Removing Irrelevant Information in Temporal Resolution Proofs

Clare Dixon

Department of Computing and Mathematics

Manchester Metropolitan University

Manchester M1 5GD, United Kingdom

EMAIL: C.Dixon@doc.mmu.ac.uk

TEL: (+44) 161 247 1549

FAX: (+44) 161 247 1483

Short Title: **Removing Irrelevant Information in Temporal Proofs**

Abstract

The generation of too much information prohibits efficient resolution proof search in classical logics. 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, further magnifies this problem. Here we develop an algorithm for the removal of irrelevant information during resolution proofs for a particular resolution proof system for propositional linear-time temporal logic. The resolution system operates on formulae in a normal form. Following a phase of classical style resolution, temporal resolution is carried out by detecting sets of formulae that together represent an $\Box$-formula to be resolved with a $\Diamond$-formula. It is before the application of the temporal resolution rule that the new algorithm is applied to efficiently remove irrelevant information. The completeness of the new algorithm is discussed and its complexity considered. Its practical efficiency is demonstrated by giving timings for a set of examples.

1 Introduction

Resolution (Robinson, 1965) 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 (Wos et al., 1965). 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 (Wos, 1988). 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* (Wos et al., 1965) 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* (Wos et al., 1964) 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 (Manna and Pnueli, 1992; Manna and Pnueli, 1995) have been proposed (Abadi and Manna, 1990; Cavalli and Fariñas del Cerro, 1984; Venkatesh, 1986). Temporal logics have been used for the specification and verification of properties of concurrent systems, see for example (Barringer, 1987; Hailpern, 1982; Manna and Pnueli, 1981; Owicki and Lamport, 1982). Proof methods have also been developed for these logics based on tableau (Gough, 1984; Wolper, 1985), automata (Vardi and Wolper, 1986; Wolper et al., 1983) and translation (Nonnengart, 1996). 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 (Sistla and Clarke, 1985). 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 (Fisher, 1991) 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 $\square p$ (always p) and $\diamondsuit \neg p$ (eventually $\neg p$). Fisher's method has been shown complete (Peim, 1994) 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 temporal logic (PTL), §2, and explain Fisher's resolution system in more detail, §3. In §4 we develop an algorithm to remove irrelevant information before the application of the temporal resolution rule. The completeness of the algorithm and its complexity are discussed in §5. In §6 we give experimental results showing in most cases a reduction in the time for this part of the procedure. Finally in §7 we present our conclusions and consider future work.

2 Propositional Temporal Logic

Propositional Temporal Logic (PTL) is based on a linear, discrete model of time with finite past and infinite future (Gabbay et al., 1980). This logic may be viewed as a classical propositional logic augmented with both future-time and past-time temporal connectives (we also refer to temporal connectives as temporal operators). Future-time temporal connectives include '$\Diamond$' *sometime in the future*, '$\Box$' *always in the future*, '$\bigcirc$' *in the next moment in time*, '$\mathcal{U}$' *until*, and '$\mathcal{W}$' *unless (weak until)*, while past-time connectives include '$\blacklozenge$' *sometime in the past*, '$\blacksquare$' *always in the past*, '$\odot$' *in the previous moment in time (strong last)*, '$\bullet$' *in the previous moment in time (weak last)*, '$\mathcal{S}$' *since*, and '$\mathcal{Z}$' *zince (weak since)*. We also use an operator '**start**' meaning *at the beginning of time.* Note, when considering models of time with finite past allowing past-time temporal operators does not add expressiveness (Gabbay et al., 1980) but allows the expression of certain statements in a more natural way, see for example (Lichtenstein et al., 1985).

2.1 Syntax

PTL formulae are constructed using the following connectives and proposition symbols.

- A set $\mathcal{P}$ of propositional symbols.
- Propositional and temporal constants **true**, **false** and **start**.
- Propositional connectives, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, and $\Leftrightarrow$.
- Future-time temporal connectives, $\bigcirc$, $\Diamond$, $\Box$, $\mathcal{U}$, and $\mathcal{W}$.
- Past-time temporal connectives $\odot$, $\bullet$, $\blacklozenge$, $\blacksquare$, $\mathcal{S}$, and $\mathcal{Z}$.

The set of well-formed formulae of PTL, WFF, is inductively defined as the smallest set satisfying :

- Any element of $\mathcal{P}$ is in WFF
- **true**, **false** and **start** are in WFF
- If A and B are in WFF then so are

$$\begin{array}{llllll} \neg A & A \vee B & A \wedge B & A \Rightarrow B & A \Leftrightarrow B & \\ \Diamond A & \Box A & A \mathcal{U} B & A \mathcal{W} B & \bigcirc A & \\ \blacklozenge A & \blacksquare A & A \mathcal{S} B & A \mathcal{Z} B & \odot A & \bullet A \end{array}$$

A *literal* is defined as either a proposition or the negation of a proposition.

2.2 Semantics

PTL is interpreted over a discrete, linear model of time, for example the natural numbers. Models of PTL can be represented as a sequence of *states*

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

where each state, s_i, is a set of propositions, representing those propositions which are satisfied in the i^{th} moment in time. As formulae in PTL are interpreted at a particular state in the sequence (i.e. at a particular moment in time), the notation

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

denotes the truth (or otherwise) of formula A in the model σ at state index $i \in \mathbf{N}$. For any formula A, model σ and state index $i \in \mathbf{N}$, then either $(\sigma, i) \models A$ holds or $(\sigma, i) \models A$ does not hold, denoted by $(\sigma, i) \not\models A$. If there is some σ such that $(\sigma, 0) \models A$, then A is said to be *satisfiable*. If $(\sigma, 0) \models A$ for all models, σ, then A is said to be *valid* and is written $\models A$. Note that formulae here are interpreted at s_0. This is an alternative definition (known as anchored temporal logic, see (Manna and Pnueli, 1989)) to the one usually used in temporal logics, but is equally expressive. The semantics of a proposition is defined by the valuation given to it in the model

$$(\sigma, i) \models p \quad \text{iff} \quad p \in s_i \text{ where } p \in \mathcal{P} \text{ (the set of propositional symbols).}$$

The semantics of the standard propositional connectives are as in classical logic, for example

$$(\sigma, i) \models A \wedge B \quad \text{iff} \quad (\sigma, i) \models A \text{ and } (\sigma, i) \models B.$$

The semantics of a negated formula is defined as follows

$$(\sigma, i) \models \neg A \quad \text{iff} \quad (\sigma, i) \not\models A.$$

The semantics of the **start** connective is defined as follows

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

The semantics of the unary future-time temporal connectives are defined as follows

$$\begin{array}{lll}
(\sigma, i) \models \bigcirc A & \text{iff} & (\sigma, i+1) \models A; \\
(\sigma, i) \models \Diamond A & \text{iff} & \text{there exists a } j \geqslant i \text{ s.t. } (\sigma, j) \models A; \\
(\sigma, i) \models \Box A & \text{iff} & \text{for all } j \geqslant i \text{ then } (\sigma, j) \models A.
\end{array}$$

The semantics of the binary future-time temporal connectives are defined as follows

$$\begin{array}{lll}
(\sigma, i) \models A \,\mathcal{U}\, B & \text{iff} & \text{there exists a } k \geqslant i \text{ s.t. } (\sigma, k) \models B \text{ and for all } i \leqslant j < k \text{ then } (\sigma, j) \models A; \\
(\sigma, i) \models A \,\mathcal{W}\, B & \text{iff} & (\sigma, i) \models A \,\mathcal{U}\, B \text{ or } (\sigma, i) \models \Box A.
\end{array}$$

The semantics of the unary past-time temporal connectives are defined as follows

$$
\begin{array}{lll}
(\sigma, i) \models \circledcirc A & \text{iff} & i > 0 \text{ and } (\sigma, i-1) \models A; \\
(\sigma, i) \models \bullet A & \text{iff} & i = 0 \text{ or } (\sigma, i-1) \models A; \\
(\sigma, i) \models \blacklozenge A & \text{iff} & \text{there exists } h \text{ s.t. } 0 \leqslant h < i \text{ and } (\sigma, h) \models A; \\
(\sigma, i) \models \blacksquare A & \text{iff} & \text{for all } h \text{ s.t. } 0 \leqslant h < i \text{ then } (\sigma, h) \models A.
\end{array}
$$

