Algorithms for Guiding Clausal Temporal Resolution*

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

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

Abstract. Clausal temporal resolution is characterised by a *translation* of the formulae whose satisfiability is to be established to a normal form, *step* resolution (similar to classical resolution) on formulae occurring at the same states and *temporal* resolution between formulae describing properties over a longer period. The most complex part of the method occurs in searching for candidates for the temporal resolution operation, something that may need to be carried out several times.
In this paper we consider a new technique for finding the candidates for the temporal resolution operation. Although related to the previously developed external search procedure, this new approach not only allows the temporal resolution operation to be carried out at *any* moment, but also simplifies any subsequent search required for similar temporal formulae.
Finally, in contrast with previous approaches, this search can be seen as an inherent part of the resolution process, rather than an external procedure that is only called in certain situations.

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 [12], the implementation of temporal query languages [4], and temporal logic programming [1]. 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 [16], automates [14] and resolution [2, 3, 10, 15], the approach adopted here. Resolution based methods have the advantage that, as in the classical case [13], a range of strategies can be used.

A particularly successful strategy for classical resolution has been the set of support strategy [17], which restricts the application of the resolution rule, pruning the search space. Our aim is to develop a set of support (SOS) strategy for propositional temporal logic, PTL, the logic used in this paper. The extension of the SOS strategy for the fragment of PTL without eventualities (clauses involving the operator '$\Diamond$', meaning sometime in the future) has been achieved

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

[7] using techniques developed from SOS for classical logic. The definition of the strategy for full PTL is non trivial and we intend to achieve it using the algorithms proposed in this paper together with the strategy defined for the case without eventualities.

Clausal temporal resolution [10] is characterised by the translation to a normal form, the application of classical style resolution between formulae that occur at the same moment in time (*step resolution*), together with a novel *temporal 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 many step resolution inferences have been carried out. This means that, in cases where a large amount of step resolution can occur, the method may be very expensive.

As the search for the candidates for the temporal resolution operation is the most expensive part of the method we need to guide it and, if possible, avoid much unnecessary subsequent step resolution. In this sense, we propose an algorithm based on step resolution to guide the search. In this approach, we choose a candidate formula for the temporal resolution operation and we check whether such a candidate is appropriated to perform the resolution operation. Our intention is to re-use as much information as possible in those cases where further searches are required. Thus, we propose a second algorithm which is based on the original one and is used to guide further searches. The structure of the paper is as follows. In Section 2 we define the temporal logic considered, namely Propositional Temporal Logic [11]. In Section 3 we review the basic resolution method. In Section 4 we describe an algorithm to find candidates for the temporal resolution operation using only step resolution. Its completeness is shown in Section 5. In Section 6 we propose a second algorithm algorithm for the cases when further searches are needed. Completeness is also shown in this section.

2 Syntax and Semantics of PTL

In this section we present the syntax and semantics of (PTL), based on a discrete, linear temporal logic with finite past and infinite future. The future-time connectives that we use include '$\Diamond$' (*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 interpreting such temporal connectives is $(\mathbb{N}, <)$, i.e., the Natural Numbers ordered by the usual 'less than' relation.

2.1 Syntax

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

- A set, $\mathcal{P}$, of propositional symbols.

- Nullary connectives: **true** and **false**.
- Propositional connectives: $\neg$, $\vee$, $\wedge$, $\Rightarrow$ and $\Leftrightarrow$.
- Temporal connectives: $\bigcirc$, $\Diamond$, $\Box$, $\mathcal{U}$, and $\mathcal{W}$ and the nullary temporal connective **start**.

The set of well-formed formulae of PTL[1], denoted by WFF_p, is defined as the set satisfying:

- 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 $\neg\phi$, $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \Rightarrow \psi$, $\phi \Leftrightarrow \psi$, $\Diamond\phi$, $\Box\phi$, $\phi\mathcal{U}\psi$, $\phi\mathcal{W}\psi$, $\bigcirc\phi$

2.2 Semantics

We define a model, M, for PTL as a structure $\langle \mathcal{D}, R, \pi_p \rangle$ where

- $\mathcal{D}$ is the temporal domain, e.g, the natural numbers and
- R is the ordering relation, e.g. $<$.
- $\pi_p : \mathcal{D} \times \mathcal{P} \rightarrow \{T, F\}$ is a function assigning T or F to each atomic proposition at each moment in time.

