,

$$(\sigma, i) \models p \quad \text{iff} \quad p \in s_i.$$

The full syntax and semantics of PTL will not be presented here, but can be found in [Fis91].

2.2 A Normal Form for PTL

Formulae in PTL can be transformed to a normal form, Separated Normal Form (SNF), which is the basis of the resolution method used in this paper. SNF was introduced first in [Fis91] and has been extended both in [FN92] and [Fis92]. While the translation from an arbitrary temporal formula to SNF will not be described here, we note that such a transformation preserves satisfiability and so any contradiction generated from the formula in SNF implies a contradiction in the original formula. Formulae in SNF are of the general form

$$\square \bigwedge_i R_i$$

where each R_i is known as a *rule* and must be one of the following forms.

$$\begin{array}{lll}
\bullet\,\mathbf{false} \Rightarrow \bigvee_{b=1}^{r} l_b & & \text{(an } \textit{initial } \square\text{–rule)} \\
\circledcirc \bigwedge_{a=1}^{g} k_a \Rightarrow \bigvee_{b=1}^{r} l_b & & \text{(a } \textit{global } \square\text{–rule)} \\
\bullet\,\mathbf{false} \Rightarrow \diamondsuit l & & \text{(an } \textit{initial } \diamondsuit\text{–rule)} \\
\circledcirc \bigwedge_{a=1}^{g} k_a \Rightarrow \diamondsuit l & & \text{(a } \textit{global } \diamondsuit\text{–rule)}
\end{array}$$

Here k_a, l_b, and l are literals. The outer '$\square$' operator, that surrounds the conjunction of rules is usually omitted.

We take this opportunity to note a variant on SNF called merged-SNF (SNF_m) [Fis91], for combining pairs of rules, that will be used later in this paper. Given a set of rules in SNF, the relevant set of SNF_m rules may be generated by repeatedly applying the following rule.

$$\frac{\begin{array}{c}\circledcirc A \Rightarrow F \\ \circledcirc B \Rightarrow G\end{array}}{\circledcirc (A \wedge B) \Rightarrow F \wedge G}$$

The right hand side of this SNF rule generated may have to be further translated into Disjunctive Normal Form (DNF), if either F or G are disjunctive, to maintain the SNF rule structure. Thus, SNF_m represents all possible conjunctive combinations of SNF rules.

3 Overview of the Resolution Procedure

We here present a review of the temporal resolution method and an overview of its implementation. This provides the background necessary to understand the implementation techniques presented in this paper. For further details of the temporal resolution method the reader is referred to the work presented in [Fis91, Fis92]. The algorithms implemented, together with their correctness are presented in §4 and §5, respectively.

The clausal temporal resolution method consists of repeated applications of both 'non-temporal' and 'temporal' resolution steps on sets of formulae in SNF, together with various simplification steps.

3.1 The Non-Temporal Resolution Procedure

'Non-temporal' resolution consists of the application of standard classical resolution rule to formulae representing constraints at a particular moment in time, together with simplification rules for transferring contradictions within states to constraints on previous states.

The non-temporal resolution rule is a form of classical resolution applied between $\Box$–rules, representing constraints applying to the same moment in time. Pairs of initial $\Box$–rules, or global $\Box$–rules, may be resolved using the following (non-temporal resolution) rule where $\mathcal{L}_1$ and $\mathcal{L}_2$ are both last-time formulae (with the same last-time operator).

$$\frac{\begin{array}{r}\mathcal{L}_1 \Rightarrow A \lor r\\ \mathcal{L}_2 \Rightarrow B \lor \neg r\end{array}}{(\mathcal{L}_1 \land \mathcal{L}_2) \Rightarrow A \lor B}$$

Once a contradiction within a state is found using non-temporal resolution, the following rule can be used to generate extra constraints.

$$\frac{\circledcirc P \Rightarrow \mathbf{false}}{\bullet\, \mathbf{true} \Rightarrow \neg P}$$

This rule states that if by satisfying P in the last moment in time a contradiction is produced then P must never be satisfied in *any* moment in time. The new constraint therefore represents $\Box\neg P$ (though it must first be translated into SNF before being added to the rule-set).

