An Algorithm for Guiding Temporal Resolution[⋆]

M. Carmen Fernández Gago, Michael Fisher, and Clare Dixon

Department of Computer Science, University of Liverpool, L69 7ZF, United Kingdom
{M.C.Gago, M.Fisher, C.Dixon}@csc.liv.ac.uk

Abstract. The clausal resolution method developed for discrete temporal logics involves translation to a normal form, classical resolution on formulae within states (termed step resolution) and temporal resolution between states. Step resolution may generate an unnecessarily large set of clauses. In addition, the most expensive part of the method is the application of the temporal resolution operation. In this paper we develop an algorithm to guide the search for the set of clauses needed for the application of temporal resolution. The algorithm is based on the outputs of a refined temporal resolution rule which allows us to generate temporal resolvents earlier within the process. In particular, this can also help us to avoid unnecessary step resolution and focus search for the most relevant clauses.

1 Introduction

The effective mechanisation of temporal logic is vital to the application of temporal reasoning in many fields, for example the verification of reactive systems [11], the implemention of temporal query languages [4], and temporal logic programming citeAbadiManna89-templog. Consequently, a range of proof methods have been developed, implemented and applied. The development of proof methods for temporal logic has followed three main approaches: tableau [17], automata [15] and resolution [2,3,9, 16]. Resolution based methods have the advantage that, as in the classical case [13], a range of strategies can potentially be used. Also the complexity of satisfiability of PTL, the logic used in this paper, is PSPACE-complete [14] which motivates the need for strategies to guide proof search.

In classical resolution a popular strategy has been the set of support strategy [18], which restricts the application of the resolution rule, pruning the search space. Our general aim here is to develop a set of support strategy to temporal resolution. We have extended this strategy for PTL without eventualities ('$\Diamond$', meaning sometime in the future) as this follows from the classical case. The extension of the strategy to full PTL is non trivial and we intend to achieve it using the algorithm which we propose in this paper together with the revised temporal resolution rule presented.

Throughout, we use a proof method for temporal logics based upon the use of clausal resolution [9]. The resolution procedure is characterised by the translation to a normal form, the application of a classical style resolution between formulae that occur at the same moment in time (*step resolution*), together with a novel *temporal*

⋆ This work was partially supported by EPSRC grant GR/M44859.

resolution rule, which derives contradictions over temporal sequences. Although the clausal temporal resolution method has been defined, proved correct and implemented, it sometimes generates an unnecessarily large set of formulas that may be irrelevant to the refutation. Not only that, but temporal resolution operations occur only after all step resolution inferences have been carried out. Thus no search for temporal resolution can be made earlier in the process. This means that in cases where a large amount of step resolution can occur the method may be very expensive.

The temporal resolution operation requires us to search for a set of clauses which satisfy a specified condition. As the search for candidates for this operation is the most expensive part but, is certain to be required for the temporal resolution process, our intention is to try to avoid all unnecessary step resolution operations and apply temporal resolution earlier in the process. In this sense, we propose an algorithm (and show soundness, completeness and termination of it) based on the outputs of a revised temporal resolution rule [8]. This temporal resolution rule allows us to generate temporal resolvents earlier, with the resolvents generated then being used in order to guide further search.

The structure of the paper is as follows. In section 2 we define the temporal logic considered, namely Propositional Temporal logic (PTL) [10]. In section 3 we review the basic resolution method while in section 4 we introduce the refinement of the temporal resolution rule which will be used in the rest of the paper. In the subsequent sections we propose the algorithm for guiding the temporal resolution search and give an example of its use.

2 Syntax and Semantics of PTL

In this section we present the syntax and semantics of a discrete, linear temporal logic with finite past and infinite future (PTL). The future-time connectives that we use include '$\diamondsuit$' (*sometime in the future*), '$\bigcirc$' (*in the next moment in time*), '$\square$' (*always*) '$\mathcal{U}$' (*until*), and '$\mathcal{W}$' (*unless, or weak until*). A choice for domain in which to interpret such temporal connectives is $\mathbb{N}$, i.e., the Natural Numbers ordered by the usual 'less than' relation.