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.

$$
\begin{array}{lll}
\langle M,u \rangle \models p & \text{iff } \pi_p(u,p) = T & (\text{where } p \in \mathcal{P}) \\
\langle M,u \rangle \models \textbf{true} & & \\
\langle M,u \rangle \not\models \textbf{false} & & \\
\langle M,u \rangle \models \textbf{start} & \text{iff } u = 0 & \\
\langle M,u \rangle \models \phi \wedge \psi & \text{iff } \langle M,u \rangle \models \phi \text{ and } \langle M,u \rangle \models \psi & \\
\langle M,u \rangle \models \phi \vee \psi & \text{iff } \langle M,u \rangle \models \phi \text{ or } \langle M,u \rangle \models \psi & \\
\langle M,u \rangle \models \phi \Rightarrow \psi & \text{iff } \langle M,u \rangle \not\models \phi \text{ or } \langle M,u \rangle \models \psi & \\
\langle M,u \rangle \models \neg\phi & \text{iff } \langle M,u \rangle \not\models \phi & \\
\langle M,u \rangle \models \phi \Leftrightarrow \psi & \text{iff } \langle M,u \rangle \models \phi \Rightarrow \psi \text{ and } \langle M,u \rangle \models \psi \Rightarrow \phi & \\
\langle M,u \rangle \models \bigcirc\phi & \text{iff } \langle M,u+1 \rangle \models \phi & \\
\langle M,u \rangle \models \Diamond\phi & \text{iff there exists a } k \in \mathbb{N} \text{ such that } k \geq u \ \langle M,k \rangle \models \phi & \\
\langle M,u \rangle \models \Box\phi & \text{iff for all } j \in \mathbb{N}, \text{ if } j \geq u \text{ then } \langle M,j \rangle \models \phi & \\
\langle M,u \rangle \models \phi\mathcal{U}\psi & \text{iff there exists a } k \in \mathbb{N}, \text{ s.t. } k \geq u \text{ and } \langle M,k \rangle \models \psi \text{ and} & \\
& \quad \text{for all } j \in \mathbb{N}, \text{ if } u \leq j < k \text{ then } \langle M,j \rangle \models \phi & \\
\langle M,u \rangle \models \phi\mathcal{W}\psi & \text{iff } \langle M,u \rangle \models \phi\mathcal{U}\psi \text{ or } \langle M,u \rangle \models \Box\phi &
\end{array}
$$

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

3 Clausal Resolution Method for PTL

The resolution method presented here is clausal, that means that to assure the validity of some PTL formula we negate it and translate into a normal form. Then, both step resolution and temporal resolution are applied. We terminate when either a contradiction has been derived or no new information can be derived.

3.1 Separated Normal Form

The resolution method depends on formulae being transformed into a normal form (SNF). The normal form, which is presented in [9], 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 [10]).

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

$$
\begin{array}{lll}
\mathbf{start} \Rightarrow \bigvee\limits_{c} l_c & & \text{(an \textit{initial} clause)} \\
\bigwedge\limits_{a} k_a \Rightarrow \bigcirc \bigvee\limits_{d} l_d & & \text{(a \textit{step} clause)} \\
\bigwedge\limits_{b} k_b \Rightarrow \Diamond l & & \text{(a \textit{sometime} clause)}
\end{array}
$$

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

To apply the temporal resolution operation described below, one or more step clauses may need to be combined. Then a variant on SNF called *merged-SNF* (*SNF_m*)[8] is also defined. Given a set of clauses in SNF, any clause in SNF is also a clause in SNF_m. Any two clauses in SNF_m may be combined to produce a clause in SNF_m as follows.

$$
\frac{\begin{array}{c} \phi_1 \Rightarrow \bigcirc \psi_1 \\ \phi_2 \Rightarrow \bigcirc \psi_2 \end{array}}{(\phi_1 \wedge \phi_2) \Rightarrow \bigcirc (\psi_1 \wedge \phi_2)}
$$

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:

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

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 temporal contradiction is produced:

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

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 merged step clauses [10] 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 $\models \psi_i \Rightarrow \bigvee_{j=1}^{n} \phi_j$, 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 $\phi_i \Rightarrow \bigcirc\psi_i$ merged 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 an $\mathcal{W}$ 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 both established in [10].