The non-temporal resolution process terminates when either no new resolvents are derived, or **false** is derived in the form of one of the following rules.

$$\begin{array}{r}\bullet\, \mathbf{true} \Rightarrow \mathbf{false}\\ \bullet\, \mathbf{false} \Rightarrow \mathbf{false}\\ \circledcirc\, \mathbf{true} \Rightarrow \mathbf{false}\end{array}$$

3.2 Temporal Resolution

The temporal resolution process consists of (possibly multiple) applications of the temporal resolution rule resolving together formulae with the $\Box$ and $\Diamond$ operators. However, as described in §1, the interactions between the '$\bigcirc$' and '$\Box$' operators in PTL ensure that the definition of such a rule is non-trivial. Further, as the translation to SNF restricts the rules to be of a certain form, resolution will be between a $\Diamond$–rule and a *set* of rules that together imply an invariance which will contradict the $\Diamond$–rule. Thus, given a set of rules in SNF, then for every rule of the form $\mathcal{L}Q \Rightarrow \Diamond\neg p$ where $\mathcal{L}$ may be either of the last-time operators, temporal resolution may be applied between this $\Diamond$–rule and a set of global $\Box$–rules, which taken together force p always to be satisfied. Here if Q is satisfied then the rule is satisfiable unless the set of global $\Box$–rules force p to always be **true**. In this case, we must ensure that there is no situation where both Q and any of these rules are satisfied at the same moment. Similarly, once Q has been satisfied we must also ensure that none of the global $\Box$–rules are satisfied *unless* $\neg p$ is satisfied. To resolve with the above rule then, a set of rules must be identified so that, the set of rules, when combined, have the effect of $S \Rightarrow \Box p$. Here, the S may only be of the form $\circledcirc A$ where A is in DNF. So the general resolution rule, written as an inference rule, becomes

$$\begin{array}{r@{\;}l} \circledcirc A \Rightarrow & \Box p \\ \mathcal{L}Q \Rightarrow & \Diamond\neg p \\ \hline \bullet\,\mathbf{true} \Rightarrow & \neg A \vee \neg Q \\ \mathcal{L}Q \Rightarrow & (\neg A)\,\mathcal{W}\,(\neg p) \end{array}$$

where the first resolvent shows that both A and Q cannot be satisfied together, and the second that once Q has occurred then $\neg p$ must occur (i.e. the eventuality must be satisfied) before A can be satisfied.

The full temporal resolution rule is given by the following

$$\begin{array}{r@{\;}l} \circledcirc A_0 \Rightarrow & F_0 \\ \dots & \dots \\ \circledcirc A_n \Rightarrow & F_n \\ \mathcal{L}Q \Rightarrow & \Diamond\neg p \\ \hline \bullet\,\mathbf{true} \Rightarrow & \neg Q \vee \bigwedge_{i=0}^{n} \neg A_i \\ \mathcal{L}Q \Rightarrow & (\bigwedge_{i=0}^{n} \neg A_i)\,\mathcal{W}\,\neg p \end{array} \quad \begin{array}{c}\text{with}\\ \text{side}\\ \text{conditions}\end{array} \left\{ \begin{array}{r} \text{for all } 0 \le i \le n \vdash F_i \Rightarrow p \\ \text{and} \vdash F_i \Rightarrow \bigvee_{j=0}^{n} A_j \end{array} \right\}$$

where the side conditions ensure that each $\Box$–rule makes p true and the right hand side of each $\Box$–rule ensures that the left hand side of one of the $\Box$–rules will be satisfied.[4] So if any of the A_i are satisfied then p will be *always* be satisfied, i.e.,

$$\circledcirc \bigvee_{k=0}^{n} A_k \Rightarrow \Box p.$$

[4] In its original presentation [Fis91] there was a third side condition but this was found to be unnecessary [Pei94].

Such a set of rules are known as a *loop* in p.

Thus, the first of the new resolvents ensures that none of the rules which form a loop in p can occur at the same time as a Q occurs, so none of the left-hand sides of the rules which ensure a loop in p, (i.e., A_i for $i = 0, 1, \ldots, n$) can be satisfied at the same time as Q. The second new resolvent ensures that once Q has been satisfied, meaning that the eventuality $\Diamond\neg p$ must be satisfied, none of the conditions for entering a loop is allowed to occur, i.e., none of the A_i (for $i = 0, 1, \ldots, n$) can be satisfied until the eventuality $(\neg p)$ has been satisfied. For more details of the resolution method see [Fis91].

3.3 Implementing Temporal Resolution: An Overview

To apply the temporal resolution rule, an appropriate set of $\Box$–rules must be detected for resolution with one of the $\Diamond$–rules. As such sets may not be immediately apparent from inspection of the rule-set, graph-theoretic techniques are used to identify them.