Formulae are constructed using the following connectives and proposition symbols.
- A set, $\mathcal{P}$, of propositional symbols.
- Nullary connectives: **true** and **false**.
- Propositional connectives: $\neg$, $\vee$, $\wedge$ and $\Rightarrow$
- Temporal connectives: $\bigcirc$, $\diamondsuit$, $\square$, $\mathcal{U}$, $\mathcal{W}$ and **start**.

The set of well-formed formulae of PTL[1], denoted by WFF_p, is defined as follows:
- Any element of $\mathcal{P}$ is in WFF_p.
- **true**, **false** and **start** are in WFF_p.
- If ϕ and ψ are in WFF_p then so are
$\neg\phi$, $\phi \vee \psi$, , $\phi \wedge \psi$, $\phi \Rightarrow \psi$, $\diamondsuit\phi$, $\square\phi$, $\phi\mathcal{U}\psi$, $\phi\mathcal{W}\psi$, $\bigcirc\phi$

We define a model, M, for PTL as a structure $\langle \mathbb{N}, <, \pi_p \rangle$ where $\pi_p : \mathbb{N} \times \mathcal{P} \to T, F$ is a function assigning T or F to each atomic proposition at each moment in time.

[1] As usual, parentheses are also allowed to avoid ambiguity

As usual we define the semantics of the language via the satisfaction relation '$\models$'. For PTL, this relation holds between pairs of the form $\langle M, u\rangle$ (M is a model and u$\in$ $\mathbb{N}$) and well-formed formulae. The rules defining the satisfaction relation are as follows

$\langle M,u\rangle \models p$ iff $\pi_p(u,p) = T$ (where $p \in \mathcal{P}$)
$\langle M,u\rangle \models \mathbf{true}$
$\langle M,u\rangle \models$ **start** iff $u = 0$
$\langle M,u\rangle \models \phi \wedge \psi$ iff $\langle M,u\rangle \models \phi$ and $\langle M,u\rangle \models \psi$
$\langle M,u\rangle \models \neg\phi$ iff $\langle M,u\rangle \not\models \phi$
$\langle M,u\rangle \models \bigcirc\phi$ iff $\langle M,u+1\rangle \models \phi$
$\langle M,u\rangle \models \Box\phi$ iff for all $j \in \mathbb{N}$, if $j \geq u$ then $\langle M,u\rangle \models \phi$
$\langle M,u\rangle \models \phi\mathcal{U}\psi$ iff there exists a $k \in \mathbb{N}$, s.t. $k \geq u$ and $\langle M,k\rangle \models \psi$ and
for all $j \in \mathbb{N}$, if $i \leq j < k$ then $\langle M,j\rangle \models \phi$

We define '$\vee$', '$\Rightarrow$', '$\Leftrightarrow$', '$\Diamond$' and '$\mathcal{W}$' as follows:

$$\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi) \quad \phi \Leftrightarrow \psi \equiv (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi) \quad \phi\mathcal{W}\psi \equiv (\phi\mathcal{U}\psi) \vee \Box\phi$$
$$\phi \Rightarrow \psi \equiv \neg\phi \vee \psi \quad \Diamond\phi \equiv \neg\Box\neg\phi \quad \textbf{false} \equiv \neg\ \textbf{true}$$

3 Clausal Resolution Method for PTL

Discrete temporal logics are diffcult to reason about principally because we must make sure that formulae being resolved holds at the same moment in time. Further problems occur due to the inductive interaction between the $\Box$-operator and $\bigcirc$-operator. The resolution method described in [9] addresses these problem, by utilising a normal form, called Separated Normal Form (SNF), which separates out complex formulae from their contexts through the use of *renaming* [12], and a new temporal resolution operation introduced specifically for formulae in the normal form.