The semantics of the binary past-time temporal connectives are defined as follows

$$
\begin{array}{lll}
(\sigma, i) \models A \mathcal{S} B & \text{iff} & \text{there exists } g \text{ s.t. } 0 \leqslant g < i \text{ and } (\sigma, g) \models B \text{ and for all } h \text{ s.t. } g < h < i \\
& & \text{then } (\sigma, h) \models A; \\
(\sigma, i) \models A \mathcal{Z} B & \text{iff} & (\sigma, i) \models A \mathcal{S} B \text{ or } (\sigma, i) \models \blacksquare A.
\end{array}
$$

We note that since our temporal models assume a finite past, the operator **start** has been introduced for convenience, to detect the beginning of time. It can be defined in terms of the last-time operator as follows

$$\mathbf{start} \equiv \neg \circledcirc \mathbf{true}.$$

An alternative definition of PTL is via a core set of operators $\neg$, $\wedge$, $\bigcirc$, $\mathcal{U}$, $\circledcirc$ and $\mathcal{S}$. The other operators are considered abbreviations (omitting the definitions for the remaining classical operators).

$$
\begin{array}{l}
\mathbf{true} \equiv p \vee \neg p, \text{ for some fixed atom } p \\
\mathbf{false} \equiv \neg \mathbf{true} \\
\Diamond A \equiv \mathbf{true}\, \mathcal{U}\, A \\
\Box A \equiv \neg(\mathbf{true}\, \mathcal{U}\, (\neg A)) \\
A \mathcal{W} B \equiv (A \mathcal{U} B) \vee (\Box A) \\
\bullet A \equiv \neg \circledcirc \neg A \\
\blacklozenge A \equiv \mathbf{true}\, \mathcal{S}\, A \\
\blacksquare A \equiv \neg(\mathbf{true}\, \mathcal{S}\, (\neg A)) \\
A \mathcal{Z} B \equiv (A \mathcal{S} B) \vee (\blacksquare A)
\end{array}
$$

2.3 Notation

Where possible we adopt the following conventions.

- We use the lowercase letters $p, p_1, \ldots, q, t$ to represent propositions.
- We use the lowercase letters $k, k_1, \ldots l, l_1 \ldots$ to represent literals.
- We use the uppercase letters A, B, C, D as meta-variables ranging over formulae of the logical languages we consider.
- We use the uppercase letter R to represent sets of formulae.

2.4 A Normal Form for PTL

Formulae in PTL can be transformed to a normal form, Separated Normal Form (SNF) (Fisher, 1991), which is the basis of the resolution method used in this paper. SNF was introduced first in (Fisher, 1991) and has been extended to first-order temporal logic in (Fisher, 1992). 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.

The translation into SNF depends on three main steps, renaming, the removal of unwanted temporal operators and classical operations to rewrite formulae into the correct form. Renaming is the replacement of subformulae by a new proposition symbol and linking the truth value of this new proposition symbol to the subformula at all moments in time. The removal of unwanted temporal operators is carried out using their fixpoint definitions with the introduction of new proposition symbols. Each of these operations give a linear increase in the number of rules and proposition symbols. However, the application of classical operations, for example rewriting the right hand sides of formulae into DNF, may be exponential.

Formulae in SNF are of the general form

$$\Box \bigwedge_i A_i$$

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

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

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 A_i. We note that negations only appear in front of propositions and rules in SNF.

We note a variant on SNF called merged-SNF (SNF_m) (Fisher, 1991) for combining global $\Box$-rules that is used in the application of the temporal resolution rule in Section 3.2. Given a set X of global $\Box$-rules the set X_m of merged SNF rules is generated as follows

- any rule in X is also in X_m;
- for any rules $\bigcirc A \Rightarrow C$ and $\bigcirc B \Rightarrow D$ in X_m a (simplified) rule equivalent to $\bigcirc (A \wedge B) \Rightarrow C \wedge D$ is in X_m.

3 The resolution procedure

Here we present a review of the temporal resolution method (Fisher, 1991). 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 (Peim, 1994).

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 $\square$–rules, or global $\square$–rules, may be resolved using the following (step resolution) rule where $\mathcal{L}_1$ and $\mathcal{L}_2$ are both of the form $\bullet X$, where X is a conjunction of literals, or both **start**.

$$\begin{array}{rcl} \mathcal{L}_1 & \Rightarrow & A \vee p \\ \mathcal{L}_2 & \Rightarrow & B \vee \neg p \\ \hline (\mathcal{L}_1 \wedge \mathcal{L}_2) & \Rightarrow & A \vee B \end{array}$$

Once a contradiction within a state is found using step resolution, the rule with **false** on the right hand side is removed and replaced by an additional pair of rules as follows.

$$\{ \bullet A \Rightarrow \mathbf{false} \} \quad \longrightarrow \quad \left\{ \begin{array}{rcl} \mathbf{start} & \Rightarrow & \neg A \\ \bullet\mathbf{true} & \Rightarrow & \neg A \end{array} \right\}$$

This rule states that if, by satisfying A in the last moment in time a contradiction is produced, then A must never be satisfied in *any* moment in time. The new constraints therefore represent $\square\neg A$

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.

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

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 $\square\neg l$, for example a set of rules that together have the effect of $\bullet A \Rightarrow \square\neg l$. However the interaction between the '$\bigcirc$' and '$\square$' 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 $\square$–operators. So, resolution will be between a $\Diamond$–rule and a *set* of rules that together imply an $\square$–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 $\square$–rules, which taken together force $\neg l$ always to be satisfied.

The temporal resolution rule is given by the following

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

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. This rule has the following 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 once $\bigvee_i A_i$ has been satisfied in the last moment in time, the set of rules $\mathbf{O} A_i \Rightarrow B_i$ together imply $\Box \neg l$. So if any of the A_i are satisfied then $\neg l$ will *always* be satisfied, i.e.,

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

Such a set of rules are known as a *loop* in $\neg l$. The efficient detection of loops has been considered in (Dixon, 1996). The algorithm that has been shown to be the most efficient from those considered, known as Breadth-First Search, is described in (Dixon, 1998).

3.3 The temporal resolution algorithm

Given any temporal formula A to be shown unsatisfiable the following steps are performed.

1. Translate A into a set of SNF rules A_s.

2. Perform step resolution (including simplification and subsumption) until either

    (a) false is derived - terminate noting A unsatisfiable; or

    (b) no new resolvents are generated - continue at step 3.

3. Select an eventuality from the right hand side of a $\Diamond$–rule within A_s, for example $\Diamond l$. Search for loops in $\neg l$ and generate the resolvents.

4. If any new formulae have been generated, translate the resolvents into SNF add them to the rule-set and go to step 2, otherwise continue to step 5.

5. Terminate declaring A satisfiable.

3.4 Completeness

Completeness, i.e. given an unsatisfiable set of rules there is a refutation by the temporal resolution method, is shown in (Peim, 1994). An outline of the proof is given below.

A behaviour graph $G = (N, E)$, where N is the set of nodes and E the set of edges in G, is constructed representing the possible models for a set of rules. Nodes, N are pairs (V, Y) where V is a valuation of all the propositions in the set of rules and Y is a subset of the literals appearing on the right hand side of the $\Diamond$-rules. Any additional propositions required from translating resolvents from temporal resolution into SNF are added at the start. For sets of rules with no $\Diamond$-rules (i.e. the Y component of each node will be $\emptyset$) there is an edge $e \in E$ between two nodes $(V, \emptyset)$ and $(V', \emptyset)$ if and only if for every global $\Box$-rule whose left hand side is satisfied by V then its right hand side must be satisfied by V'. The Y component of each node is used to keep track of unsatisfied eventualities. Nodes whose valuations satisfy the right hand side of the initial $\Box$ rules are known as initial nodes. The behaviour graph for the set of rules is the set of nodes and edges reachable from an initial node. Deletions in the graph must be carried out as all paths through the graph may not represent models for the set of rules. For example any node with no successors must be deleted and any nodes containing eventualities that cannot be satisfied must also be deleted. Deletions in the graph produce a reduced behaviour graph. It is shown that the reduced behaviour graph for a set of rules is empty if and only if the set of rules is unsatisfiable. Deletions in the graph are shown to correspond with resolution rules. In particular the deletion of nodes with no successor is shown to correspond with the application of step resolution rules and the rewriting of constraints that represent false. The removal of subgraphs where an eventuality is not satisfied corresponds to temporal resolution.

