Temporal Resolution: Removing Irrelevant Information*

Clare Dixon
Department of Computing
Manchester Metropolitan University
Manchester M1 5GD, United Kingdom
C.Dixon@doc.mmu.ac.uk

Abstract

The generation of too much information prohibits efficient resolution proof search in classical logics. Hence subsumption is used to discard redundant information and strategies have been developed to guide the proof search avoiding irrelevant information. The extension of the resolution method to temporal logics, for example that by Fisher for propositional linear-time temporal logics, further magnifies this problem. Here we provide an algorithm to efficiently remove irrelevant information prior to the application of Fisher's temporal resolution rule, show it retains the completeness of the temporal resolution system and demonstrate its efficiency on a set of examples.

1 Introduction

Resolution [20] is a decision procedure for classical propositional and first-order logic claimed to be *machine oriented*. However, the proof of theorems may be slowed or even not finish due to the generation of too much information [28]. This information could either be *irrelevant*, i.e. when a clause is produced that is not necessary for the proof or *redundant*, i.e. when the information is already in the database [26]. In classical logics, to overcome such problems subsumption is used to remove redundant information while strategies have been suggested to guide resolution based proofs avoiding irrelevant information. The *set of support strategy* [28] for example, restricts the application of the resolution rule so that at least one of the parent clauses must be from or derived from the set of support. *Unit resolution* [27] only allows resolution inferences to take place if at least one of the parent clauses is a unit clause. However, the imposition of such strategies may compromise the completeness of the resolution system, the ideal strategy being efficient while maintaining completeness. For example the set of support strategy is efficient as it restricts the number of resolution inferences that can be made and completeness can be maintained if the set of support is chosen so the conjunction of clauses *not* in the set of support is satisfiable. Likewise, given sets of Horn Clauses, unit resolution is a strategy that is both efficient and complete.

Since the resolution principle was first suggested, extensions to more exotic logics such as temporal logics [15] have been proposed [1, 5, 23]. Temporal logics have been used for the specification and verification of properties of concurrent systems, see for example [2, 12, 14, 18]. Proof methods have also been developed for these logics based on tableau [11, 24], automata [22, 25] and translation [17]. Resolution based methods have the advantage that they are not limited to finite state problems and, as for resolution in classical logics, strategies can be used to reduce the search space. The complexity of such logics is a barrier to efficient proof, for example the complexity of satisfiability of PTL, the logic used in this paper, is PSPACE-complete [21]. Again, this motivates the need for strategies to guide proof search, hopefully avoiding the worst case complexity in many situations.

In this paper we examine the resolution method devised by Fisher [10] for propositional linear-time temporal logic and we describe a way to reduce the amount of information prior to the application of the temporal resolution rule. Fisher's temporal resolution system depends on three main parts; the translation to a normal form; classical style resolution of formulae that occur within the same state; and temporal resolution over states, of formulae such as $\Box p$ (always p) and $\Diamond \neg p$ (eventually $\neg p$). Fisher's method has been shown complete [19] and, as it has just one temporal resolution rule, is particularly suited to mechanization. Unfortunately, even for small problems a large amount of information is generated, motivating the use of an algorithm to remove information irrelevant to temporal resolution.

In the rest of the paper we describe propositional

*This work was supported partially by an EPSRC PhD Studentship and partially by EPSRC Research Grant GR/K57282

temporal logic (PTL), §2, and explain Fisher's resolution system in more detail, §3. We present an algorithm to remove irrelevant information before the application of the temporal resolution rule and mention completeness and complexity issues in §4. In §5 we give experimental results showing in most cases a reduction in the time for this part of the procedure. Finally in §6 we present our conclusions.

2 A Linear temporal logic

Here we summarise the syntax and semantics of the logic used and describe the normal form required for the resolution method.

2.1 Syntax and semantics

The logic used in this paper is Propositional Temporal Logic (PTL), in which we use 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, we define an operator **start** in terms of the last-time operator '$\odot$'

$$\textbf{start} \equiv \neg \odot \textbf{true}$$

which only holds at the beginning of time.