3.1 Separated Normal Form

The resolution method depends on formulae being transformed into a normal form (SNF), inspired by Gabbay's separation result [10]. The normal form, which is presented in [7], comprises formulae that are implications with present-time formulae on the left-hand side and (present or) future-time formulae on the right-hand-side. The transformation of formulae into SNF depends on three main operations: the renaming of complex subformulae; the removal of temporal operators; and classical style rewrite operations. In this section we review SNF but do not consider the transformation procedure (we note that the transformation to SNF preserves satisfiability [7]).
Formulae in SNF are of the general form $\Box \bigwedge_i (\phi_i \Rightarrow \psi_i)$, where each $\phi_i \Rightarrow \psi_i$ is known as a *clause* and is one of the following forms

start $\Rightarrow \bigvee\limits_c l_c$ (an *initial* clause)

$\bigwedge\limits_a k_a \Rightarrow \bigcirc \bigvee\limits_d l_d$ (a *step* clause)

$\bigwedge\limits_b k_b \Rightarrow \Diamond l$ (a *sometime* clause)

where each k_a, k_b, l_c, l_d and l represent literals.

3.2 Resolution Operations

Step resolution consists of the application of the standard classical resolution rule in two different contexts. Pairs of initial or step clauses may be resolved as follows:

$$\frac{\begin{array}{l}\mathbf{start} \Rightarrow \psi_1 \vee l \\ \mathbf{start} \Rightarrow \psi_2 \vee \neg l\end{array}}{\mathbf{start} \Rightarrow \psi_1 \vee \psi_2} \qquad \frac{\begin{array}{r}\phi_1 \Rightarrow \bigcirc(\psi_1 \vee l) \\ \phi_2 \Rightarrow \bigcirc(\psi_2 \vee \neg l)\end{array}}{(\phi_1 \wedge \phi_2) \Rightarrow \bigcirc(\psi_1 \vee \psi_2)}$$

The *simplification operations* are similar to those used in the classical case, consisting of both simplification and subsumption. An additional operation is required when a contradiction in a state is produced:

$$\frac{\phi \Rightarrow \bigcirc \mathbf{false}}{\begin{array}{l}\mathbf{start} \Rightarrow \neg\phi \\ \mathbf{true} \Rightarrow \bigcirc\neg\phi\end{array}}$$

This means that, if a formula ϕ leads to a contradiction in the next moment, then ϕ must never be satisfied.
Temporal resolution operations resolve one sometime clause with a set of step clauses [9] as follows:

$$\frac{\begin{array}{rcl}\phi_1 & \Rightarrow & \bigcirc\psi_1 \\ \vdots & \vdots & \vdots \\ \phi_n & \Rightarrow & \bigcirc\psi_n \\ \chi & \Rightarrow & \Diamond\neg l\end{array}}{\chi \Rightarrow (\neg \bigvee_{i=1}^{n} \phi_i)\mathcal{W}\neg l}$$

with the side condition that for all i, $1 \leq i \leq n$, then $\models \psi_i \Rightarrow l$ and $\models \psi_i \Rightarrow \bigvee_{i=1}^{n} \phi_i$, from which we can derive $\bigwedge_{i=1}^{n}(\phi_i \Rightarrow \bigcirc(l \wedge \bigvee_{j=1}^{n} \phi_j))$. This side condition ensures that the set of $\phi_i \Rightarrow \bigcirc\psi_i$ clauses together imply $\bigvee_{i=1}^{n} \phi_i \Rightarrow \bigcirc \Box l$. Such a set of clauses is known as a *loop* in l. The resolvent produced includes a $\mathcal{W}$ operator that must be translated into SNF before any further resolution steps can be applied.
Termination. If **start**$\Rightarrow$ **false** is produced, the original formula is unsatisfiable and the resolution process terminates.
Correctness. The soundness and (refutation) completeness of the original temporal resolution method have been established in [9].

4 A Refined Rule

In this paper we use a version of the temporal resolution operation described in [8]. The basic idea behind this operation is that, rather than insisting we already have a set of clauses that characterise a loop, we derive a more complex resolvent that allows for the possibility that such a loop does not exist. If the set of clauses chosen do comprise a loop, then the new resolvent turns out to be the resolvent from the original temporal resolution rule. More importantly, using this revised temporal resolution rule, if the clauses chosen do not comprise a loop, we will show, in this paper, how the resolvents produced can be used to guide further resolution in such a way that allows us to detect a loop if possible.