3.5 Example

To illustrate the temporal resolution method we consider an example showing how a contradiction can be obtained from the following unsatisfiable set of rules.

$$
\begin{array}{lrcl}
1. & \mathbf{start} & \Rightarrow & \Diamond l \\
2. & \mathbf{start} & \Rightarrow & c \\
3. & \mathbf{start} & \Rightarrow & \neg b \\
4. & \mathbf{start} & \Rightarrow & \neg l \\
5. & \bigcirc a & \Rightarrow & w \\
6. & \bigcirc b & \Rightarrow & \neg w \\
7. & \bigcirc (c \wedge \neg a) & \Rightarrow & c \\
8. & \bigcirc (c \wedge \neg b) & \Rightarrow & c \\
9. & \bigcirc c & \Rightarrow & \neg l \\
10. & \bigcirc \neg d & \Rightarrow & c \\
11. & \bigcirc e & \Rightarrow & \neg d
\end{array}
$$

First we trying applying the step resolution rule between the initial $\square$-rules (rules 2–4) or between the global $\square$-rules (rules 5–11). We can only apply the step resolution rule between rules 5 and 6 giving

$$\bigcirc(a \wedge b) \Rightarrow \mathbf{false}.$$

This is rewritten as the following pair of rules.

$$\begin{array}{rrcll} 12. & \mathbf{start} & \Rightarrow & \neg a \vee \neg b & [5,6 \text{ step resolution}] \\ 13. & \bigcirc\mathbf{true} & \Rightarrow & \neg a \vee \neg b & [5,6 \text{ step resolution}] \end{array}$$

Note, rule 12 is subsumed by rule 3 and can be deleted. As no more step resolution can be carried out we try apply the temporal resolution rule to the initial $\diamond$-rule (rule 1) and search for a loop in $\neg l$. By combining rules 7, 9, 13 and 8, 9, 13 we obtain the following loop in $\neg l$.

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

This is resolved with rule 1 giving the following.

$$\mathbf{start} \Rightarrow (\neg(c \wedge \neg a) \wedge \neg(c \wedge \neg b))\, \mathcal{W}\, l$$

This resolvent is then translated into SNF obtaining several rules one being

$$\begin{array}{rrcll} 14. & \mathbf{start} & \Rightarrow & l \vee \neg c \vee b & [1,7,8,9,13 \text{ temporal resolution}]. \end{array}$$

A contradiction can be obtained by performing step resolution.

$$\begin{array}{rrcll} 15. & \mathbf{start} & \Rightarrow & l \vee \neg c & [3,14 \text{ step resolution}] \\ 16. & \mathbf{start} & \Rightarrow & l & [2,15 \text{ step resolution}] \\ 17. & \mathbf{start} & \Rightarrow & \mathbf{false} & [4,16 \text{ step resolution}] \end{array}$$

4 Reducing the Rule-Set

As the application of the temporal resolution rule 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 (Dixon, 1996). We note that even for small problems, such as the list of valid temporal formulae found in (Manna and Pnueli, 1981), 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.

4.1 Overview

When looking at non-trivial examples we may have to search for loops in large rule-sets. The algorithm proposed below aims to reduce the size of rule-sets in which we must search without removing any rules that are necessary for the loops. First we look at different SNF rules that may be combined to make a loop and use our observations to develop an algorithm to remove rules that we know will not form part of the loop.

Let us partition the literals, from the set of rules that form a loop, into those for *loop propagation* and those that provide the *required literal*. For example if we are resolving with the eventuality $\Diamond l$ and have the following rules

$$\begin{array}{rcl} \bigcirc a & \Rightarrow & a \\ \bigcirc a & \Rightarrow & \neg l \end{array}$$

that may be combined to make the loop

$$\bigcirc a \Rightarrow a \wedge \neg l$$

then the literal *a propagates the loop* and $\neg l$ is the *required literal*. For any literals *propagating the loop* occurring on the left hand side of a rule there must be some rules with this literal occurring on the right hand side so that this rule will *fire*. The set of SNF rules

$$\begin{array}{rcl} \bigcirc a & \Rightarrow & \neg b \\ \bigcirc \neg b & \Rightarrow & a \\ \bigcirc a & \Rightarrow & \neg l \\ \bigcirc \neg b & \Rightarrow & \neg l \end{array}$$

may be combined to form the loop

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

if we are resolving with the eventuality $\Diamond l$. We note that a and $\neg b$, the literals for *loop propagation*, each occur on both the left hand side of some rules and the right hand side of some rules.

Loops we detect may involve rules that have a literal that is positive (i.e. a proposition with no negations directly preceding it) on the left of one of the rules and negative (i.e. a proposition with a negation directly preceding it) on the left of another, even though the loop does not depend on this literal (in the terminology we have used above this literal does not propagate the loop). For example, in the rules below, the literals b and $\neg b$ do not propagate the loop but we require all three of these rules to detect the loop.

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

This "matching" of positive and negative literals on the left hand side of rules could be viewed as *left hand side resolution*. For example we could allow a rule such as the following as part of the resolution system.

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

Such a rule is sound, however is not needed for completeness and in practice produces many extra rules.

If we have not performed all the step resolution possible then SNF rules that can be combined to form loops may have a literal that is positive on the right hand side of one and negative on the right hand side of the other, even though the loop does not depend on that literal. For example with the following rule-set

$$\begin{aligned}
\bigcirc a &\Rightarrow a \vee \neg b \\
\bigcirc a &\Rightarrow a \vee b \\
\bigcirc a &\Rightarrow \neg l
\end{aligned}$$

performing step resolution we would resolve the first two rules to obtain $\bigcirc a \Rightarrow a$ which would then subsume these two rules.

Finally the *literal we require*, $\neg l$ may just occur on the right hand side of SNF rules merged to form loops or may be also used to *propagate the loop*, i.e. appear on the left hand side. Its negation however, the literal l, will not appear on its own on the right hand side as the definition for the loop requires each right hand side to imply $\neg l$. Similarly if l occurs as a disjunct on the right hand side of a rule then we may always resolve it with a rule that has $\neg l$ on the right hand side to remove the l. Further, l should not appear on the left hand side. We try to use the above observations to delete rules that will not form part of the loop.

Note the above observations have ignored parts of loops that can be deleted without affecting the resolvents obtained. For example the loop

$$\begin{aligned}
\bigcirc (a \wedge b) &\Rightarrow (a \wedge \neg l) \\
\bigcirc a &\Rightarrow (a \wedge \neg l)
\end{aligned}$$

is equivalent to

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

as the resolvent from the former when resolved with $\mathbf{start} \Rightarrow \Diamond l$ is

$$\mathbf{start} \Rightarrow (\neg(a \wedge b) \wedge \neg a)\ \mathcal{W}\, l$$

which is equivalent to

$$\mathbf{start} \Rightarrow (\neg a)\, \mathcal{W}\, l$$

the resolvent obtained when resolving the latter loop with $\mathbf{start} \Rightarrow \Diamond l$.

Finally we ignore the *lead into the loop* for example the rule $\bigcirc b \Rightarrow (a \wedge \neg l)$ in the following.

$$\begin{aligned}
\bigcirc b &\Rightarrow (a \wedge \neg l) \\
\bigcirc a &\Rightarrow (a \wedge \neg l)
\end{aligned}$$

This can be safely ignored as any resolvents obtained from resolving this loop with, for example, $\mathbf{start} \Rightarrow \Diamond l$ can be obtained by temporal resolution between

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

and the same $\diamondsuit$-rule and and extra step resolution. To illustrate this consider the following rule-set.

$$\begin{array}{rrcl}
1. & \boldsymbol{\bigcirc} b & \Rightarrow & a \\
2. & \boldsymbol{\bigcirc} a & \Rightarrow & a \\
3. & \boldsymbol{\bigcirc} b & \Rightarrow & \neg l \\
4. & \boldsymbol{\bigcirc} a & \Rightarrow & \neg l \\
5. & \mathbf{start} & \Rightarrow & \diamondsuit l
\end{array}$$