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

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

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 semantics of the temporal connectives used in the normal form or the resolution rule are defined as follows

$(\sigma, i) \models \textbf{start}$ iff $i = 0$;
$(\sigma, i) \models \odot A$ iff $i > 0$ and $(\sigma, i-1) \models A$;
$(\sigma, i) \models \Diamond A$ iff there exists a $j \geqslant i$ s.t. $(\sigma, j) \models A$;
$(\sigma, i) \models \Box A$ iff for all $j \geqslant i$ then $(\sigma, j) \models A$;
$(\sigma, i) \models A \mathcal{U} B$ iff there exists a $k \geqslant i$ s.t. $(\sigma, k) \models B$ and for all $i \leqslant j < k$ then $(\sigma, j) \models A$;
$(\sigma, i) \models A \mathcal{W} B$ iff $(\sigma, i) \models A \mathcal{U} B$ or $(\sigma, i) \models \Box A$.

The full syntax and semantics of PTL will not be presented here, but can be found in [10].

2.2 A normal form for PTL

Formulae in PTL can be transformed to a normal form, Separated Normal Form (SNF) [10], which is the basis of the resolution method used in this paper. 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

$$\Box \bigwedge_i R_i$$

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

$$\begin{array}{lcll}
\textbf{start} & \Rightarrow & \bigvee_{b=1}^{r} l_b & \text{(an \textit{initial} } \Box\text{–rule)} \\
\odot \bigwedge_{a=1}^{g} k_a & \Rightarrow & \bigvee_{b=1}^{r} l_b & \text{(a \textit{global} } \Box\text{–rule)} \\
\textbf{start} & \Rightarrow & \Diamond l & \text{(an \textit{initial} } \Diamond\text{–rule)} \\
\odot \bigwedge_{a=1}^{g} k_a & \Rightarrow & \Diamond l & \text{(a \textit{global} } \Diamond\text{–rule)}
\end{array}$$

Here k_a, l_b, and l are literals. The outer '$\Box$' operator, that surrounds the conjunction of rules is usually omitted. Similarly, for convenience the conjunction is dropped and we consider just the set of rules R_i.

We note a variant on SNF called merged-SNF (SNF_m) [10] used for combining rules by applying the following transformation.

$$\begin{array}{rcl}
\odot A & \Rightarrow & F \\
\odot B & \Rightarrow & G \\
\hline
\odot (A \wedge B) & \Rightarrow & F \wedge G
\end{array}$$

The right hand side of the rule generated may have to be further translated into Disjunctive Normal Form (DNF), if either F or G are disjunctive, to maintain the general SNF rule structure.

3 The resolution procedure

Here we present a review of the temporal resolution method [10]. The clausal temporal resolution method consists of repeated applications of both 'step' and 'temporal' resolution on sets of formulae in SNF, together with various simplification steps. The step and temporal resolution rules are applied until either a contradiction has been detected or no new resolvents can be generated. Completeness of the resolution procedure has been shown in [19].

3.1 Step resolution

'Step' 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. Simplification and subsumption rules are also applied.

Pairs of initial $\Box$–rules, or global $\Box$–rules, may be resolved using the following (step resolution) rule where $\mathcal{L}_1$ and $\mathcal{L}_2$ are both last-time formulae or both **start**.

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

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

$$\begin{array}{rcl} \circledbullet P & \Rightarrow & \mathbf{false} \\ \hline \mathbf{start} & \Rightarrow & \neg P \\ \circledbullet \mathbf{true} & \Rightarrow & \neg P \end{array}$$

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 constraints therefore represent $\Box\neg P$

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

$$\begin{array}{rcl} \mathbf{start} & \Rightarrow & \mathbf{false} \\ \circledbullet \mathbf{true} & \Rightarrow & \mathbf{false} \end{array}$$

3.2 Temporal resolution

During temporal resolution the aim is to resolve a $\Diamond$–rule, $\mathcal{L} \Rightarrow \Diamond l$, where $\mathcal{L}$ may be either a last-time formula or **start**, with a set of rules that together imply $\Box\neg l$, for example a set of rules that together have the effect of $\circledbullet A \Rightarrow \Box\neg l$. However the interaction between the '$\bigcirc$' and '$\Box$' operators in PTL makes the definition of such a rule non-trivial and further the translation from PTL to SNF will have removed all but the outer level of $\Box$–operators. So, resolution will be between a $\Diamond$–rule and a *set* of rules that together imply an $\Box$–formula which will contradict the $\Diamond$–rule. Thus, given a set of rules in SNF, then for every rule of the form $\mathcal{L} \Rightarrow \Diamond l$ temporal resolution may be applied between this $\Diamond$–rule and a set of global $\Box$–rules, which taken together force $\neg l$ always to be satisfied.