This new temporal resolution rule is

$$\frac{\begin{array}{ccc} \phi_1 & \Rightarrow & \bigcirc\psi_1 \\ \vdots & \vdots & \vdots \\ \phi_n & \Rightarrow & \bigcirc\psi_n \\ \chi & \Rightarrow & \Diamond\neg l \end{array}}{\chi \Rightarrow \left[\Box(\bigwedge_{i=1}^{n}(\phi_i \Rightarrow \bigcirc(l \wedge \bigvee_{j=1}^{n}\phi_j))) \Rightarrow (\neg\bigvee_{i=1}^{n}\phi_i)\mathcal{W}\neg l \right]}$$

Notice that there is no side-condition. This has been incorporated into the resolvent, which can be further transformed using classical manipulation to

$$\chi \Rightarrow \left[(\Diamond\neg\bigwedge_{i=1}^{n}(\phi_i \Rightarrow \bigcirc(l \wedge \bigvee_{j=1}^{n}\phi_j))) \vee (\neg\bigvee_{i=1}^{n}\phi_i)\mathcal{W}\neg l \right]$$

This new resolvent can be seen to introduce a new eventuality, $\Diamond\neg$**looping**, (where **looping** is the side condition for the original resolution rule), that effectively provides a check to establish that an appropriate loop was chosen. If an appropriate loop is not present, then this check will fail.

Once we rename $\Diamond\neg$**looping** as $\Diamond s$ and $s \Rightarrow \neg$**looping** then step resolution will give us **true**$\Rightarrow \bigcirc\neg s$ if we have chosen the correct loop (as **looping** is valid. This will be proved below) and the *temporal check* (essentially a minimal form of temporal resolution) will take place between the clauses $\Diamond s$ and **true**$\Rightarrow \bigcirc\neg s$.

As the choice of ϕ_i in temporal resolution is crucial, our main aim is to guide the search for a loop using this new temporal resolution rule. More specifically, we will use the output of a failed temporal check process to guide the subsequent search for loops. We proceed as follows. We guess a formula for the ϕ_i such that if our guess is correct, that is $\bigvee_{i=1}^{n}\phi_i \Rightarrow \bigcirc\Box l$, then the loop search phase terminates. Otherwise we use output from subsequent resolution steps to make a new guess based on the previous one. We show that this process terminates with the detection of a loop if one exists.

We assume that correct loops are in their simplest form, that is, factoring rule (similar to the classical case) and sbsumption have been applied wherever possible.

5 Algorithm

In this section we propose an algorithm to guide the search for a loop based on the temporal resolution rule introduced in section 4. For each eventuality $\Diamond\neg l$ occuring on the right hand side of a sometime clause, the algorithm constructs a sequence of DNF formulae, G_i, using the previous one and F_j, where F_j are disjunctions of literals which appear in the outputs of the revised temporal resolution rule. The algorithm is the following.

1. Choose $G_{-1} = \textbf{true}$.
2. Given G_i apply the revised temporal resolution rule to it.
3. For all clauses **true** $\Rightarrow \bigcirc(\neg s \vee F_j)$, $1 \leq j \leq m$ obtained during the last proof attempt, then G_{i+1} is obtained as $G_{i+1} = G_i \wedge (\bigvee_{j=1}^{m} \neg F_j)$.
4. Go to 2 until either
   (a) **true** $\Rightarrow \bigcirc\neg s$ is obtained and, hence, a loop has been found or
   (b) $G_{i+1} = \textbf{false}$, in which case we terminate without having found a loop.

6 Completeness

In the following we will prove completeness for this algorithm by relating it to the completeness of the Breadth-First Search Algorithm proposed in [5]. First, we introduce the Breadth-First Search Algorithm.