Representing Rules as Graphs. In order to apply various graph-based algorithms, we must first represent the SNF rules as a graph. We use global $\Box$–rules in SNF to represent the edges in this graph. For example a rule $\bigcirc a \Rightarrow b$ could be represented as shown in Fig. 1a. Any rule containing a disjunct on its right-hand side, for example

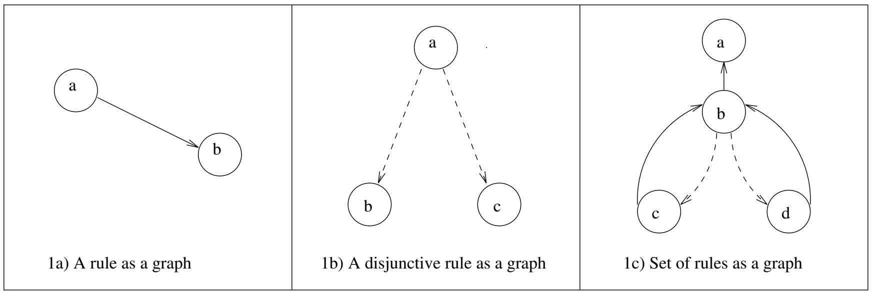


Fig. 1. Some rules represented as graphs

$\bigcirc a \Rightarrow b \vee c$, can be represented as in Fig. 1b using special 'disjunctive' arcs. In fact, for our purposes we need not distinguish between the two types of arcs as the edges leading out of a node will only relate to one rule (in SNF_m), however, we do so for clarity. The set of rules $\bigcirc b \Rightarrow c \vee d$, $\bigcirc c \Rightarrow b$, $\bigcirc d \Rightarrow b$, $\bigcirc b \Rightarrow a$ can be represented as shown in Fig. 1c.

Although there are other ways that sets of SNF rules could correspond to graph structures (e.g., see [Fis91]), the above representation is particularly suitable for our needs.

Overview of the Algorithm. The aim of our algorithm is to detect loops by constructing as little of the graph as possible. Assuming we wish to apply temporal resolution to $\mathcal{L}Q \Rightarrow \Diamond\neg p$, then we would like to construct a portion of the graph representing $S \Rightarrow \Box p$, i.e., a subgraph where p is generated at each node as the graph is built. If this subgraph forms a strongly-connected component, we have a loop in p, that represents $S \Rightarrow \Box p$, and we can apply the temporal resolution rule. In fact other subgraphs may be sufficient to ensure we have a loop, as long as each edge from a node leads back into the subgraph. So, the basic idea underlying the algorithm is to construct the graph piecemeal until either a suitable subgraph representing a loop is found, or until we have tried all possible rules and combinations of rules to construct the "next" node from each node in the graph and no loop has been found (in which case no temporal resolution was possible). Note, however, that the only edges that we use to construct the graph are ones which will give p at each node; all other possible edges are ignored.

For example if we are trying to resolve with $\mathcal{L}Q \Rightarrow \Diamond\neg p$, given a set of global $\Box$-rules R of which $\{\bigcirc b \Rightarrow c \lor d,\ \bigcirc c \Rightarrow b,\ \bigcirc d \Rightarrow b,\ \bigcirc b \Rightarrow a,\ \bigcirc a \Rightarrow p,\ \bigcirc b \Rightarrow p,\ \bigcirc c \Rightarrow p,\ \bigcirc d \Rightarrow p\}$ is a subset, the portion of the graph representing these rules is given in Fig. 1c. The subgraph formed from the nodes b, c and d represent a loop in p as once we generate one of these nodes then we will always be led back to a node in this subgraph and p is always generated.

Alternative Algorithms. An initial naive approach was to take all possible combinations of rules, represent them as an AND/OR graph and search for the strongly connected components in this graph where p held at every node [Fis91]. Our approach is an improvement on this basic algorithm in that we only construct as little of the graph structure as necessary, combining as few SNF rules as possible. Further, in contrast to the naive approach, we only use rules or combinations of rules that will ensure that p is produced at every node.

4 Implementing Temporal Resolution: Detailed Algorithms

The algorithms for loop detection, the different search methods, and selection and combination of rules into SNF_m are given in this section, followed by several examples.