4 Algorithm for Searching for Loops

In order to apply the resolution rule presented in Section 3 a loop must be detected. Thus, given an eventuality $\Diamond\neg l$, our aim is to detect a set of merged step clauses that comprises a loop to be resolved with $\Diamond\neg l$.

4.1 Motivation

In [5] a breadth-search approach is used to detect loops. Although this algorithm is correct, in some cases, when further searches for loops need to be carried out, the information obtained in a previous search is not reused. Our approach here is based on step resolution and allows us to re-use previous search information. Assume we are searching for a loop in l, our search produces a sequence of *guesses*, G_i, which are DNF formulae. We show that these are equivalent to the DNF formulae H_i output by the Breadth-First Search algorithm (see [5]). In Breadth-First Search each new DNF formula H_{i+1} satisfies the property $H_{i+1} \Rightarrow \bigcirc(H_i \wedge l)$. Similarly we also have $G_{i+1} \Rightarrow \bigcirc(G_i \wedge l)$. In order to find G_{i+1} we add $\mathbf{true} \Rightarrow \bigcirc(\neg G_i \vee \neg l)$ to the original set of clauses and resolve. The left hand side of clauses $Z \Rightarrow \bigcirc\mathbf{false}$ satisfy $Z \Rightarrow \bigcirc(G_i \wedge l)$. As we want to save clauses derived during this process and possibly use them later, we add the clause $s_i^{\neg l} \Rightarrow \bigcirc(\neg G_i \vee \neg l)$ and thus search for clauses $s_i^{\neg l} \wedge Z \Rightarrow \bigcirc\mathbf{false}$, derived from resolving with $s_i^{\neg l} \Rightarrow \bigcirc(\neg G_i \vee \neg l)$ or its resolvents with other clauses which are rewritten as $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee \neg Z)$.

4.2 Step Loop Search Algorithm

In this section we propose an algorithm to search for a loop. For each eventuality $\lozenge\neg l$ occurring on the right hand side of a sometime clause, the algorithm constructs a sequence of DNF formulae, G_i, by using the previous guess together with F_j, where F_j are disjunctions of literals derived by the application of the algorithm to G_i. The algorithm is the following.

1. Choose $G_{-1} \Leftrightarrow \mathbf{true}$
2. Given a guess G_i add the clause $s_i^{\neg l} \Rightarrow \bigcirc(\neg G_i \vee \neg l)$ and apply Step Resolution.
3. For all clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee F_j)$ obtained during the generation of resolvents, let $G_{i+1} \Leftrightarrow G_i \wedge (\bigvee_{j=1}^{m} \neg F_j)$.
4. Go to 2 until either
   (a) $G_i \Leftrightarrow G_{i+1}$ (we terminate having found a loop).
   (b) G_{i+1} is empty. (we terminate without having found a loop).

4.3 Example

Let the loop be $(a \wedge b \wedge c \wedge d) \Rightarrow \bigcirc\,\square l$, derived from the following SNF clauses.