6.1 Breadth-First Search Algorithm

The Breadth-First Search Algorithm constructs a sequence of formulae, H_i for $i \geq 0$, that are formulae in Disjunctive Normal Form and contain no temporal operators. They are constructed from the conjunctions of literals on the left hand sides of step clauses or combinations of step clauses in the SNF-clause-set that satisfy certain properties (see below). Assuming we are resolving with $\Diamond\neg l$ each formula H_i satisfies $H_i \Rightarrow \bigcirc l$ and given H_i each new formula H_{i+1} satisfies $H_{i+1} \Rightarrow \bigcirc H_i$. When termination occurs we have $H_{i+1} \Leftrightarrow H_i$ so that $H_i \Rightarrow \bigcirc \Box l$ for resolution with $\Diamond\neg l$. We assume that all necessary step resolution has been carried out.

Breadth-First Search Algorithm For each eventuality $\Diamond\neg l$ occuring on the right hand side of a sometime clause do the following.

1. Search for all the step clauses of the form $C_k \Rightarrow \bigcirc l$, for $k = 0$ to b (called *start PTL-clauses*), disjoin the left hand sides and generate the *top formula* H_0 equivalent to this, i.e.

   $$H_0 \Leftrightarrow \bigvee_{k=0}^{b} C_k.$$

   Simplify H_0. If $\models H_0$ we terminate having found a loop-formula (**true**).

2. Given formula H_i, build formula H_{i+1} for $i = 0, 1, \ldots$ by looking for step clauses or combinations of clauses of the form $A_j \Rightarrow \bigcirc B_j$, for $j = 0 mboxtoc$ where $\models B_j \Rightarrow H_i$ and $\models A_j \Rightarrow H_0$. Disjoin the left hand sides so that

$$H_{i+1} \Rightarrow \bigvee_{j=0}^{c} A_j$$

and simplify as previously.
3. Repeat (2) until
   (a) $\models H_i$. We terminate having found a loop-formula and return **true**.
   (b) $\models H_i \Leftrightarrow H_{i+1}$. We terminate having found a loop-formula and return the DNF formula H_i.
   (c) The new formula is empty. We terminate without having found a loop-formula.

Soudness, Completeness and Termination for the BFS-algorithm Given a set of SNF clauses R, that contains a loop $A \Rightarrow \bigcirc \Box l$, applying BFS algorithm will output a DNF formula A' such that $A' \Rightarrow \bigcirc \Box l$ and $A \Rightarrow A'$. Termination is also established.[5]

6.2 Completeness of the New Algorithm

To show the completeness of the algorithm we will prove that for all $i \geq 0, G_i \Leftrightarrow H_i$ by induction. Let R be a set of SNF-clauses and $\Diamond \neg l$ be the right hand side of a sometime clause, we assume that R contains a loop in l.

Lemma 1. $G_0 \Leftrightarrow H_0$

Proof (Outline). In order to obtain G_0, according to the algorithm, the revised temporal resolution rule must be applied to G_{-1}. Some of the clauses derived during the proof are **true** $\Rightarrow \bigcirc(\neg s \vee$ **true**$)$ and $s \Rightarrow \bigcirc(\neg l \vee \neg$**true**$)$, which are the clauses **true** $\Rightarrow \bigcirc$**true** and $s \Rightarrow \bigcirc \neg l$ (1).

As R contains a loop, in the initial set of clauses there must be some clauses such that they may be resolved together to obtain $A_i \Rightarrow \bigcirc l$, $1 \leq i \leq k$. By resolution with clause (1) the resolvents are $s \wedge A_i \Rightarrow \bigcirc$**false**, $1 \leq i \leq k$, and by applying simplification to the above clauses, **true** $\Rightarrow \bigcirc(\neg s \vee \neg A_i)$, $1 \leq i \leq k$, is obtained. These latter clauses are used to obtain G_0 as follows: $G_0 =$ **true** $\wedge (A_1 \vee \ldots \vee A_k) = A_1 \vee \ldots \vee A_k$. In building H_0 by Breadth-First Search Algorithm, the left hand sides of the clauses $A_i \Rightarrow \bigcirc l$ are disjoined, giving, $H_0 \Leftrightarrow A_1 \vee \ldots \vee A_k$. Hence, $G_0 \Leftrightarrow H_0$

Theorem 1. *For all* $i \in \mathbb{N}$ $H_{i+1} \Leftrightarrow G_{i+1}$.