4.1 Definitions of Terms

In the description of the algorithm and its usage the following terms will be used. Given a set of rules R, a graph of R, $G_R = (N, E)$ is the set of nodes, N, and the set of edges E. A *node*, n_i, is a set of literals l_k, i.e. $n_i = (l_1, l_2, \ldots, l_m)$. The set of edges E is the distributed union of *sets of disjunctive edges for rules*, i.e.

$$E = \bigcup_{i=1}^{n} D_i.$$

A *set of disjunctive edges for a rule*, D_i are directed edges in the graph formed from the rule

$$\bigcirc A \Rightarrow \bigvee_{j=1}^{n} B_j.$$

so that $D_i = \{(A, B_1), \ldots, (A, B_n)\}$. A *path* is a structured list of nodes

$$[n_1, n_2, \ldots, n_m, [n_{m11}, n_{m12} \ldots], [n_{m21}, n_{m22} \ldots], \ldots, [n_{mr1}, n_{mr2} \ldots]]$$

such that $(n_i, n_{i+1}) \in E$ for $i = 1, 2, \ldots, m-1$, $(n_{mxi}, n_{mxi+1}) \in E$ for $x = 1, 2 \ldots, r$, $i = 1, 2, \ldots$, and $(n_m, n_{mx1}) \in E$ for $x = 1, 2, \ldots, r$ where each sublist may be structured in the same way. A *loop* is a subgraph $G_l = (N_l, E_l)$ of G such that

1. for every edge $e_j = (n_j, n_k)$ such that $e_j \in E_l$, then $n_j \in N_l$ and $n_k \in N_l$; and
2. for every n_j in N_l there exists an edge $e_j = (n_j, n_k)$ leading out of this node and $e_j \in E_l$ and if $e_j \in D_r$ then $D_r \subseteq E_l$, where D_r is a set of disjunctive edges for a rule.

A *self-contained loop* is one where there is only one element in each set of disjunctive edges for a rule. A *partial loop* is a subgraph of the loop where $D_r \not\subseteq E_l$ for some r. Typically, we will have followed one of the disjuncts, but will have others remaining to be explored, and will have found a looping path back to one of the nodes in the partial loop.

4.2 The Loop Search Algorithm

The graph is searched for loops as it is constructed. It is constructed in a depth-first manner using the SNF rules i.e. when there is a choice of rules to use in order to expand the graph, only one is followed. If we find a situation where no rules may be applied to create a new node, then backtracking is invoked and a different choice is made at an earlier choice point.

The algorithm used (given that we want to resolve with $\mathcal{L}Q \Rightarrow \Diamond\neg p$) is given below. The forwards and backwards search algorithms mentioned in the steps 4 and 6 are described in §4.3.

1. Search for all the rules of the form $\bigcirc R_i \Rightarrow p$
2. Use the literal (or conjunction of literals) R_i as the initial nodes in the graph.
3. Set the current node, n_1, equal to the next initial node if one is available, and path equal to $[n_1]$, otherwise the algorithm terminates having found no new loops.
4. Perform a backwards search from n_1 until either
   (a) no loop has been detected from any of the successors to n_1 - repeat step 3;
   (b) a self-contained loop has been found - the algorithm terminates with this loop;
   (c) a partial loop has been found - remove any nodes that do not form part of the loop (the prefix to the loop) and continue processing with step 5.
5. Set n_1, the current node equal to a new disjunct if one is available, otherwise the algorithm terminates with a loop.
6. Perform a forward search from n_1 until either

(a) no loop has been detected from any of the successors to n_1, backtrack to where the disjunctive rule was used (i.e. if previously performing a backwards search to within step 4, or if previously performing a forwards search to within step 6) and continue processing;
(b) a self-contained loop has been found - the algorithm terminates with a loop;
(c) a partial loop has been found - continue processing with step 5.

For a rule-set with more than one eventuality the algorithm is carried out using each $\Diamond$–rule in turn until a new loop is detected or there are no more $\Diamond$–rules to process.

4.3 Backward and Forward Search Algorithms

If the initial nodes in our graph are R_j, for $j = 1, \ldots, m$ a new node n_{i+1} may be generated using the backwards or forwards depth-first search algorithms, where applicable, from the current node n_i (we abuse our notation slightly and are assuming that n_i represents a literal or conjunction of literals) if there exists a global $\Box$–rule

$$\circledcirc \bigwedge_{a=1}^{g} k_a \Rightarrow \bigvee_{b=1}^{r} l_b$$

and the following conditions are satisfied.