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

According to algorithm 1 the first guess is $G_{-1} \Leftrightarrow \mathbf{true}$. For such a guess we add the clause $s_{-1}^{\neg l} \Rightarrow \bigcirc(\mathbf{false} \vee \neg l)$. Some of the resolvents derived by applying step resolution among this clause and 1-6 are

$$
\begin{array}{lrll}
7. & s_{-1}^{\neg l} & \Rightarrow \bigcirc \neg l & \\
8. & s_{-1}^{\neg l} \wedge a & \Rightarrow \bigcirc \mathbf{false} & [1, 7] \\
9. & \mathbf{true} & \Rightarrow \bigcirc(\neg s_{-1}^{\neg l} \vee \neg a) & [\text{Simp.}8]
\end{array}
$$

Therefore, the next guess will be, $G_0 \Leftrightarrow \mathbf{true} \wedge a \Leftrightarrow a$.

$$
\begin{array}{lrll}
10. & s_0^{\neg l} & \Rightarrow \bigcirc(\neg l \vee \neg a) & \\
11. & s_0^{\neg l} \wedge a \wedge c \wedge d & \Rightarrow \bigcirc \mathbf{false} & [1, 3, 10] \\
12. & \mathbf{true} & \Rightarrow \bigcirc(\neg s_0^{\neg l} \vee \neg a \vee \neg c \vee \neg d) & [\text{Simp.}11]
\end{array}
$$

Next guess is $G_1 \Leftrightarrow a \wedge (a \wedge c \wedge d) \Leftrightarrow a \wedge c \wedge d$.

$$
\begin{array}{lrll}
13. & s_1^{\neg l} & \Rightarrow \bigcirc(\neg l \vee \neg a \vee \neg c \vee \neg d) & \\
14. & s_1^{\neg l} \wedge a \wedge b \wedge c \wedge d & \Rightarrow \bigcirc \mathbf{false} & [1, 2, 3, 5, 13] \\
15. & \mathbf{true} & \Rightarrow \bigcirc(\neg s_1^{\neg l} \vee \neg a \vee \neg b \vee \neg c \vee \neg d) & [\text{Simp.}14]
\end{array}
$$

According to the algorithm the next guess is

$$G_2 \Leftrightarrow (a \wedge c \wedge d) \wedge (a \wedge b \wedge c \wedge d) \Leftrightarrow a \wedge c \wedge d \wedge b$$

$$
\begin{array}{lrll}
16. & s_2^{\neg l} & \Rightarrow \bigcirc(\neg l \vee \neg a \vee \neg c \vee \neg d \vee \neg b) & \\
17. & s_2^{\neg l} \wedge a \wedge b \wedge c \wedge d & \Rightarrow \bigcirc \mathbf{false} & [1, 2, 3, 4, 5, 16] \\
18. & \mathbf{true} & \Rightarrow \bigcirc(\neg s_2^{\neg l} \vee \neg a \vee \neg b \vee \neg c \vee \neg d) & [\text{Simp.}17]
\end{array}
$$

If we apply the algorithm to G_2 then the next guess will again be

$$
\begin{array}{l}
G_3 \Leftrightarrow (a \wedge b \wedge c \wedge d) \wedge (a \wedge b \wedge c \wedge d) \Leftrightarrow (a \wedge b \wedge c \wedge d). \\
G_3 \Leftrightarrow G_2
\end{array}
$$

which means termination as $G_3 \Leftrightarrow G_2$ and so G_2 is a loop, i.e, $G_2 \Rightarrow \bigcirc \Box l$.

5 Completeness

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

5.1 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 $H_{i+1} \Rightarrow \bigcirc H_i$ and $H_{i+1} \Rightarrow H_i$. When termination occurs we have $H_{i+1} \Leftrightarrow H_i$ so that $H_i \Rightarrow \bigcirc \square l$ for resolution with $\diamondsuit \neg l$. The algorithm assumes that all necessary step resolution has been carried out.

Breadth-First Search Algorithm For each eventuality $\diamondsuit \neg l$ occurring 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, disjoin the left hand sides and generate the 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 $j = 0$ to c where $\models B_j \Rightarrow H_i$ and $\models A_j \Rightarrow H_0$. Disjoin the left hand sides so that $H_{i+1} \Leftrightarrow \bigvee_{j=0}^{c} A_j$ 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.