Proof (Outline).
Base case: By Lemma 1 $H_0 \Leftrightarrow G_0$.
Induction case: We assume $H_k \Leftrightarrow G_k$ for all $k \leq i, k \in \mathbb{N}$ and we prove the hypothesis for $i+1$. We know the following valid statements about H_{i+1} from the definition of the Breadth First Search algorithm:

$(a).\ H_{i+1} \Rightarrow \bigcirc H_i$ $(c).\ \ G_i \ \Leftrightarrow H_i$ (Induction Hypothesis)
$(b).\ H_{i+1} \Rightarrow \ \bigcirc l$ $\ (d).\ H_{i+1} \Rightarrow H_i$

Because of the translation into the normal form of the resolvent obtained from applying the new temporal resolution rule to G_i, we obtain the clauses
1. $s \Rightarrow \bigcirc(\neg l \vee \neg G_i)$ and 2. **true** $\Rightarrow \bigcirc(\neg s \vee G_i)$.
Using property (c) these two clauses are transformed to
3. $s \Rightarrow \bigcirc(\neg l \vee \neg H_i)$ and 4. **true** $\Rightarrow \bigcirc(\neg s \vee H_i)$
Then, applying step resolution, we obtain:
5. $s \wedge H_{i+1} \Rightarrow \bigcirc\neg H_i$ [Step Resolution b,3]
6. $s \wedge H_{i+1} \Rightarrow \bigcirc$**false** [Step Resolution 5,a]
7. **true** $\Rightarrow \bigcirc(\neg s \vee \neg H_{i+1})$ [Simp.6]
In order to obtain G_{i+1} the algorithm is applied, where the clauses **true** $\Rightarrow \bigcirc(\neg s \vee F_j)$ considered in this case are 4 and 7.
$G_{i+1} = G_i \wedge [\neg H_i \vee H_{i+1}] = G_i \wedge [\neg G_i \vee H_{i+1}] = (G_i \wedge \neg G_i) \vee (G_i \wedge H_{i+1}) =$ **false** $\vee (H_i \wedge H_{i+1}) = H_i \wedge H_{i+1}$
and by property (d) $H_i \wedge H_{i+1} \Leftrightarrow H_{i+1}$, which means $G_{i+1} \Leftrightarrow H_{i+1}$

Theorem 2. *Let L be a DNF formula. Then by applying the new temporal resolution rule, the clause* **true** $\Rightarrow \bigcirc\neg s$ *is obtained if L is a loop.*

Proof (Outline). Let $L = D_1 \vee D_2 \vee ... \vee D_n$. We are assuming L is a loop, i.e., $\square[L \Rightarrow \bigcirc(L \wedge l)]$. After renaming, the clauses $t \Rightarrow \lozenge s$ and $s \Rightarrow \neg[L \Rightarrow \bigcirc(L \wedge l)]$ are obtained.

Because of the translation of the second clause into the normal form, the clauses produced are $s \Rightarrow L$ and $s \Rightarrow \bigcirc(\neg l \vee \neg L)$. The first clause produces **true** $\Rightarrow \bigcirc(\neg s \vee D_1 \vee ... \vee D_n)$ (1) and the second one, $s \Rightarrow \bigcirc(\neg l \vee \neg D_i)$ (B_i) $1 \leq i \leq n$.

As L is a loop, there must be a set of clauses in the initial set of clauses such that together represent $L \Rightarrow \bigcirc l$ and $L \Rightarrow \bigcirc L$. Clauses $B_1, ..., B_n$ together with the previous ones produce the clauses $s \wedge D_i \Rightarrow \bigcirc$**false**, $1 \leq i \leq n$. Applying simplification to these clauses we obtain **true** $\Rightarrow \bigcirc(\neg s \vee \neg D_1)$, $1 \leq i \leq n$. The previous clauses can be resolved with clause (1) producing **true**$\Rightarrow \bigcirc\neg s$.