1. For backwards search there exists a b such that $1 \leq b \leq r$, $l_b \Rightarrow n_i$, and for all b, $l_b \Rightarrow \bigvee_{j=1}^{m} R_j$, and $\bigwedge_{a=1}^{g} k_a \Rightarrow \bigvee_{j=1}^{m} R_j$ then n_{i+1} will be $(k_1, k_2, \ldots, k_g)$. One of the disjuncts from the right hand side of the global $\Box$–rule implies the current node, the other disjuncts and the new node, n_{i+1} imply one of the initial nodes, thus ensuring p in the previous node and the new node will be the conjunction of literals from the left hand side of the rule.
2. For forwards search $n_i \Rightarrow \bigwedge_{a=1}^{g} k_a$ and for all b such that $1 \leq b \leq r$, $l_b \Rightarrow \bigvee_{j=1}^{m} R_j$ then n_{i+1} will be l_b for some b such that $1 \leq b \leq r$. The current node implies the literal or conjunction of literals on the left hand side of the rule, all the disjuncts on the right hand side of the global $\Box$–rule imply one of the initial nodes and the next node, n_{i+1}, will be one of the disjuncts.

When the global $\Box$–rule used for backwards or forwards search has $r > 1$ we store the disjuncts that have not been matched to the current node during backwards search, or used as the new node in forwards search, in a list for future processing.

4.4 Combinations of Rules

If no successor to the current node can be found, rules are combined using the SNF_m transformation in an attempt to generate a successor. To avoid the expense of combining all the rules, only the rules which may generate an appropriate successor are extracted from the rule-set for combination purposes. Initial rules can also be used for combination purposes. Any literals p (assuming the eventuality we are resolving with is $\Diamond\neg p$) generated through combinations on the right hand side of a combined rule, not being explicitly noted, as it is assumed that p holds at each node in the graph.

4.5 Addition of Rules

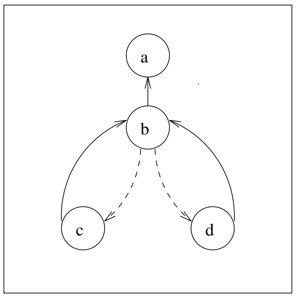
Before each cycle of temporal resolution we need to add the (valid) rule $\bigcirc\,\mathbf{true} \Rightarrow l \vee \neg l$ for any pair of complementary literals, l and $\neg l$, occurring on the left hand sides of any rules. This is to enable us to be able to detect loops in sets of clauses such as $\bullet\,\mathbf{false} \Rightarrow \Diamond\neg p$, $\bigcirc\neg l \Rightarrow p$, $\bigcirc\, l \Rightarrow p$.

4.6 Examples

The following examples, together with their explanations, show how the algorithm is used to detect loops from several rule-sets. It will be assumed that the original formulae have already been translated into SNF and, with each of these examples, the graph structure constructed during the search will be given.

Example 1: A Disjunctive Loop

1. $\bigcirc\, b \Rightarrow c \vee d$
2. $\bigcirc\, c \Rightarrow b$
3. $\bigcirc\, d \Rightarrow b$
4. $\bigcirc\, b \Rightarrow a$
5. $\bigcirc\, a \Rightarrow l$
6. $\bigcirc\, b \Rightarrow l$
7. $\bigcirc\, c \Rightarrow l$
8. $\bigcirc\, d \Rightarrow l$
9. $\bullet\,\mathbf{false} \Rightarrow \Diamond\neg l$



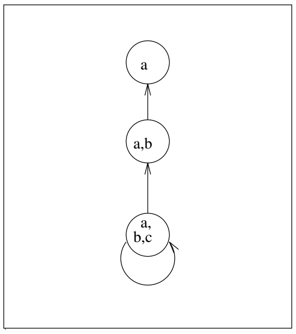
1. The $\Diamond$–rule is rule 9, and so we search for loops in l.
2. Initial nodes are (a), (b), (c), and (d), from rules 5–8. Without loss of generality the current node is set to (a) and path to [(a)].
3. Backwards search matches a with the right hand side of rule 4. The new current node is b, and the path is [(b), (a)].
4. Backwards search matches (b) with the right hand side of rule 2. The new current node becomes (c), and the path becomes [(c), (b), (a)].
5. Backwards search matches (c) with the right hand side of the disjunctive rule 1. The new current node becomes (b), and (d) is stored waiting to be solved. The path is [(b), [(c), (b), (a)], [(d)]]. As (b) has already been visited we have found a partial loop, and need to begin the search from remaining disjuncts. The node (a) does not form part of this partial loop and is removed from the path. The path becomes [(b), [(c), (b)], [(d)]].