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

5.2 Completeness of the Step Loop Search Algorithm

To show the completeness of the new 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 $\diamondsuit \neg l$ 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. In order to obtain G_0, according to the algorithm, the clause $s_{-1}^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg \mathbf{true})$ is added, which is the clause $s_{-1}^{\neg l} \Rightarrow \bigcirc \neg l$ (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 $s_{-1}^{\neg l} \wedge A_i \Rightarrow \bigcirc \mathbf{false}$, $1 \leq i \leq k$ and by simplification $\mathbf{true} \Rightarrow \bigcirc(\neg s_{-1}^{\neg l} \vee \neg A_i)$.
These last clauses are used in order to obtain G_0 as $G_0 \Leftrightarrow \mathbf{true} \wedge (A_1 \vee ... \vee A_k) \Leftrightarrow A_1 \vee ... \vee A_k$.

For building H_0 by the Breadth-First Search Algorithm, the left hand sides of the clauses $A_i \Rightarrow \bigcirc l$ are disjoined, giving , $H_0 \Leftrightarrow A_1 \vee ... \vee A_k$, and so, $G_0 \Leftrightarrow H_0$ □

Theorem 1. *For all $n \in \mathbb{N}$ $H_n \Leftrightarrow G_n$.*

Proof. 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:

$$\begin{array}{lrcll} (a). & H_{i+1} & \Rightarrow & \bigcirc H_i & \\ (b). & H_{i+1} & \Rightarrow & \bigcirc l & \\ (c). & G_i & \Leftrightarrow & H_i & \text{(Induction Hypothesis)} \\ (d). & H_{i+1} & \Rightarrow & H_i & \end{array}$$

Assume we have generated guess G_i and we are about to derive G_{i+1}. From the algorithm the clause $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg G_i)$ is added. Using property (c) the clause 1 is transformed into

$$s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg H_i). \tag{1}$$

Then, applying step resolution, we obtain:

$$\begin{array}{lrcll} 3. & s_i^{\neg l} \wedge H_{i+1} & \Rightarrow & \bigcirc \neg H_i & [\text{b, 1}] \\ 4. & s_i^{\neg l} \wedge H_{i+1} & \Rightarrow & \bigcirc \mathbf{false} & [3,\text{a}] \\ 5. & \mathbf{true} & \Rightarrow & \bigcirc(\neg s_i^{\neg l} \vee \neg H_{i+1}) & [\text{Simp. } 4] \end{array}$$

In order to obtain G_{i+1} the algorithm is applied, where the clauses
$\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee F_j)$ considered in this case just consist of clause 5 and thus $G_{i+1} \Leftrightarrow G_i \wedge [H_{i+1}] \Leftrightarrow G_i \wedge H_{i+1} \Leftrightarrow H_i \wedge H_{i+1}$.
By property (d) $(H_i \wedge H_{i+1}) \Leftrightarrow H_{i+1}$, and then $G_{i+1} \Leftrightarrow H_{i+1}$ □

Theorem 2. *If G_i is a loop, then $G_i \Leftrightarrow G_{i+1}$.*

Proof. Let $G_i \Leftrightarrow D_1 \vee D_2 \vee ... \vee D_n$ be a loop.
As usual for the application of the algorithm the clause

$$s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg G_i) \tag{2}$$

is added.

As G_i is a loop, there must be a set of clauses in the initial set of clauses that together represent $G_i \Rightarrow \bigcirc l$ and $G_i \Rightarrow \bigcirc G_i$.

Resolution can be applied among these clauses and clause 2 producing
$s_i^{\neg l} \wedge G_i \Rightarrow \bigcirc \mathbf{false}$. By applying simplification this latter clause is transformed into $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee \neg G_i)$. Then, $G_{i+1} \Leftrightarrow G_i \wedge G_i$ and so $G_{i+1} \Leftrightarrow G_i$. □

Theorem 3. *The Step Loop Search Algorithm is complete.*

Proof. By Theorem 1 and 2.

□

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 $G_n \Leftrightarrow G_{n+1}$. Thus, the algorithm terminates by 2 *(a)*.
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 $s_i^{\neg l} \wedge C_i \Rightarrow \bigcirc \mathbf{false}$ and therefore no clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee F_i)$ will be derived during the proof and G_{i+1} will be empty.