Combining rules 1 and 3, 2 and 4 to obtain the first of our loops we get the resolvent

$$\mathbf{start} \Rightarrow (\neg a \wedge \neg b)\,\mathcal{W}\,l.$$

Translating this into SNF we obtain the rules

$$\begin{array}{rrcl}
6. & \mathbf{start} & \Rightarrow & l \vee \neg a \\
7. & \mathbf{start} & \Rightarrow & l \vee \neg b \\
8. & \mathbf{start} & \Rightarrow & l \vee t \\
9. & \boldsymbol{\bigcirc} t & \Rightarrow & l \vee \neg a \\
10. & \boldsymbol{\bigcirc} t & \Rightarrow & l \vee \neg b \\
11. & \boldsymbol{\bigcirc} t & \Rightarrow & l \vee t
\end{array}$$

However if we resolve with the loop

$$\boldsymbol{\bigcirc} a \quad \Rightarrow \quad (a \wedge \neg l)$$

from combining only rules 2 and 4 we obtain the rules 6, 8, 9 and 11. Now we can perform step resolution

$$\begin{array}{rrcll}
12. & \boldsymbol{\bigcirc}(t \wedge b) & \Rightarrow & l & [1,9 \text{ step resolution}] \\
13. & \boldsymbol{\bigcirc}(t \wedge b) & \Rightarrow & \mathbf{false} & [3,12 \text{ step resolution}] \\
14. & \mathbf{start} & \Rightarrow & \neg t \vee \neg b & [13 \text{ step resolution}] \\
15. & \boldsymbol{\bigcirc}\,\mathbf{true} & \Rightarrow & \neg t \vee \neg b & [13 \text{ step resolution}] \\
16. & \mathbf{start} & \Rightarrow & l \vee \neg b & [8,14 \text{ step resolution}] \\
17. & \boldsymbol{\bigcirc} t & \Rightarrow & l \vee \neg b & [11,15 \text{ step resolution}]
\end{array}$$

where rules 16 and 17 are the same as the resolvents from the first loop.

4.2 The Rule Reduction Algorithm I

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

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

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

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

$\mathfrak{L}$ is the set of propositions occuring positively on the left hand side of the set of rules or negatively on the right hand side of the set of rules; and

$\mathfrak{R}$ is the set of propositions occuring positively on the right hand side of the set of rules or negatively on the left hand side of the set of rules.

Using these sets we delete rules that aren't required for combination into loops based upon the above observations about propagating the loop and left and right hand side resolution.

Algorithm Assume we are resolving with the eventuality $\Diamond l$ and that R is the set of global $\Box$-rules before rule reduction takes place.

1. Delete any rules containing a positive occurrence of l either on the left or right hand sides.
2. Calculate the sets $\mathfrak{L}$ and $\mathfrak{R}$ from R.
3. Delete l from both sets.
4. If $\mathfrak{L} = \mathfrak{R}$ terminate returning R otherwise perform step 5.
5. Repeat until $\mathfrak{L} = \mathfrak{R}$.
   (a) If $p \in \mathfrak{L}$ and $p \notin \mathfrak{R}$ delete all rules in R containing p or $\neg p$ and set R to be the set of SNF rules following deletions.
   (b) If $p \in \mathfrak{R}$ and $p \notin \mathfrak{L}$ delete all rules in R containing p or $\neg p$ and set R to be the set of SNF rules following deletions.
6. Recalculate the sets $\mathfrak{L}$ and $\mathfrak{R}$ from R and proceed to step 4.

4.2.1 Example

To illustrate the algorithm we use the set of global $\Box$-rules given in § 3.5 prior to performing temporal resolution. These rules are repeated below. Recall we are resolving with the eventuality $\Diamond l$.

$$\begin{array}{rrcl}
5. & \bigcirc a & \Rightarrow & w \\
6. & \bigcirc b & \Rightarrow & \neg w \\
7. & \bigcirc (c \wedge \neg a) & \Rightarrow & c \\
8. & \bigcirc (c \wedge \neg b) & \Rightarrow & c \\
9. & \bigcirc c & \Rightarrow & \neg l \\
10. & \bigcirc \neg d & \Rightarrow & c \\
11. & \bigcirc e & \Rightarrow & \neg d \\
13. & \bigcirc \mathbf{true} & \Rightarrow & \neg a \vee \neg b
\end{array}$$

1. We build sets $\mathfrak{L}$ and $\mathfrak{R}$ as described and having deleted l from both sets we obtain $\mathfrak{L} = \{a, b, c, d, e, w\}$ and $\mathfrak{R} = \{a, b, c, d, w\}$.

2. We have $e \in \mathfrak{L}$ but $e \notin \mathfrak{R}$ so we delete any rules containing e or $\neg e$ (rule 11).

3. Next we recalculate $\mathfrak{L}$ and $\mathfrak{R}$, using rules 5–10 and 13 and obtain $\mathfrak{L} = \{a, b, c, w\}$ and $\mathfrak{R} = \{a, b, c, d, w\}$.

4. Now we have $d \in \mathfrak{R}$ but $d \notin \mathfrak{L}$ so we delete any rules containing d or $\neg d$ (rule 10) and remain with rules 5–9 and 13.

5. Again recalculating $\mathfrak{L}$ and $\mathfrak{R}$ we get $\mathfrak{L} = \{a, b, c, w\}$ and $\mathfrak{R} = \{a, b, c, w\}$, so $\mathfrak{L} = \mathfrak{R}$ and the algorithm returns with rules 5–9 and 13.

4.3 Improvements to the Algorithm

We have assumed that all step resolution possible has been carried out so retaining rules on the grounds of right hand side resolution is unnecessary. As an improvement on our previous algorithm we define four sets of propositions so that rules would not be allowed to remain if the only reason was because of right hand side resolution.

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

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

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

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

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

These sets are compared with each other and if, for example, a proposition occurs in $\mathfrak{L}^+$ and not in either $\mathfrak{L}^-$ or $\mathfrak{R}^+$ then any rules with this proposition occurring positively on the left hand side are deleted and the proposition is deleted from $\mathfrak{L}^+$. If a proposition p did occur in both $\mathfrak{L}^+$ and $\mathfrak{R}^+$ then we do not want to delete rules with p occuring positively on the left or positively on the right, as p might be a literal that is propagating the loop and these rules may be the ones combined to form the loops. Similarly if p is a member of both $\mathfrak{L}^+$ and $\mathfrak{L}^-$ then although p may not propagate the loop the rules with p (and $\neg p$) on the left hand side may be required to build the loop although p and $\neg p$ may be removed through some form of left hand side resolution.

4.3.1 Implementation Improvements

If we store the propositions only we must recalculate the members of each set after all deletions have been made as we may end up with a proposition occurring in one of the sets that belonged to a rule that has been deleted. However to prevent the need for recalculation we can, instead of building sets of propositions, build multi-sets of propositions. After each cycle of deletions from the set of global $\square$-rules, rather than having to recalculate the four sets of propositions, the number of occurrences of

p in for example $\mathfrak{L}^+$, for example, indicates the number of occurrences of p in a left hand side positive position left in the rule-set. When a rule is deleted, each literal occurring in that rule must have a proposition deleted from the relevant set. Thus the definitions for $\mathfrak{L}^+$, $\mathfrak{L}^-$, $\mathfrak{R}^+$, $\mathfrak{L}^-$ are amended to make sets into multi-sets.

4.4 The Rule Reduction Algorithm II

Let us assume we are trying to resolve with the eventuality $\Diamond l$ 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.
   (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 $\circledcirc a \Rightarrow a$, $\circledcirc a \Rightarrow \neg l$ that together form the loop $\circledcirc a \Rightarrow a \wedge \neg l$, 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

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

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.

4.5 Example

Again we illustrate the improved algorithm by showing its operation on the set of global $\square$-rules in §3.5 prior to applying the temporal resolution rule. To aid readability we repeat the set of rules. Recall we are resolving with the eventuality $\Diamond l$.

$$\begin{array}{lrcl} 5. & \circledcirc a & \Rightarrow & w \\ 6. & \circledcirc b & \Rightarrow & \neg w \\ 7. & \circledcirc(c \wedge \neg a) & \Rightarrow & c \\ 8. & \circledcirc(c \wedge \neg b) & \Rightarrow & c \\ 9. & \circledcirc c & \Rightarrow & \neg l \\ 10. & \circledcirc \neg d & \Rightarrow & c \\ 11. & \circledcirc e & \Rightarrow & \neg d \\ 13. & \circledcirc \mathbf{true} & \Rightarrow & \neg a \vee \neg b \end{array}$$

Firstly we build the multi-sets $\mathfrak{L}^+$, $\mathfrak{L}^-$, $\mathfrak{R}^+$, and $\mathfrak{R}^-$ taking each rule in turn. In rule 5, 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 6, 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 $\mathfrak{L}^+ = \{a, b, c, c, c, e\}$, $\mathfrak{L}^- = \{a, b, d\}$, $\mathfrak{R}^+ = \{c, c, c, w\}$, and $\mathfrak{R}^- = \{a, b, d, w\}$. 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 5) and that c occurs positively on the right hand side of three rules (rules 7, 8 and 10).