6. The remaining disjunct d is solved using forward search. Forward search matches d with the left hand side of rule 3. The current node becomes (b), the sub-path for the disjunct becomes $[(d), (b)]$. As (b) has already been searched through before, we have found another partial loop, so the node (d) in the previous path is replaced by this sub-path to give a new path $[(b), [(c), (b)], [(d), (b)]]$.
7. There are no remaining disjuncts left to solve so the algorithm terminates.

If either of the other three initial nodes had been used to start the search, or we had selected rule 3 instead of rule 2 in step 4 above, the algorithm would still have found the same loop.

Example 2: A Loop Using Combinations of Rules

1. $\bigcirc a \Rightarrow l$
2. $\bigcirc b \Rightarrow l$
3. $\bigcirc c \Rightarrow l$
4. $\bigcirc (a \wedge b) \Rightarrow a$
5. $\bigcirc (b \wedge c) \Rightarrow b$
6. $\bigcirc (a \wedge c) \Rightarrow c$
7. $\bullet\, \textbf{false} \Rightarrow \Diamond \neg l$



1. From rule 7, we look for a loop in l.
2. Initial nodes are (a), (b), and (c), from rules 1, 2, and 3. The current node is set to be (a) and the path is $[(a)]$.
3. Backwards search from a matches a with the right hand side of rule 4. The new current node is set to (a, b), and the path is $[(a, b), (a)]$.
4. Backwards search is carried out attempting to match the right hand side of a rule with node (a, b). As no rule matches, rules 4 and 5 are extracted as they will give us an a, or a b, on the right hand side, and are combined using SNF$_m$ to give a new rule $\bigcirc (a \wedge b \wedge c) \Rightarrow a \wedge b$. The right hand side of this new rule now matches the current node, so the new current node is set to (a, b, c) and the path becomes $[(a, b, c), (a, b), (a)]$.
5. Again no rule in SNF matches the current node (a, b, c). Rules 4, 5, and 6 are extracted as they will generate the required nodes on the right hand side to match with the current node. Combining these rules into SNF$_m$ gives a new rule $\bigcirc (a \wedge b \wedge c) \Rightarrow (a \wedge b \wedge c)$. The current node matches with the right hand side of this rule, the new current node is set to be the left hand side of the rule (a, b, c), and the path becomes $[(a, b, c), (a, b, c), (a, b), (a)]$. As we have already searched through node (a, b, c), we have found a loop. The nodes (a, b) and (a) are removed as they are not part of the loop, leaving the path as $[(a, b, c), (a, b, c)]$, and the algorithm terminates as there are no disjuncts to process.

If either of the other two initial nodes (b) and (c) had been had been used to start the search from, the algorithm would still have found the same loop.

4.7 Integration of the Temporal Resolution Step into a Theorem Prover

The implementation of the temporal resolution step has been integrated with other sub-programs performing the translation of formulae into SNF and non temporal resolution to produce a complete resolution theorem prover for PTL. The resolution theorem prover has been implemented on Sun Workstations using SICSTUS PROLOG [CW91].

5 Formal Analysis of the Algorithm

In this section, we provide outline proofs exhibiting the correctness of the algorithm. Full versions of the proofs are available, but are omitted due to space restrictions.

5.1 Soundness

If the algorithm terminates detecting a loop for the literal p then the set of rules R which give rise to this loop imply the invariance of p.

Proof. Assume that a loop in the literal p has been found. Let the rules, R, in SNF_m which represent this loop be

$$\left\{ \begin{array}{l} \bigcirc A_0 \Rightarrow B_0 \\ \quad\dots \qquad \dots \\ \bigcirc A_n \Rightarrow B_n \end{array} \right\}$$

The matching of the left or right hand sides of rules to the current node, only stopping when we have reached a node we have visited before and when there are no remaining disjuncts means that we will always be brought back to a rule in the set. The restriction of only allowing the expansion of nodes using rules that will generate a p means that p is ensured at each node. So once one of the A_i rules is triggered, no edge can lead out of the loop and p will always be generated i.e.

$$\bigcirc \bigvee_{i=1}^{n} A_i \Rightarrow \Box p$$

So R does imply a $\Box$–formula.