The temporal resolution rule is given by the following

$$\begin{array}{rcl} \circledbullet A_0 & \Rightarrow & B_0 \\ \dots & & \dots \\ \circledbullet A_n & \Rightarrow & B_n \\ \mathcal{L} & \Rightarrow & \Diamond l \\ \hline \mathcal{L} & \Rightarrow & (\bigwedge_{i=0}^{n} \neg A_i)\, \mathcal{W}\, l^1 \end{array}$$

with side conditions

$$\left\{ \begin{array}{rcl} \text{for all } 0 \leq i \leq n & \vdash & B_i \Rightarrow \neg l \\ \text{and} & \vdash & B_i \Rightarrow \bigvee_{j=0}^{n} A_j \end{array} \right\}$$

where the side conditions ensure that the set of rules $\circledbullet A_i \Rightarrow B_i$ together imply $\Box\neg l$. So if any of the A_i are satisfied then $\neg l$ will *always* be satisfied, i.e.,

$$\circledbullet \bigvee_{i=0}^{n} A_i \Rightarrow \Box\neg l.$$

Such a set of rules are known as a *loop* in $\neg l$.

4 Reducing the rule-set

As the application of the temporal resolution is the most expensive part of the algorithm it is on this we concentrate. Algorithms for applying the temporal resolution rule are given and compared in [7]. We note that even for small problems, such as the list of valid temporal formulae found in [14], large sets of rules are generated making the search for loops costly.

In this section we describe an algorithm that removes rules, prior to the application of one of the loop search algorithms without affecting the completeness of the overall method. This is known as *rule reduction*. The motivation being that the time taken for rule reduction will be offset by the reduced time required to perform loop search.

[1]In previous presentations of this rule two resolvents are given. However only the resolvent given here is necessary for completeness, so for simplicity we omit the other resolvent.

4.1 Overview

We begin by considering a loop in literal $\neg l$ and examine the cases when the deletion of a rule does not affect the loop remaining. An example of a loop for resolution with the eventuality $\Diamond l$ is L

$$\begin{array}{rcl} \bigcirc(a \wedge c) & \Rightarrow & a \wedge \neg l \\ \bigcirc(a \wedge \neg c) & \Rightarrow & a \wedge \neg l \end{array}$$

formed from the rules R

$$\begin{array}{rcl} \bigcirc(a \wedge c) & \Rightarrow & a \\ \bigcirc a & \Rightarrow & \neg l \\ \bigcirc(a \wedge \neg c) & \Rightarrow & a. \end{array}$$

To get the required "looping" the literal a appears on both the left and right hand sides of rules in R and the proposition c and its negation both occur on the left hand sides of rules in R[2]. Finally $\neg l$, the literal in which we are looking for a loop, *should* occur on the right hand side of rules and *may* occur on the left but its negation, l, should not occur at all.

The idea behind the algorithm is that, given a set of global $\Box$-rules we want to be able to delete rules that we know will not form part of the loop before actually detecting the loop itself. For example if we are resolving with $\Diamond\neg l$, given the rule $\bigcirc x \Rightarrow a$ and the rule-set R above, as there are no rules with $\neg x$ on the left hand side or x on the right hand side we can ignore this rule for the purposes of loop search. The deletion of the rule $\bigcirc(a \wedge c) \Rightarrow a$ however from rule-set R, means that no loop can be detected in the remaining rules. The deletion of a rule crucial to the loop is avoided by noting that there is a rule in R with $\neg c$ on the left hand side to correspond with the c on the left side of this rule (or a c on the right hand side). Similarly the deletion of $\bigcirc a \Rightarrow \neg l$ means that no loop can be detected from the set of rules remaining in R. To avoid this we allow $\neg l$ (the literal in which we are looking for a loop) to appear anywhere and again make sure that such a rule is only deleted if we can't find an a on the right hand side or $\neg a$ on the left hand side of any rule in R.