When carrying out deletions, starting with $\mathfrak{L}^+$ we see a and b are both in $\mathfrak{L}^-$, c is in $\mathfrak{R}^+$ but e is neither in $\mathfrak{L}^-$ or $\mathfrak{R}^+$. We need to remove all the rules with e in the $\mathfrak{L}^+$ position and adjust all the other multi-sets. That is, the algorithm tells us that any rule with a positive e on the left hand side can be deleted as it cannot form part of the loop. We see rule 11 is the only rule containing e in a $\mathfrak{L}^+$ position so remove this rule, delete e from $\mathfrak{L}^+$ and as $\neg d$ occurs on the right hand side of rule 11, delete d from $\mathfrak{R}^-$. The rules remaining in our set are rules 5–10 and rule 13 and our four multi-sets are $\mathfrak{L}^+ = \{a, b, c, c, c\}$, $\mathfrak{L}^- = \{a, b, d\}$, $\mathfrak{R}^+ = \{c, c, c, w\}$, and $\mathfrak{R}^- = \{a, b, w\}$.

Next considering $\mathfrak{L}^-$ the propositions a and b are both in $\mathfrak{L}^+$ however d is not a member of either $\mathfrak{L}^+$ or $\mathfrak{R}^-$. Thus we delete any rules with a negated d on the left hand side, i.e. rule 10. The proposition d is deleted from $\mathfrak{L}^-$ and c is deleted from $\mathfrak{R}^+$ as c occurs on the right hand side of rule 10. The rules 5–9 and 13 remain and the multi-sets are $\mathfrak{L}^+ = \{a, b, c, c, c\}$, $\mathfrak{L}^- = \{a, b\}$, $\mathfrak{R}^+ = \{c, c, w\}$, and $\mathfrak{R}^- = \{a, b, w\}$.

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. We see rule 5 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 5, from $\mathfrak{L}^+$. The rules remaining in our set are rules 6–9 and 13 and our four multi-sets are $\mathfrak{L}^+ = \{b, c, c, c\}$, $\mathfrak{L}^- = \{a, b\}$, $\mathfrak{R}^+ = \{c, c\}$, and $\mathfrak{R}^- = \{w, a, b\}$.

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 6 is the only rule with $\neg w$ on the right hand side so we delete this rule leaving rules 7–9 and 13, 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 7, 8, 9, 13. We may now use a loop finding algorithm to detect any loops and again return the loop

$$\begin{aligned} \bigcirc(c \wedge \neg a) &\Rightarrow (c \wedge \neg a \wedge \neg l) \vee (c \wedge \neg b \wedge \neg l) \\ \bigcirc(c \wedge \neg b) &\Rightarrow (c \wedge \neg a \wedge \neg l) \vee (c \wedge \neg b \wedge \neg l) \end{aligned}$$

from merging rules 7, 9, 13, and rules 8, 9, 13.

5 Completeness and complexity

5.1 Completeness

Definition 1 *(A necessary set of rules for a loop.) Take a loop L of the form $\bigcirc A_i \Rightarrow B_i$ for $i = 0$ to m and for any rule $\bigcirc A_j \Rightarrow B_j \in L$ for some $j = 0$ to m, such that $A_j \Rightarrow A_k$ for some $k = 0$ to m, $j \neq k$, delete $\bigcirc A_j \Rightarrow B_j$ from L. Continue until no more deletions are possible. The remaining set of rules is known as* a necessary set of rules for L.

Lemma 1 *Let L be a loop in $\neg l$ and L' a necessary set of rules for L. L' is also a loop in l and the resolvents from L when resolving with $\mathcal{L} \Rightarrow \Diamond l$ (where $\mathcal{L}$ is a last-time formula or* **start***) are equivalent to the resolvents from L' resolved with the same $\Diamond$-rule.*

Proof Let L be the loop $\bullet A_i \Rightarrow B_i$ for $i = 0$ to m and L' be a necessary set of rules, $\bullet A'_j \Rightarrow B'_j$ for $j = 0$ to m', formed from L. First we check that L' is a loop in $\neg l$. For L' to be a loop in $\neg l$ the following two conditions must be satisfied

1. for all $0 \leq j \leq m' \vdash B'_j \Rightarrow \neg l$;
2. $\vdash B'_j \Rightarrow \bigvee_{i=0}^{m'} A'_i$.

The former holds as for any rule $r \in L'$ we have $r \in L$ as rules have only been deleted from L to form L'. As $B_i \Rightarrow \neg l$ for each rule in L we have $B'_j \Rightarrow \neg l$ for each rule in L'. As L is a loop, for each B_i in L we have

$$B_i \Rightarrow \bigvee_{j=0}^{m} A_j.$$

To obtain L' we have deleted rules $\bullet A_j \Rightarrow B_j$, for some $j = 0$ to m, such that $A_j \Rightarrow A_k$ for some $k = 0$ to m. Hence

$$\bigvee_{i=0}^{m} A_i \Leftrightarrow \bigvee_{j=0}^{m'} A'_j$$

and

$$B'_j \Rightarrow \bigvee_{j=0}^{m'} A'_j$$

for each $j = 0$ to m' as required. As a necessary set of rules is also a loop it is also known as a *necessary loop*.

The resolvent from resolving $\mathcal{L} \Rightarrow \Diamond l$ with L is

$$\mathcal{L} \Rightarrow \neg \left(\bigvee_{i=0}^{m} A_i \right) \mathcal{W} \, l$$

and with L' is

$$\mathcal{L} \Rightarrow \neg \left(\bigvee_{i=0}^{m'} A'_i \right) \mathcal{W} \, l$$

As

$$\bigvee_{i=0}^{m} A_i \Leftrightarrow \bigvee_{j=0}^{m'} A'_j$$

the resolvents are equivalent.

Definition 2 *(A compact loop in $\neg l$.) Given a necessary loop L in $\neg l$, $\bullet A_i \Rightarrow B_i$ for $i = 0$ to m perform the following deletions until no more can be applied. Delete any rule $r = \bullet A_k \Rightarrow B_k \in L$ for $k = 0$ to m such that r itself is not a loop in $\neg l$, $L - r$ is still a loop in $\neg l$, and $B_k \Rightarrow \bigvee_{i=0}^{k-1} A_i \bigvee_{i=k+1}^{m} A_i$. The loop remaining is called a* compact loop *in $\neg l$.*

Definition 3 *(The lead into a loop.) Given a necessary loop L in $\neg l$ form the compact loop L' by deleting rules from L. The set of rules $L - L'$ is known as the* lead into the loop.

Lemma 2 *Let L be a necessary loop in $\neg l$. Let L' be a loop in $\neg l$ formed from L by deleting one rule r following the conditions for a compact loop. For any resolvent r' from resolution between L and $\mathcal{L} \Rightarrow \Diamond l$ (where $\mathcal{L}$ is a last-time formula or* **start***) a resolvent r'' such that $r'' \Rightarrow r'$ can be obtained by resolution between L' and the same $\Diamond$-rule and additional step resolution.*

Proof Let L be the loop $\bigcirc A_i \Rightarrow B_i$ for $i = 0$ to m. Without loss of generality assume we have deleted the rule $\bigcirc A_0 \Rightarrow B_0$ so L' is the set of rules $\bigcirc A_j \Rightarrow B_j$ for $j = 1$ to m. L' must be a loop in $\neg l$ by the definition of a compact loop.