5.2 Completeness

If there is a set of rules R of the form $\bigcirc A_i \Rightarrow B_i$ in SNF_m for $i = 0, \dots, n$ such that

$$\bigcirc \bigvee_{i=1}^{n} A_i \Rightarrow \Box p$$

then, by taking suitable choices the algorithm will find the loop corresponding to this set of rules.

Proof. The proof refers to the algorithm described in §4.2, the numbers in brackets referring to the relevant part of the algorithm. Let the rules in SNF be

$$\chi = \left\{ \begin{array}{c} \circledcirc X_0 \Rightarrow Y_0 \\ \ldots \quad \ldots \\ \circledcirc X_p \Rightarrow Y_p \end{array} \right\} \qquad \text{and} \qquad \psi = \left\{ \begin{array}{c} \circledcirc Z_0 \Rightarrow W_0 \\ \ldots \quad \ldots \\ \circledcirc Z_q \Rightarrow W_q \end{array} \right\}$$

where the members of χ are the rules that have been combined to make the $\circledcirc A_i \Rightarrow B_i$ rules and the members of ψ are the remaining SNF rules.

Initially the algorithm searches for rules of the form $\circledcirc R_i \Rightarrow p$ (step 1 of the algorithm). Some of these are going to be in the set χ in order to generate p at every node of the loop. The search for a node always commences from one of these rules (step 2 and 3 of the algorithm) so at some point we will begin the search for the loop from a rule of the form $\circledcirc R_i \Rightarrow p$ which is in the set χ.

Assume we've picked the rule $\circledcirc A_i \Rightarrow B_i$ in SNF_m, then to generate the next node we need to choose a rule of the form $\circledcirc C \Rightarrow D$ in SNF_m. There are two options:

1. $\circledcirc C \Rightarrow D$ is made from combining at least one rule from ψ with zero or more of the rules from χ.
2. $\circledcirc C \Rightarrow D$ is made by combining one or more rules from χ (and none of the rules from ψ) and is equivalent to one of the $\circledcirc A_i \Rightarrow B_i$ rules.

By guiding the search always to take the second option, rules selected will all be equivalent to one of the $\circledcirc A_i \Rightarrow B_i$ rules and will be made from combining one or more of the rules from χ until a $\circledcirc A_i \Rightarrow B_i$ rule generated is already in the set. If we have no remaining disjuncts we are done and a loop has been detected (step 4(b) of the algorithm).

However, if we have used a disjunctive rule (step 4(c) of the algorithm), then we must search from each remaining disjunct in turn. Again by guiding the search always to take combinations of rules from χ to produce one of the $\circledcirc A_i \Rightarrow B_i$ rules until a the $\circledcirc A_i \Rightarrow B_i$ rule generated is already in the set, a partial loop will have been detected (step 6(c) of the algorithm). Further partial loops are found from the remaining disjuncts until there are no disjuncts remaining (step 5 of the algorithm). As we know there is a loop, then by combining one or more rules from χ it is always possible to detect it.

6 Conclusions and Further Work

6.1 Related Work

A variety of resolution proof systems have been developed for modal and temporal logics. Methods for modal resolution may be found in [Min90, Ohl88, EdC89] for example. However the problems considered in this report, in detecting the rules which together imply a $\square$–rule, with which to apply the resolution rule, due to the interaction between the $\square$ and $\bigcirc$ operators do not occur as generally modal logics do not use the $\bigcirc$ operator.

Resolution methods for temporal logics can be found in [Aba87, AM90, CdC84, Ven86] for example. The systems described in [CdC84, Ven86] are both for propositional linear temporal logic and, as we do, both convert the formula into a normal form before the application of any resolution rules, whereas the resolution method outlined in [Aba87, AM90] for propositional and first order temporal logic is, unlike ours, non-clausal. None of the systems, include any past-time temporal operators while the implementation outlined in [Aba87, AM90] is limited to propositional temporal logic for operators $\bigcirc$, $\square$ and $\lozenge$.

6.2 Future Work

The use of resolution strategies to guide the refutation in classical logics, have been fully described in [CL73, WOLB84, Lov78]. The application of these strategies could certainly be implemented for the non-temporal resolution and their development investigated for use to direct the application of the temporal resolution rule.