4.2 Definitions

We assume that rules are in their simplest forms, i.e. all negations are pushed through to propositions and simplified and that all step resolution possible has been carried out. A positive proposition is a proposition not in the scope of any negations and a negative proposition is a proposition in the scope of one negation. For example in the rule

$$\bigcirc(a \wedge \neg b) \Rightarrow \neg c \vee d$$

a and d are positive propositions and b and c are negative propositions.

Let $\mathfrak{L}^+$,$\mathfrak{L}^-$,$\mathfrak{R}^+$ and $\mathfrak{R}^-$ be multi-sets of propositions defined as follows. Throughout, we only consider global $\Box$-rules.

$\mathfrak{L}^+$ is the multi-set of propositions that occur positively on the left hand side of the set of rules.

$\mathfrak{L}^-$ is the multi-set of propositions that occur negatively on the left hand side of the set of rules.

$\mathfrak{R}^+$ is the multi-set of propositions that occur positively on the right hand side of the set of rules.

$\mathfrak{R}^-$ is the multi-set of propositions that occur negatively on the right hand side of the set of rules.

4.3 The rule reduction algorithm

Let us assume we are trying to resolve with

$$\mathcal{L} \Rightarrow \Diamond l$$

and the set of global-$\Box$ rules in SNF having performed all the step resolution possible is R.

1. If l occurs in a rule on the right or left hand side delete the rule.

2. Create the empty multi-sets $\mathfrak{L}^+$,$\mathfrak{L}^-$,$\mathfrak{R}^+$ and $\mathfrak{R}^-$.

3. For each rule $r \in R$ in turn, add a proposition to the relevant multi-set for each proposition occurring in r as follows.

   (a) If p is a proposition occuring positively on the left hand side of r then add p to $\mathfrak{L}^+$.

   (b) If p is a proposition occuring negatively on the left hand side of r then add p to $\mathfrak{L}^-$.

   (c) If p is a proposition occuring positively on the right hand side of r then add p to $\mathfrak{R}^+$.

   (d) If p is a proposition occuring negatively on the right hand side of r then add p to $\mathfrak{R}^-$.

4. Remove the proposition l from any of the multi-sets $\mathfrak{L}^+$, $\mathfrak{L}^-$,$\mathfrak{R}^+$ or $\mathfrak{R}^-$.

5. Remove rules as follows.

[2] We could actually perform resolution on the left hand side of two of these rules to produce a rule $\bigcirc a \Rightarrow a$ and thus an equivalent loop $\bigcirc a \Rightarrow a \wedge \neg l$ but in general doing this in practice produces a much larger rule-set.

(a) If p is in $\mathfrak{L}^+$ and neither $\mathfrak{L}^-$ or $\mathfrak{R}^+$ have p as a member then delete all rules with p occurring positively on the left hand side, delete all occurrences of p from $\mathfrak{L}^+$ and delete a copy of the other propositions occurring in these rules in their respective multi-sets.

(b) If p is in $\mathfrak{L}^-$ and neither $\mathfrak{L}^+$ or $\mathfrak{R}^-$ have p as a member delete all rules with p occurring negatively on the left hand side, delete all occurrences of p from $\mathfrak{L}^-$ and delete a copy of the other propositions occurring in these rules in their respective multi-sets.

(c) If p is in $\mathfrak{R}^+$ and p is not a member of $\mathfrak{L}^+$ then delete all rules with p occurring positively on the right hand side, delete all occurrences of p from $\mathfrak{R}^+$ and delete a copy of the other propositions occurring in these rules in their respective multi-sets.

(d) If p is in $\mathfrak{R}^-$ and p is not a member of $\mathfrak{L}^-$ then delete all rules with p occurring negatively on the right hand side, delete all occurrences of p from $\mathfrak{R}^-$ and delete a copy of the other propositions occurring in these rules in their respective multi-sets.