Considering the rule that has been deleted, $\bigcirc A_0 \Rightarrow B_0$, we have

$$\begin{array}{rcl} B_0 & \Rightarrow & \bigvee_{i=1}^{m} A_i \\ B_0 & \Rightarrow & \neg l \end{array}$$

The resolvents obtained from resolving L' with $\mathcal{L} \Rightarrow \Diamond l$ are

$$\begin{array}{rcl} \mathcal{L} & \Rightarrow & \neg A_i \vee l \\ \bigcirc t & \Rightarrow & \neg A_i \vee l \\ \mathcal{L} & \Rightarrow & t \vee l \\ \bigcirc t & \Rightarrow & t \vee l \end{array}$$

for $i = 1$ to m where t is a new proposition. By resolving with L rather than L' we would have obtained the additional resolvents

$$\begin{array}{rcl} \mathcal{L} & \Rightarrow & \neg A_0 \vee l \\ \bigcirc t & \Rightarrow & \neg A_0 \vee l. \end{array}$$

We have

$$B_0 \Rightarrow \bigvee_{i=1}^{m} A_i$$

so

$$B_0 \wedge \bigwedge_{i=1}^{m} \neg A_i$$

will give a contradiction. Resolving rules that have been combined to produce $\bigcirc A_0 \Rightarrow B_0$ with some of the $\bigcirc t \Rightarrow \neg A_i \vee l$ rules will produce

$$\bigcirc t \wedge X \Rightarrow l$$

where $A_0 \Rightarrow X$. X is is the conjunction of the literals on the left hand side of some of the rules that have been combined to make $\bigcirc A_0 \Rightarrow B_0$. As $B_0 \Rightarrow \neg l$ and we assume all step resolution has been carried out there is a rule $\bigcirc Y \Rightarrow \neg l$ that has been combined to make $\bigcirc A_0 \Rightarrow B_0$ so resolving with this rule we obtain

$$\bigcirc t \wedge X \wedge Y \Rightarrow \mathbf{false}$$

where $A_0 \Rightarrow (X \wedge Y)$. Rewriting, the rules obtained are

$$\begin{aligned} \textbf{start} &\Rightarrow \neg t \vee \neg X \vee \neg Y \\ \bigcirc \textbf{true} &\Rightarrow \neg t \vee \neg X \vee \neg Y \end{aligned}$$

Resolution with the third and fourth resolvent from L' gives us

$$\mathcal{L} \Rightarrow l \vee \neg X \vee \neg Y$$
$$\bigcirc t \Rightarrow l \vee \neg X \vee \neg Y$$

As $A_0 \Rightarrow X \wedge Y$ we have $\neg X \vee \neg Y \Rightarrow \neg A_0$ so $l \vee \neg X \vee \neg Y \Rightarrow l \vee \neg A_0$ as required.

Lemma 3 *Let L be a necessary loop in $\neg l$. The resolvents from resolution with $\mathcal{L} \Rightarrow \Diamond l$ (where $\mathcal{L}$ is a last-time formula or* **start***) can be obtained by resolution between L', a compact loop formed from L, and the same $\Diamond$-rule plus some additional step resolution.*

Proof Let $L_0 = L$, L_1 be the loop having deleted one rule from L_0, L_2 be the result of having deleted one rule from L_1 etc., where if n deletions are required to make a compact loop $L_n = L'$. From the definition of a compact loop $L_1, L_2 \ldots L_n$ are all loops. From Lemma 2 we can generate (more general) resolvents by resolving with L_{j+1} and the $\Diamond$-rule plus some step resolution than resolving with L_j and the $\Diamond$-rule for $j = 0$ to $n-1$. Hence resolving with $L_n = L'$ (a compact loop) plus step resolution produces the same resolvents as resolving with L.

Lemma 4 *Let L be a necessary loop in $\neg l$ and L' a compact loop formed from L. Applying the rule reduction algorithm before loop search will not remove any rules that have been combined to form L'.*

Proof

Let L' be a set of SNF_m rules of the form $\bigcirc A_i \Rightarrow B_i$ for $i = 0$ to m then as L' is a loop in $\neg l$ the following holds

- for all $i = 0$ to m, $B_i \Rightarrow \neg l$; and
- for all $i = 0$ to m, $B_i \Rightarrow \bigvee\limits_{j=0}^{m} A_j$.

Let us assume a rule, $r \in R$, has been deleted by the rule reduction algorithm but is required to be combined for a rule, $\bigcirc A_h \Rightarrow B_h$ in L' and examine each case in turn. We assume all the step resolution has been carried out on the global $\square$–rules.

1. Let l be a literal on the left or right hand side of r. The former cannot happen as each right hand side in L' implies $\neg l$ and each right hand side in L' implies the disjunction of the left hand sides. If a left hand side contains l it can be deleted leaving a loop remaining. Hence L' was not a compact loop contradicting our assumptions so l cannot be a literal on the left hand side of a rule. If l is a literal on the right hand side of r then as $B_h \Rightarrow \neg l$ holds and all step resolution has been carried out we must have combined a rule of the form $\bigcirc X \Rightarrow \neg l$ with the rules combined to give $\bigcirc A_h \Rightarrow B_h$. By performing step resolution between r and $\bigcirc X \Rightarrow \neg l$ we can remove l from the

right hand side of r to give the rule r'. If we combine r' rather than r with the other rules we still obtain $\bigcirc A_h \Rightarrow B_h$. We have assumed that all step resolution has been carried out so deleting r will make no difference to L'.

2. Let x be a positive literal on the left hand side of r then we can delete this rule if

   (a) there is no rule with x on the right hand side $(\mathfrak{L}^+, \mathfrak{R}^+)$; and

   (b) there is no rule with $\neg x$ on the left hand side $(\mathfrak{L}^+, \mathfrak{L}^-)$.

   The literal x will be a conjunct on the left hand side of $\bigcirc A_h \Rightarrow B_h$. From the second condition for being a loop we know each right hand side in L' must imply the disjunction of the left hand sides. For r either

   (a) to make r 'fire' there must be some B_i with an x as a conjunct as one of the disjuncts of B_i, so there must be an SNF rule with x on the right hand side to build this SNF_m rule; or

   (b) as for all $i = 0$ to m, $B_i \Rightarrow \bigvee_{j=0}^{m} A_j$ and no rules can be deleted we must be able to do some left hand side resolution so there must be some A_i where $\neg x$ is one of the conjuncts of A_i, but then $\neg x$ must appear on the left hand side of one of the SNF rules.

   Either way we have got a contradiction with our assumptions so this rule cannot be part of the loop. The argument is similar for $\neg x$ a negative literal on the left hand side of r and deals with the cases $(\mathfrak{L}^-, \mathfrak{R}^-)$ and $(\mathfrak{L}^-, \mathfrak{L}^+)$.

3. Let x be a positive literal on the right hand side of r then we can delete r if there is no rule with x on the left hand side $(\mathfrak{R}^+, \mathfrak{L}^+)$.

   Then either we must have an A_i with x as a conjunct on the left hand side, i.e. there must be an SNF rule with x on the left hand side or we have combined the rules too many times and we can delete this from the loop and still obtain a loop. The argument is similar for $\neg x$ a negative literal on the right hand side of r giving the case $(\mathfrak{R}^-, \mathfrak{L}^-)$.

Theorem 1 *(Completeness of the Rule Reduction Algorithm) Let L be a loop in $\neg l$ for resolution with $\mathcal{L} \Rightarrow \Diamond l$. If the rule reduction algorithm is applied before searching for a loop in $\neg l$ then any resolvents produced from resolution with L can be derived by applying rule reduction, loop search and step resolution.*

From Lemma 4 we show that applying rule reduction will not delete any rules needed to be combined to make L' the compact loop for L. Lemma 3 shows that the resolvents from a necessary loop formed from L and $\mathcal{L}Q \Rightarrow \Diamond l$ can be generated by resolution between the compact loop L' for L and $\mathcal{L}Q \Rightarrow \Diamond l$ and by step resolution between these rules and other rules in the rule-set. Finally Lemma 1 shows that any resolvents from L are equivalent to any resolvents from a necessary loop formed from L. So by applying rule reduction before loop search we will still find a loop from which we can derive the same (or more general) rules through temporal and then step resolution as we could if no rule reduction had been applied.

5.2 Complexity

Here we consider the complexity of various parts of the rule reduction algorithm. An alternative method to using multi-sets is to construct the sets $\mathfrak{R}^+$ etc as sets of pairs (p, c) where p is the proposition being considered and c is the number of times it occurs in this context. We use this slightly altered setting to consider the complexity. First we consider the cost of constructing the sets $\mathfrak{R}^+, \mathfrak{R}^-, \mathfrak{L}^+$ and $\mathfrak{L}^-$.