So far the theorem prover has been developed for PTL only, but extensions for first order logic are being investigated. Although *full* First-Order Temporal Logic is undecidable [Aba87], it is conjectured that there are useful subsets of First-Order Temporal Logic to which this form of temporal resolution can be successfully applied.

The need to combine rules in SNF together into SNF_m, in certain cases, to enable the expansion of a node, is the clearly the costly aspect of this algorithm. Although the need to combine rules into SNF_m means the complexity of the algorithm is therefore exponential, a detailed study of the complexity still needs to be carried out.

The temporal resolution method may also be used for verification of programs or systems represented as state-transitions, as SNF is particularly useful for describing these. The development of a *temporal* logic programming language may also be possible using this method, just as the logic programming language PROLOG has been developed from resolution in classical logics. In fact, the normal form required for the resolution procedure, SNF, is derived from that developed for use in METATEM [BFG+89], which is itself an executable temporal logic.

6.3 Conclusions

The work described in this paper has shown how graph-theoretic techniques can be applied to the implementation of a temporal resolution step, which has been incorporated into a resolution theorem prover. This theorem prover has been used to determine the unsatisfiability of temporal formulae. The method is more widely applicable than the other resolution methods, some of which have been mentioned in §6.1, as it deals with past-time, future-time and a wider range of temporal operators directly and has been automated successfully.

A comparison of the resolution theorem prover and DP, [Gou84], an automated decision procedure for temporal logics based on the tableau method has been carried out. For the results of the comparison and further details of the full theorem-prover, see [Dix92].

References

[Aba87] M. Abadi. *Temporal-Logic Theorem Proving*. PhD thesis, Department of Computer Science, Stanford University, March 1987.

[AM90] M. Abadi and Z. Manna. Nonclausal Deduction in First-Order Temporal Logic. *ACM Journal*, 37(2):279–317, April 1990.

[BFG+89] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: A Framework for Programming in Temporal Logic. In *Proceedings of REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, Mook, Netherlands, June 1989.

[CdC84] A. Cavali and L. Fariñas del Cerro. A Decision Method for Linear Temporal Logic. In R. E. Shostak, editor, *Proceedings of the 7th International Conference on Automated Deduction*, pages 113–127. LNCS 170, 1984.

[CL73] C-L. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[CW91] M. Carlsson and J. Widen. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, Kista, Sweden, September 1991.

[Dix92] C. Dixon. A graph-based approach to resolution in temporal logic. Master's thesis, Department of Computer Science, University of Manchester, Oxford Road, Manchester, December 1992.

[EdC89] P. Enjalbert and L. Fariñas del Cerro. Modal Resolution in Clausal Form. *Theoretical Computer Science*, 65:1–33, 1989.

[Fis91] M. Fisher. A Resolution Method for Temporal Logic. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, Sydney, Australia, August 1991. Morgan Kaufman.

[Fis92] M. Fisher. A Normal Form for First-Order Temporal Formulae. In *Proceedings of Eleventh International Conference on Automated Deduction (CADE)*, Saratoga Springs, New York, June 1992.

[FN92] M. Fisher and P. Noël. 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., February 1992.

[Gou84] G. D. Gough. Decision Procedures for Temporal Logic. Master's thesis, Department of Computer Science, University of Manchester, October 1984.

[Lov78] D. Loveland. *Automated Theorem Proving: a Logical Basis*. North-Holland, Inc., 1978.

[Min90] G. Mints. Gentzen-Type Systems and Resolution Rules, Part I: Propositional Logic. *Lecture Notes in Computer Science*, 417:198–231, 1990.

[Ohl88] H-J. Ohlbach. A Resolution Calculus for Modal Logics. *Lecture Notes in Computer Science*, 310:500–516, May 1988.

[Pei94] Martin Peim. Propositional Temporal Resolution Over Labelled Transition Systems. (Unpublished Technical Note), 1994.

[SV89] S. Safra and M. Y. Vardi. On ω-Automata and Temporal Logic. In *STOC*, pages 127–137, Seattle, Washington, May 1989. ACM.

[Ven86] G. Venkatesh. A Decision Method for Temporal Logic based on Resolution. *Lecture Notes in Computer Science*, 206:272–289, 1986.

[VW86] M. Vardi and P. Wolper. Automata-theoretic Techniques for Modal Logics of Programs. *Journal of Computer and System Sciences*, 32(2):183–219, April 1986.

[Wol83] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56, 1983.

[WOLB84] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning – Introduction and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

This article was processed using the LaTeX macro package with LLNCS style