□

6 The Search for a Subsequent Loop

We assume that we have detected a sequence of guesses by applying the previous algorithm to a set of SNF-clauses, X, with an eventuality $\diamondsuit\neg l$ but, later we need to apply temporal resolution again to $\diamondsuit\neg l$. Further, the application of the temporal resolution rule to this or other eventualities has led to the generation of new clauses which we may use in order to generate a new loop. Let Y be the set of new SNF-clauses generated. Thus the set of SNF-clauses is now updated as $X \cup Y$.

Rather than carrying out the full loop search again, our intention is to re-use the original loop search, even though a loop may not have been detected in the first search.

6.1 Repeated Step Loop Search Algorithm

Assume we have $G_0, ..., G_n$ guesses from a previous search for a loop and, the added set of clauses is Y. We can generate a new sequence of DNF formula, K_i as follows:

1. Guess $K_{-1} \Leftrightarrow \mathbf{false}$.
2. Given a guess K_i add the clause $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$ to the piece of proof of the previous search corresponding with s_i, together with the clauses Y.
3. Apply Step Resolution.
4. For all new clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee F_j')$ obtained during the proof, let
$$K_{i+1} \Leftrightarrow (G_i \vee K_i) \wedge (\bigvee_{j=1}^{m} \neg F_j')$$
5. Go to 2 until either
   (a) $K_i \Rightarrow K_{i+1} \vee G_{i+1}$ where $K_i \not\Rightarrow \mathbf{false}$ or
   (b) $K_i \vee G_i \Leftrightarrow K_{i+1} \vee G_{i+1}$,
   whatever is the earliest.

6.2 Example

We now consider the example in Section 4.3 where clause 4 has been deleted. In this case, all the guesses remain the same except the last one which now will be the following

16.	$s_2^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg a \vee \neg c \vee \neg d \vee \neg b)$	
17.	$s_2^{\neg l} \wedge a \wedge b \wedge c \wedge d \Rightarrow \bigcirc \neg b$	$[1, 2, 3, 5, 16]$

As there are not clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_2^{\neg l} \vee ....)$, then $G_3 \Leftrightarrow \mathbf{false}$.
Thus, the search for a loop failed as the set of clauses did not comprise a loop but we still can use these previous guesses in order to find a loop if the appropriate clause is later added.
Now we consider the same example but we add clause 4, that is, $d \wedge a \Rightarrow \bigcirc b$.
Algorithm 2 is now applied, with $K_{-1} \Leftrightarrow \mathbf{false}$. The new clause added is

18. $s_{-1}^{\neg l} \Rightarrow \bigcirc\mathbf{true}$

Nothing new is added and no clauses from 7-9 in example 4.3 can be resolved with 4. Thus, $K_0 \Leftrightarrow \mathbf{false}$ and $T_0 \Leftrightarrow a \vee \mathbf{false} \Leftrightarrow a$.

19. $s_0^{\neg l} \Rightarrow \bigcirc\mathbf{true}$

Again nothing new is added and no clauses from 10-12 in the example 4.3 can be resolved with 4. Thus, $K_1 \Leftrightarrow \mathbf{false}$ and $T_1 \Leftrightarrow (a \wedge b) \vee \mathbf{false} \Leftrightarrow (a \wedge b)$.

20. $s_1^{\neg l} \Rightarrow \bigcirc\mathbf{true}$

As happened in the previous cases no new clauses are added from resolution between 13-15 and 4, so $K_2 \Leftrightarrow \mathbf{false}$ and $T_2 \Leftrightarrow (a \wedge b \wedge c \wedge d) \vee \mathbf{false} \Leftrightarrow a \wedge b \wedge c \wedge d$.

20.	$s_2^{\neg l} \Rightarrow \bigcirc\mathbf{true}$	
21.	$s_2^{\neg l} \wedge a \wedge b \wedge c \wedge d \Rightarrow \bigcirc\mathbf{false}$	$[17, 4]$
22.	$\mathbf{true} \Rightarrow \bigcirc(\neg s_2^{\neg l} \vee \neg a \vee \neg b \vee \neg c \vee \neg d)$	[Simp.22]

$K_3 \Leftrightarrow (a \wedge b \wedge c \wedge d) \wedge (a \wedge b \wedge c \wedge d) \Leftrightarrow (a \wedge b \wedge c \wedge d)$.
$T_3 \Leftrightarrow (a \wedge b \wedge c \wedge d)$.
$T_2 \Leftrightarrow T_3$. Then T_3 is a loop.