Given n different propositions in the set of rules being considered, the cost of adjusting the sets $\mathfrak{R}^+, \mathfrak{R}^-, \mathfrak{L}^+$ and $\mathfrak{L}^-$, to update with one literal from the set of rules will be at worst n. This is because we know which set the literal should belong to from its position whether it is preceded by a negation or not and we may have to search at most n propositions in a particular set before the relevant proposition is found. Given m rules each rule may contain at most $2n$ propositions so the total length of the rules in terms of the number of propositions will be $m \times 2n$. The cost of building the four sets of propositions $\mathfrak{R}^+, \mathfrak{R}^-, \mathfrak{L}^+$ and $\mathfrak{L}^-$, will be at worst $O(m \times n^2)$.

Next we consider the cost of deleting propositions from the four sets. The worst case is if all n propositions can be deleted from the four sets of propositions. We must consider the cost of ensuring a proposition occurs in one set and not another. This will be of order n as each set has at most n propositions. Hence the cost of deleting propositions from the sets is of order n^2.

Next we consider the deletion of rules from the rule-set. For each proposition deleted from $\mathfrak{R}^+$,$\mathfrak{R}^-$,$\mathfrak{L}^+$ and $\mathfrak{L}^-$, we will have to search all the global $\square$-rules which will be at worst $m \times 2n$ propositions long. As we have four sets of at worst n propositions we may have to carry this process out $4n$ times. Therefore the cost of deleting rules from the rule-set will be at worst of the order of $m \times n^2$. Hence the complexity of rule reduction will be at worst $O(m \times n^2)$.

We can compare this with the cost of the complexity of Breadth-First Search given in (Dixon, 1998) which is $O(2^{m+2n})$.

6 Results

A prototype implementation performing the temporal resolution method has been built. The programs are written in SICStus Prolog (Carlsson and Widen, 1991) 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 (Manna and Pnueli, 1981) 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

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

The set of test examples is given in Appendix A. We note that, although not presented in the results given here, larger examples have also been tackled, for example Peterson's Algorithm (Peterson, 1981; Pnueli, 1984).

The data for examples where the number of rules (before rule reduction) is greater than 10 is given in Appendix B. The complete set of results may be found in (Dixon, 1995). 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 (Dixon, 1998) however another method could be used if desired, see for example those described in (Dixon, 1996).

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.

7 Conclusions and further work

7.1 Conclusions

Theorem proving in temporal logics is vital if we want to prove properties about systems that are for example reactive, distributed or concurrent. The complexity of such logics is high (for example the complexity of satisfiability of PTL is PSPACE-complete (Sistla and Clarke, 1985)). Although this worst case complexity will not be affected by proposing strategies to guide the proof, such as that described in this paper, we hope the worst case complexity can be avoided in many cases. The proposed strategy gives an algorithm to quickly remove rules prior to the application of the temporal resolution rule so that the number of rules needing to be considered when applying the temporal resolution rule is reduced. In addition, the application of the algorithm will not affect the completeness of the method, that is, a refutation is guaranteed if and only if the set of rules the resolution method is applied to is unsatisfiable. 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. Thus the strategy is a step towards the goal of efficient temporal proof.

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.

The algorithm described is particular to the resolution method described in §3 and may not be applicable to other proof methods. This is no surprise as we would expect some strategies to be directly related to the proof method being used. However the resolution system has been extended to other logics such as the branching time temporal logics CTL (Bolotov and Fisher, 1997) and the temporal logics of knowledge (Dixon et al., 1997). Similarly the rule reduction algorithm may be extended to work in these different frameworks.

7.2 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 (Dixon et al., 1995). 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.

7.3 Related work

Decision procedures based on resolution have been developed for linear-time temporal logics in (Abadi and Manna, 1990; Cavalli and Fariñas del Cerro, 1984; Venkatesh, 1986), 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 (McCune, 1990). 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.

Other theorem provers for temporal logic that have been implemented are the tableau based system DP (Gough, 1984) and a module that forms part of the Logic Work Bench (LWB) (Jaeger et al., ; Heuerding, 1998) based on sequent calculi. Model checking systems are also available for example the STeP system (Bjorner et al., 1995), based on (Manna and Pnueli, 1992; Manna and Pnueli, 1995), which combines both model checking and deductive methods. As the current resolution system was developed as a prototype we leave an overall comparison with these systems until the completion of a C++ version of the resolution prover, CLATTER. We expect that rather than one approach being superior in all cases each method will have have a range of examples that it will perform better on.

Acknowledgements

Thanks to Michael Fisher, Howard Barringer, Graham Gough and Martin Peim for their comments and advice relating to this work. This work was supported partially by an EPSRC PhD Studentship and partially by EPSRC Research Grant GR/K57282. This paper is a revised and extended version of the paper (Dixon, 1997).

References

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

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

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

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

Carlsson, M. and Widen, J. (1991). *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, Kista, Sweden.

Cavalli, A. and Fariñas del Cerro, L. (1984). A Decision Method for Linear Temporal Logic. In Shostak, R. E., editor, *Proceedings of the 7th International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 113–127. Springer-Verlag.

Dixon, C. (1995). *Strategies for Temporal Resolution*. PhD thesis, Department of Computer Science, University of Manchester.

Dixon, C. (1996). Search Strategies for Resolution in Temporal Logics. In McRobbie, M. A. and Slaney, J. K., 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. Springer-Verlag.

Dixon, C. (1997). Temporal Resolution: Removing Irrelevant Information. In *Proceedings of TIME-97 the Fourth International Workshop on Temporal Representation and Reasoning*, pages 4–11, Daytona Beach, Florida. IEEE Computer Society Press.

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

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

Dixon, C., Fisher, M., and Wooldridge, M. (1997). Resolution for Temporal Logics of Knowledge. *Journal of Logic and Computation*. To appear.

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

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

Gabbay, D., Pnueli, A., Shelah, S., and Stavi, J. (1980). The Temporal Analysis of Fairness. In *Proceedings of the Seventh ACM Symposium on the Principles of Programming Languages*, pages 163–173, Las Vegas, Nevada.

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

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

Heuerding, A. (1998). *Sequent Calculi for Proof Search in Some Modal Logics, Theory - Implementation - Applications*. PhD thesis, University of Berne.

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

Lichtenstein, O., Pnueli, A., and Zuck, L. (1985). The glory of the past. In Parikh, R., editor, *Logics of Programs* (*Proc. Conf. Brooklyn USA 1985*), volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer-Verlag, Berlin.

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

Manna, Z. and Pnueli, A. (1989). The anchored version of the temporal framework. In de Bakker, J., de Roever, W.-P., and Rozenberg, G., editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 201–284. Springer-Verlag, Berlin.

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

Manna, Z. and Pnueli, A. (1995). *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York.

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

Nonnengart, A. (1996). Resolution-Based Calculi for Modal and Temporal Logics. In McRobbie, M. A. and Slaney, J. K., 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. Springer-Verlag.

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

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

Peterson, G. L. (1981). Myths about the Mutual Exclusion Problem. *Information Processing Letters*, 12(3):115–116.

Pnueli, A. (1984). In Transition From Global to Modular Temporal Reasoning about Programs. In Apt, K., editor, *Logics and Models of Concurrent Systems*, pages 123–144, La Colle-sur-Loup, France. NATO, Springer-Verlag.

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

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

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

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

Wolper, P. (1985). The Tableau Method for Temporal Logic: An overview. *Logique et Analyse*, 110–111:119–136.

Wolper, P., Vardi, M., and Sistla, A. P. (1983). Reasoning about Infinite Computation Paths. In *Proceedings of the Twentyfourth Symposium on the Foundations of Computer Science*, pages 185–194. IEEE.

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

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

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

A Examples Used

The set of examples used for the test results was taken from (Manna and Pnueli, 1981) and are given below.