6. Repeat step 5 until no more rules can be deleted.

We delete any rules containing l (step 1) as we are looking for a loop in $\neg l$. Recall that we assume that all the step resolution possible has been carried out. The right hand side of each SNF_m rule in the loop implies $\neg l$ so l may not appear on the right hand side of a rule that is combined to make this loop. The proposition l is removed from the multi-sets (step 4) as we are looking for a loop in $\neg l$. Given the rules $\bigcirc a \Rightarrow a$, $\bigcirc a \Rightarrow \neg l$ that together form the loop $\bigcirc a \Rightarrow a \wedge \neg l$, we don't want l to occur only in the multi-set $\mathfrak{R}^-$ and thus delete the latter rule. Further it does not matter if $\neg l$ happens to appear in some rules on the left hand side as we know the right hand side of each rule implies $\neg l$ anyway.

We use multi-sets and adjust these sets when a rule is deleted to avoid having to re-calculate $\mathfrak{L}^+$, $\mathfrak{L}^-$,$\mathfrak{R}^+$ and $\mathfrak{R}^-$ after every deletion. For example assume we are looking for a loop in $\neg l$ from the rules

$$\begin{array}{rcl} \bigcirc(c \wedge a) & \Rightarrow & a \\ \bigcirc a & \Rightarrow & \neg l \\ \bigcirc b & \Rightarrow & c. \end{array}$$

If we construct sets rather than multi-sets, having deleted l from the sets, we obtain $\mathfrak{L}^+ = \{a, b, c\}$ and $\mathfrak{R}^+ = \{a, c\}$ and $\mathfrak{L}^- = \mathfrak{R}^- = \emptyset$. As $b \in \mathfrak{L}^+$ but $b \notin \mathfrak{R}^+$ and $b \notin \mathfrak{L}^-$ all the rules with b in a left hand side positive position are deleted and $\mathfrak{L}^+$ becomes $\{a, c\}$. We cannot delete the c from $\mathfrak{R}^+$ as there may be another rule with c in this position. Now we must recalculate the four sets and can continue the deletions until no rules remain.

4.4 Example

Assume we are resolving with $\mathcal{L} \Rightarrow \Diamond l$ and we have the following global $\Box$-rules.

$$\begin{array}{rrcl} 1. & \bigcirc a & \Rightarrow & w \\ 2. & \bigcirc b & \Rightarrow & \neg w \\ 3. & \bigcirc \mathbf{true} & \Rightarrow & \neg a \vee \neg b \\ 4. & \bigcirc(c \wedge \neg a) & \Rightarrow & c \\ 5. & \bigcirc(c \wedge \neg b) & \Rightarrow & c \\ 6. & \bigcirc c & \Rightarrow & \neg l \end{array}$$

Firstly we build the multi-sets $\mathfrak{L}^+$, $\mathfrak{L}^-$, $\mathfrak{R}^+$, and $\mathfrak{R}^-$ taking each rule in turn. In rule 1, a occurs positively on the left hand side so we add a to the multi-set $\mathfrak{L}^+$ and w occurs positively on the right hand side so we add w to $\mathfrak{R}^+$. The four multi-sets are $\mathfrak{L}^+ = \{a\}$, $\mathfrak{L}^- = \{\}$, $\mathfrak{R}^+ = \{w\}$, and $\mathfrak{R}^- = \{\}$. In rule 2, b occurs positively on the left hand side so we add b to the multi-set $\mathfrak{L}^+$ and w occurs negatively on the right hand side so we add w to $\mathfrak{R}^-$. The four multi-sets are $\mathfrak{L}^+ = \{a, b\}$, $\mathfrak{L}^- = \{\}$, $\mathfrak{R}^+ = \{w\}$, and $\mathfrak{R}^- = \{w\}$. We continue as above until we have processed all the rules and, having deleted any occurrences of l, we obtain the four multi-sets $\mathfrak{L}^+ = \{a, b, c, c, c\}$, $\mathfrak{L}^- = \{a, b\}$, $\mathfrak{R}^+ = \{w, c, c\}$, and $\mathfrak{R}^- = \{w, a, b\}$. Each multi-set contains propositions of a particular type. For example the $\mathfrak{R}^+$ multi-set tells us that the proposition w occurs positively on the right hand side of a rule (rule 1) and that c occurs positively on the right hand side of two rules (rules 4 and 5).