Theorem 3. *If G_i is a loop, then $G_i = G_{i+1}$.*

Proof. By Theorem 2, if G_i is a loop, then **true** $\Rightarrow \bigcirc\neg s$ is generated, so according to the algorithm $G_{i+1} = G_i \wedge [\neg$**false**$] = G_i \wedge$ **true** $= G_i$

Theorem 4. *The algorithm terminates.*

Proof. 1. If there exists a loop in the initial set of clauses, then we know that Breadth Search Algorithm terminates finding a loop H_n. By Theorem 1 $H_n \Leftrightarrow G_n$, so G_n is a loop and then by Theorem 2 **true** $\Rightarrow \bigcirc\neg s$ is obtained.
2. If there does not exist a set of clauses which comprise a loop in the initial set of clauses, then at some point it will not be possible to produce the clause $s \wedge C_i \Rightarrow \bigcirc$**false** and therefore no clauses **true** $\Rightarrow \bigcirc(\neg s \vee F_i)$ will be derived during the proof apart from the clause **true** $\Rightarrow \bigcirc(\neg s \vee G_n)$, then $G_{n+1} = G_n \wedge [\neg G_n] =$ **false**

6.3 Example

Let the loop be $a \wedge b \wedge c \wedge d \Rightarrow \bigcirc \square l$ and the set of SNF clauses be as follows:

1. $a \Rightarrow \bigcirc l$	4. $d \wedge a \Rightarrow \bigcirc b$
2. $b \wedge c \Rightarrow \bigcirc d$	5. $a \wedge b \Rightarrow \bigcirc c$
3. $c \wedge d \Rightarrow \bigcirc a$	6. $\chi \Rightarrow \lozenge \neg l$

According to the algorithm the first choice is $G_{-1} = \textbf{true}$. So, some of the resolvents are

7. $\textbf{true} \Rightarrow \bigcirc(\neg s \vee \textbf{true})$
8. $s_{-1} \Rightarrow \bigcirc \neg l$
9. $s_{-1} \wedge a \Rightarrow \bigcirc \textbf{false}$ [SRES 1,8]
10. $\textbf{true} \Rightarrow \bigcirc(\neg s_{-1} \vee \neg a)$ [SIMP 9]

So, the next G_i will be, $G_0 = \textbf{true} \wedge a = a$.

11. $\textbf{true} \Rightarrow \bigcirc(\neg s_0 \vee a)$
12. $s_0 \Rightarrow \bigcirc(\neg l \vee \neg a)$
13. $s_0 \wedge a \wedge c \wedge d \Rightarrow \bigcirc \textbf{false}$ [1,3,12]
14. $\textbf{true} \Rightarrow \bigcirc(\neg s_0 \vee \neg a \vee \neg c \vee \neg d)$ [SIMP 13]
15. $\textbf{true} \Rightarrow \bigcirc(\neg s_0 \vee \neg c \vee \neg d)$ [SRES 11,14]

Clause 15 subsumes clause 14, so $G_1 = a \wedge [\neg a \vee (c \wedge d)] = a \wedge c \wedge d$.

16. $\textbf{true} \Rightarrow \bigcirc(\neg s_1 \vee a)$
17. $\textbf{true} \Rightarrow \bigcirc(\neg s_1 \vee c)$
18. $\textbf{true} \Rightarrow \bigcirc(\neg s_1 \vee d)$
19. $s_1 \Rightarrow \bigcirc(\neg l \vee \neg a \vee \neg c \vee \neg d)$
20. $s_1 \wedge a \wedge b \wedge c \wedge d \Rightarrow \bigcirc \textbf{false}$ [SRES 1,2,3,5,19]
21. $\textbf{true} \Rightarrow \bigcirc(\neg s_1 \vee \neg a \vee \neg b \vee \neg c \vee \neg d)$ [SIMP 20]
22. $\textbf{true} \Rightarrow \bigcirc(\neg s_1 \vee \neg b)$ [SRES 16,17,18,21]