No	Example
1	$\Box\neg w \Leftrightarrow \neg\Diamond w$
2	$\Diamond\neg w \Leftrightarrow \neg\Box w$
3	$\bigcirc\neg w \Leftrightarrow \neg\bigcirc w$
4	$w \Rightarrow \Diamond w$
5	$\Box w \Rightarrow w$
6	$\bigcirc w \Rightarrow \Diamond w$
7	$\Box w \Rightarrow \bigcirc w$
8	$\Box w \Rightarrow \Diamond w$
9	$\Box w \Rightarrow \bigcirc\Box w$
10	$w_1\,\mathcal{U}\,w_2 \Rightarrow \Diamond w_2$
11	$\Diamond\Box w \Rightarrow \Box\Diamond w$
12	$\Box w \Leftrightarrow \Box\Box w$
13	$\Diamond w \Leftrightarrow \Diamond\Diamond w$
14	$\Box\bigcirc w \Leftrightarrow \bigcirc\Box w$
15	$\Diamond\bigcirc w \Leftrightarrow \bigcirc\Diamond w$
16	$((\bigcirc w_1)\,\mathcal{U}\,(\bigcirc w_2)) \Leftrightarrow \bigcirc(w_1\,\mathcal{U}\,w_2)$
17	$\Box(w_1 \wedge w_2) \Leftrightarrow (\Box w_1 \wedge \Box w_2)$
18	$\Diamond(w_1 \vee w_2) \Leftrightarrow (\Diamond w_1 \vee \Diamond w_2)$
19	$\bigcirc(w_1 \wedge w_2) \Leftrightarrow (\bigcirc w_1 \wedge \bigcirc w_2)$
20	$\bigcirc(w_1 \vee w_2) \Leftrightarrow (\bigcirc w_1 \vee \bigcirc w_2)$
21	$\bigcirc(w_1 \Rightarrow w_2) \Leftrightarrow (\bigcirc w_1 \Rightarrow \bigcirc w_2)$
22	$\bigcirc(w_1 \Leftrightarrow w_2) \Leftrightarrow (\bigcirc w_1 \Leftrightarrow \bigcirc w_2)$
23	$((w_1 \wedge w_2)\,\mathcal{U}\,w_3) \Leftrightarrow ((w_1\,\mathcal{U}\,w_3) \wedge (w_2\,\mathcal{U}\,w_3))$
24	$(w_1\,\mathcal{U}\,(w_2 \vee w_3)) \Leftrightarrow ((w_1\,\mathcal{U}\,w_2) \vee (w_1\,\mathcal{U}\,w_3))$
25	$\Box w_1 \vee \Box w_2 \Rightarrow \Box(w_1 \vee w_2)$
26	$\Diamond(w_1 \wedge w_2) \Rightarrow \Diamond w_1 \wedge \Diamond w_2$
27	$((w_1\,\mathcal{U}\,w_3) \vee (w_2\,\mathcal{U}\,w_3)) \Rightarrow (w_1 \vee w_2)\,\mathcal{U}\,w_3$
28	$(w_1\,\mathcal{U}\,(w_2 \wedge w_3)) \Rightarrow ((w_1\,\mathcal{U}\,w_2) \wedge (w_1\,\mathcal{U}\,w_3))$

No	Example
29	$\Box(w_1 \Rightarrow w_2) \Rightarrow (\Box w_1 \Rightarrow \Box w_2)$
30	$\Box(w_1 \Rightarrow w_2) \Rightarrow (\Diamond w_1 \Rightarrow \Diamond w_2)$
31	$\Box(w_1 \Rightarrow w_2) \Rightarrow (\bigcirc w_1 \Rightarrow \bigcirc w_2)$
32	$\Box(w_1 \Rightarrow w_2) \Rightarrow ((w_1 \mathcal{U} w_3) \Rightarrow (w_2 \mathcal{U} w_3))$
33	$\Box(w_1 \Rightarrow w_2) \Rightarrow ((w_0 \mathcal{U} w_1) \Rightarrow (w_0 \mathcal{U} w_2))$
34	$(\Box w_1 \wedge \bigcirc w_2) \Rightarrow \bigcirc(w_1 \wedge w_2)$
35	$(\Box w_1 \wedge \Diamond w_2) \Rightarrow \Diamond(w_1 \wedge w_2)$
36	$(\Box w_1 \wedge (w_2 \mathcal{U} w_3)) \Rightarrow (w_1 \wedge w_2) \mathcal{U} (w_1 \wedge w_3)$
37	$(w \wedge \Box(w \Rightarrow \bigcirc w)) \Rightarrow \Box w$
38	$(w \wedge \Diamond \neg w) \Rightarrow \Diamond(w \wedge \bigcirc \neg w)$
39	$(\Diamond w_1 \wedge \Diamond w_2) \Rightarrow (\Diamond(w_1 \wedge \Diamond w_2) \vee \Diamond(w_2 \wedge \Diamond w_1))$
40	$\Box w \Leftrightarrow w \wedge \bigcirc \Box w$
41	$\Diamond w \Leftrightarrow w \vee \bigcirc \Diamond w$
42	$w_1 \mathcal{U} w_2 \Leftrightarrow w_2 \vee (w_1 \wedge \bigcirc(w_1 \mathcal{U} w_2))$
43	$(\neg w \mathcal{U} w) \Rightarrow \Diamond w$
44	$(\Box w_1 \wedge \Diamond w_2) \Rightarrow (w_1 \mathcal{U} w_2)$
45	$((w_1 \Rightarrow w_2) \mathcal{U} w_3) \Rightarrow ((w_1 \mathcal{U} w_3) \Rightarrow (w_2 \mathcal{U} w_3))$
46	$((w_1 \mathcal{U} w_2) \wedge (\neg w_2 \mathcal{U} w_3)) \Rightarrow (w_1 \mathcal{U} w_3)$
47	$(w_1 \mathcal{U} (w_2 \wedge w_3)) \Rightarrow ((w_1 \mathcal{U} w_2) \mathcal{U} w_3)$
48	$((w_1 \mathcal{U} w_2) \mathcal{U} w_3) \Rightarrow ((w_1 \vee w_2) \mathcal{U} w_3)$
49	$(\Diamond w_1 \wedge \Diamond w_2) \Rightarrow ((\neg w_1 \mathcal{U} w_2) \vee (\neg w_2 \mathcal{U} w_1))$

Table 2: Formulae used in Examples

B Results

Statistics relating to the number of rules removed during rule reduction and the time taken to detect loops for the examples are given in Table 3. We only present the data for examples where the number of rules (before rule reduction) is greater than 10. The complete set of results may be found in (Dixon, 1995). Table 3 shows the number of global $\square$–rules before temporal resolution takes place, the number global $\square$–rules after the rule reduction algorithm has been carried out, timings for loop detection with and without rule reduction and the time for rule reduction alone for selected examples. The times are given to the nearest 10 milliseconds. The data presented appears slightly different to that shown in (Dixon, 1995). This is because the timings have been rounded to the nearest 10 milliseconds in this presentation to avoid the effects of small timing errors. Where rule reduction was employed on the rule-set, the loop times given include the time for the rule reduction. Examples where there are different eventualities shown mean that loops are detected for each eventuality and examples where an eventuality occurs more than once means that temporal resolution must be applied more than once for this eventuality.

Eg.	$\diamondsuit$	Number of Rules: Loop Search	Number of Rules: Loop Search with Rule Red.	Time: Loop Search	Time: Loop Search with Rule Red.	Time: Rule Red. Only	Percentage Improvement
11	t_5	16	4	40	40	30	0
	t_6	32	6	90	70	50	22
12	t_5	24	5	60	60	30	0
13	t_5	24	5	60	50	30	17
14	t_4	32	6	80	70	60	13
15	t_1	32	5	70	80	50	(14)
16	w_2	62	16	1310	1130	160	14
	t_2	107	13	550	460	340	16
17	$\neg w_1$	12	5	30	30	20	0
	$\neg w_2$	26	5	70	70	40	0
	t_7	40	5	120	100	70	17
18	w_1	12	5	40	30	20	25
	w_2	26	5	60	60	40	0
	t_7	40	5	110	110	70	0
23	w_3	42	12	1310	370	90	72
24	t_7	47	20	1860	1850	90	1
	w_2	166	20	1370	1220	500	11
	w_3	172	16	910	690	520	24
27	w_3	13	5	80	80	20	0
28	t_4	16	7	110	80	20	27
32	w_3	12	4	90	70	30	22
33	w_1	13	4	70	90	20	(29)
36	w_3	14	4	90	70	20	22
39	w_2	30	9	170	110	60	35
	w_1	90	12	1260	370	230	71
45	w_3	14	4	80	50	30	38
46	w_2	14	4	100	80	40	20
47	t_3	19	6	200	130	40	35
	t_3	31	5	140	100	70	29
48	w_3	13	3	110	60	30	45

Table 3: Times for Temporal Resolution with and without Rule Reduction