When carrying out deletions, starting with $\mathfrak{L}^+$ we see a and b are both in $\mathfrak{L}^-$ and c is in $\mathfrak{R}^+$. Similarly for $\mathfrak{L}^-$ a and b are both in $\mathfrak{L}^+$. Looking at $\mathfrak{R}^+$ the proposition c occurs in $\mathfrak{L}^+$ but w does not occur in $\mathfrak{L}^+$ so we need to remove all the rules with w in the $\mathfrak{R}^+$ position and adjust all the other multi-sets. That is, the algorithm tells us that any rule with a positive w on the right hand side can be deleted as it cannot form part of the loop. We see rule 1 is the only rule containing w in a $\mathfrak{R}^+$ position so remove this rule, delete w from $\mathfrak{R}^+$ and delete an a, that occurs on the left hand side of rule 1, from $\mathfrak{L}^+$. The rules remaining in our set are rules 2–6 and our four multi-sets are $\mathfrak{L}^+ = \{b, c, c, c\}$, $\mathfrak{L}^- = \{a, b\}$, $\mathfrak{R}^+ = \{c, c\}$, and $\mathfrak{R}^- = \{w, a, b\}$. So, having deleted a rule, propositions may need to be deleted from other multi-sets so that they accurately represent the current set of rules. Then the algorithm can be re-applied and more rules may be deleted.

Looking at $\mathfrak{R}^-$ w doesn't occur in $\mathfrak{L}^-$ so we delete any rules with w in a right hand side negative position and delete a copy of any other propositions in these rules. Looking at the rule-set rule 2 is the only rule with $\neg w$ on the right hand side so we delete this rule leaving rules 3–6, and delete a copy of b from $\mathfrak{L}^+$. The multi-sets of propositions become $\mathfrak{L}^+ = \{c, c, c\}$, $\mathfrak{L}^- = \{a, b\}$, $\mathfrak{R}^+ = \{c, c\}$, and $\mathfrak{R}^- = \{a, b\}$.

We cannot delete any more rules so the reduced rule-set becomes rules 3–6. We may now use a loop finding algorithm to detect any loops. In this example the loop, from merging rules 3, 4 and 6 and rules 3, 5 and 6, is

$$\begin{array}{rcl} \odot(c \wedge \neg a) & \Rightarrow & (c \wedge \neg a \wedge \neg l) \vee (c \wedge \neg b \wedge \neg l) \\ \odot(c \wedge \neg b) & \Rightarrow & (c \wedge \neg a \wedge \neg l) \vee (c \wedge \neg b \wedge \neg l) \end{array}$$

4.5 Completeness and complexity

Completeness is shown by proving that any resolvents from the application of the temporal resolution rule can be generated by applying the rule reduction algorithm and then performing temporal resolution and some step resolution. For more details see [6].

The complexity of the algorithm will be considered in the full paper.

5 Results

A prototype implementation performing the temporal resolution method has been built. The programs are written in SICStus Prolog [4] running under UNIX and timings have been carried out on a SPARCstation 1 using compiled Prolog code. The test data is a set of valid temporal formulae taken from [14] chosen as it provides a reasonably sized collection of small problems to be proved valid. An example of the type of formulae being shown valid (we actually show the negation is unsatisfiable) is

$$\Diamond w_1 \wedge \Diamond w_2 \Rightarrow \Diamond(w_1 \wedge \Diamond w_2) \vee \Diamond(w_2 \wedge \Diamond w_1).$$

Due to space restrictions the complete set of results is omitted here but can be found in [6], however a summary is given in Table 1. Table 1 gives the number of eventualities where the time for rule reduction followed by loop finding on the resulting rule-set is less than or equal to, or greater than the original. The figures in brackets are the percentages for these values. The first row of the table gives the figures for the full data set, subsequent rows take subsets of the data. The second row contains the data for all examples where the original rule-set contains more than 10 rules matching the data shown. The third row contains data for all examples where at least one of the timings is greater than 60 milliseconds. The reason we have chosen to isolate these subsets is that for the former we expect rule reduction to be particularly beneficial where the examples are not trivially small, and the latter to exclude any timing inaccuracies that may occur for timings that are very small.