Clause 22 subsumes clause 21. According to the algorithm
$G_2 = (a \wedge c \wedge d) \wedge (\neg a \vee \neg c \vee \neg d \vee b) = \textbf{false} \vee \textbf{false} \vee \textbf{false} \vee (a \wedge c \wedge d \wedge b) = (a \wedge c \wedge d \wedge b)$, which is the correct loop, so if the algorithm is applied again $\textbf{true} \Rightarrow \bigcirc \neg s$ will be obtained, which means termination.

7 Conclusions and Future Work

In this paper we have presented an algorithm based upon a refined clausal temporal resolution rule. This new rule has not only allowed us to develop such an algorithm, but allows more flexibility in the search for a refutation.

The algorithm uses outputs from the previous attempt to select a loop to guide the choice for the next guess and its correctenss is shown with respect to an existing loop search algorithm.

In the future we intend to apply these results to the development of strategies for temporal resolution that allow us to reduce the search space. In particular, we are interested in incorporating the set of support strategy [18,6], which has been pariculary successful in the classical case. The algorithm presented in this paper provides a basis for such a strategy and allows the flexible interleaving of step resolution and loop search operations.

As our main aim is to develop a set of support strategy for the temporal case, an important task to carry out will be to combine and to compare such a strategy with alternative proof strategies and to examine the practical efficiency of such combined strategies. In this future work the results presented in this paper will be essential.

References

1. M. Abadi and Z. Manna. Temporal Logic Programming. *Journal of Symbolic Computation*, 8: 277–295, 1989.
2. M. Abadi and Z. Manna. Nonclausal Deduction in First-Order Temporal Logic. *ACM Journal*, 37(2):279–317, April 1990.
3. A. Cavalli and L. Farinas 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 *LNCS*, pages 113–127. Springer-Verlag, 1984.
4. J. Chomicki and D. Niwinski. On the Feasibility of Checking Temporal Integrity Constraints. *Journal of Computer and System Sciences*, 51(3):523–535, 1995.
5. C. Dixon. Temporal Resolution using a Breadth-First Search Algorithm. *Annals of Mathematics and Artificial Intelligence*, 22:87–115, 1998.
6. C. Dixon and M. Fisher. The Set of Support Strategy in Temporal Resolution. In *Proceedings of TIME-98 the Fifth Workshop on Temporal Representation and Reasoning*, Sanibel Island, Florida, May 1998. IEEE Computer Society Press.
7. M. Fisher. A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. *Journal of Logic and Computation*, 7(4):429–456, 1997.
8. M. Fisher and C. Dixon. Guiding Clausal Temporal Resolution. In *Advances in Temporal Logic*, volume 16 of *Applied Logic Series*, pages 167–184. Kluwer, 2000.
9. M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. In *Transactions on Computational Logic 2(1)*. January 2001.
10. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. The Temporal Analysis of Fairness. In *Proceedings of the 7th ACM Symposium on the Principles of Programming Languages*, pages 163–173, Las Vegas, Nevada, January 1980.
11. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.
12. D. A. Plaisted and S. A. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.
13. J. A. Robinson. A Machine–Oriented Logic Based on the Resolution Principle. *ACM Journal*, 12(1):23–41, January 1965.
14. A. P. Sistla and E. M. Clarke. Complexity of propositional linear temporal logics. *ACM Journal*, 32(3):733–749, July 1985.
15. A. P. Sistla, M. Vardi, and P. Wolper. The Complementation Problem for Buchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49:217–237, 1987.
16. G. Venkatesh. A Decision Method for Temporal Logic based on Resolution. *Lecture Notes in Computer Science*, 206:272–289, 1986.
17. Pierre Wolper. The Tableau Method for Temporal Logic: An overview. *Logique et Analyse*, 110–111:119–136, June-Sept 1985.
18. L. Wos, G. Robinson, and D. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *ACM Journal*, 12:536–541, October 1965.