Subset of Data	Reduction $\leqslant$ Original	Reduction $>$ Original
(i) Complete data set	43 (83%)	9 (17%)
(ii) No. rules > 10	28 (93%)	2 (7%)
(iii) Time > 60	23 (92%)	2 (8%)

Table 1 Summary of Results from Rule Reduction

The loop search used in each case is Breadth-First Search [8] however another method could be used if desired, see for example those described in [7].

The timings of the loop searches are a trade off between the overhead required to remove rules from the rule-set plus the loop finding times on a potentially smaller data set, against the time for finding the loop in the full rule-set. The timings we have obtained suggest that rule reduction does help on larger, more difficult rule-sets.

We note it is the same two examples where the time for rule reduction and loop finding is greater than the time for loop finding alone in the subsets of the data labelled (ii) and (iii). The times for these examples for the former are not larger than 30% more than the timings of the latter (in fact the actual figures are 14% and 29%) whereas the gain from using rule reduction on some examples is much greater. For example in one case the time for performing loop search alone is $3\frac{1}{2}$ times worse than for finding the loop with rule reduction.

6 Conclusions and further work

6.1 Conclusions

We have given an algorithm for removing rules irrelevant to the application of Fisher's temporal resolution rule and in more than 80% of the examples tried the times for detecting loops are the same or better than without using this algorithm. If examples with trivially small numbers of rules are ignored then this figure increases to more than 90% of the examples tried.

The algorithm is independent of the type of loop search algorithm being used so an implementation could apply the rule reduction strategy whatever loop search algorithm was chosen. Rule reduction will be most effective in loops that are small in comparison to the size of the rule-set or involve literals that don't often appear in the rest of the rule-set. However, if the loop consists of most of the rule-set or contains literals that appear frequently throughout the rule-set then rule reduction will not be as effective.

Note that we cannot carry out the loop search just by using the sets of propositions we have constructed as we have lost information relating to individual rules which is necessary to ensure that the conditions for a loop are satisfied.

6.2 Further work

Although the rule reduction algorithm seems to help us on the small problems we have tested it would be useful to try the algorithm on larger examples. For example it would be interesting to see whether the rule reduction algorithm helps tackle problems that are too large for the original theorem prover.

We have considered parallelising the basic temporal resolution method and several loop search algorithms in [9]. Likewise the rule reduction algorithm could also be parallelised. Firstly the construction of the sets of propositions $\mathfrak{L}^+$ etc. could be carried out in parallel by splitting the set of rules into n-sets given n processors and constructing the sets of propositions for each subset. Then the totals from each processor must be merged. Finally, the deletion of rules could also be carried out in parallel although care must be taken to ensure that sets of propositions are maintained correctly.

6.3 Related work

Decision procedures based on resolution have been developed for linear-time temporal logics in [1, 5, 23], however in many cases they are unsuitable for implementation either because they only deal with a small number of the temporal operators or because of problems with proof direction due to the large numbers of resolution rules that may be applied.

Subsumption is used and strategies developed to assist resolution theorem provers for classical logics, see for example OTTER [16]. For resolution in temporal logics although subsumption rules are generally used to keep the rule-set as compact as possible the author knows of no strategies developed to guide temporal resolution proofs.

Theorem provers for temporal logic that have been implemented tend to be tableau based, for example DP [11] and a module that forms part of the Logic Work Bench (LWB) [13]. Model checking systems are also available for example the STeP system [3] which combines both model checking and deductive methods.

Acknowledgements

Thanks to Howard Barringer and Michael Fisher for their guidance and encouragement during this work and to Graham Gough and Martin Peim for many helpful comments and advice. Thanks again to Michael Fisher for all the suggestions made relating to an earlier draft of this paper.

References

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

[2] H. Barringer. Using Temporal Logic in the Compositional Specification of Concurrent Systems. In A. P. Galton, editor, *Temporal Logics and their Applications*, chapter 2, pages 53–90. Academic Press Inc. Limited, London, Dec. 1987.

[3] N. Bjorner, A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H. B. Sipma, and T. E. Uribe. *STeP The Stanford Temporal Prover Educational Release Version 1.0 User's Manual*. Computer Science Department, Stanford University, California 94305, Nov. 1995.

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

[5] A. Cavalli 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*, volume 170 of *Lecture Notes in Computer Science*, pages 113–127. Springer-Verlag, 1984.

[6] C. Dixon. *Strategies for Temporal Resolution*. PhD thesis, Department of Computer Science, University of Manchester, 1995.

[7] C. Dixon. Search Strategies for Resolution in Temporal Logics. In M. A. McRobbie and J. K. Slaney, editors, *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 672–687, New Brunswick, New Jersey, July/August 1996. Springer-Verlag.

[8] C. Dixon. Temporal Resolution: A Breadth-First Search Approach. In *Proceedings of TIME-96 the Third International Workshop on Temporal Representation and Reasoning*, Key West, Florida, May 1996.

[9] C. Dixon, M. Fisher, and R. Johnson. Parallel Temporal Resolution. In *Proceedings of TIME-95 the Second International Workshop on Temporal Representation and Reasoning*, Melbourne Beach, Florida, Apr. 1995.

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

[11] G. D. Gough. Decision Procedures for Temporal Logic. Master's thesis, Department of Computer Science, University of Manchester, October 1984. Also University of Manchester, Department of Computer Science, Technical Report UMCS-89-10-1.

[12] B. T. Hailpern. *Verifying Concurrent Processes Using Temporal Logic*, volume 129 of *Lecture Notes in Computer Science*. Springer-Verlag, 1982.

[13] G. Jaeger, A. Heuerding, S. Schwendimann, F. Achermann, P. Balsiger, P. Brambilla, H. Zimmermann, M. Bianchi, K. Guggisberg, and W. Heinle. LWB–The Logics Workbench 1.0. http://lwbwww.unibe.ch:8080/LWBinfo.html. University of Berne, Switzerland.

[14] Z. Manna and A. Pnueli. Verification of Concurrent Programs: The Temporal Framework. In R. S. Boyer and J. S. Moore, editors, *The Correctness Problem in Computer Science*, pages 215–273. Academic Press, London, 1981.

[15] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.

[16] W. W. McCune. *OTTER 2.0 Users Guide*. Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439-4801, March 1990. ANL-90/9.

[17] A. Nonnengart. Resolution-Based Calculi for Modal and Temporal Logics. In M. A. McRobbie and J. K. Slaney, editors, *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 598–612, New Brunswick, New Jersey, July/August 1996. Springer-Verlag.

[18] S. Owicki and L. Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, July 1982.

[19] M. Peim. Propositional Temporal Resolution Over Labelled Transition Systems. Unpublished Technical Note, Department of Computer Science, University of Manchester, 1994.

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

[21] A. P. Sistla and E. M. Clarke. Complexity of Propositional Linear Temporal Logics. *ACM Journal*, 32(3):733–749, July 1985.

[22] M. Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proceedings IEEE Symposium on Logic in Computer Science*, pages 332–344, Cambridge, 1986.

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

[24] P. Wolper. The Tableau Method for Temporal Logic: An overview. *Logique et Analyse*, 110–111:119–136, June-Sept 1985.

[25] P. Wolper, M. Vardi, and A. P. Sistla. Reasoning about Infinite Computation Paths. In *Proceedings of the Twentyfourth Symposium on the Foundations of Computer Science*. IEEE, 1983.

[26] L. Wos. *Automated Reasoning : 33 Basic Research Problems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[27] L. Wos, D. Carson, and G. Robinson. The Unit Preference Strategy in Theorem Proving. In *Proceedings of AFIPS Fall Joint Computer Conference*, pages 615–621. Thompson Book Company, 1964.

[28] L. Wos, G. Robinson, and D. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *ACM Journal*, 12:536–541, Oct. 1965.