6.3 The New Guess

Let $G_{-1}, G_0, ..., G_j$ be the guesses generated by applying Algorithm 1 to a set of clauses X. Let Y be the new set of clauses added to X and $T_{-1}, T_0, ..., T_i$ be the guesses obtained by applying Algorithm 1 to $X \cup Y$. $K_{-1}, K_0, ..., K_p$ are the guesses obtained by applying Algorithm 2.

Theorem 5. *The new guess T_i, such that $T_i \not\Leftrightarrow G_i$, $K_i \not\Leftrightarrow$* **false** *has the property $T_i \Leftrightarrow G_i \vee K_i$.*

Proof. We assume that the new guess is generated from the fragment of proof corresponding to $s_{i-1}^{\neg l}$ and K_i $\not{i}mp$**false** whereas $K_{-1}, K_0, ..., K_{i-1} \Leftrightarrow$ **false**.
Let **true** $\Rightarrow$ $\bigcirc(\neg s_{i-1}^{\neg l} \vee F_j)$ be the clauses used for Algorithm 1 in order to generate G_i.
As a result of adding the new clauses Y, if step resolution is applied among these clauses, X or those containing s_{i-1} on the left hand side, clauses
true $\Rightarrow \bigcirc(\neg s_{i-1}^{\neg l} \vee C_k)$ may be generated, where $F_j \not\Leftrightarrow C_k$. N.B. that C_k must exist as we are assuming $K_i \not\Leftrightarrow$ **false**.
By applying algorithm 1, the new guess will be

$$\begin{aligned}
T_i &\Leftrightarrow G_{i-1} \wedge [(\bigvee_{j=1}^{m} \neg F_j) \vee (\bigvee_{k=1}^{p} \neg C_k)] \\
&\Leftrightarrow (G_{i-1} \wedge (\bigvee_{j=1}^{m} \neg F_j)) \vee (G_{i-1} \wedge (\bigvee_{k=1}^{p} \neg C_k)) \\
&\Leftrightarrow G_i \vee K_i.
\end{aligned}$$

□

We have shown that the first new guess is like $T_i \Leftrightarrow G_i \vee K_i$. If $K_i \not\Leftrightarrow$ **false** then trivially $T_i \Leftrightarrow G_i \vee$ **false**. Next we will show that this is the case for all the new guesses.

Theorem 6. *Let i the first index such that $T_i \not\Leftrightarrow G_i$. Then for all j, $j \geq i$, $T_j \Leftrightarrow G_j \vee K_j$.*

Proof. We assume that $T_j \Leftrightarrow G_j \vee K_j$. We will prove that $T_{j+1} \Leftrightarrow G_{j+1} \vee K_{j+1}$. In order to obtain T_{j+1} the original algorithm must be applied to T_j. Thus, the clause added is $s_j^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg T_j)$. As we know that $T_j \Leftrightarrow G_j \vee K_j$ the previous clause can be rewritten as

$$s_j^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg G_j) \qquad (1)$$
$$s_j^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_j) \qquad (2)$$

As clause (1) was produced while obtaining G_{j+1}, the clauses
true $\Rightarrow \bigcirc(\neg s_j^{\neg l} \vee F_{j+1})$ that appeared in the original search will be generated again. From clause (2) and $X \cup Y$ new clauses **true** $\Rightarrow \bigcirc(\neg s_j^{\neg l} \vee C_{k+1})$ may be derived. Then by applying algorithm 1.

$$\begin{aligned}
T_{j+1} &\Leftrightarrow T_j \wedge [(\bigvee_{j=1}^{m} \neg F_{j+1}) \vee (\bigvee_{k=1}^{p} \neg C_{k+1})] \\
&\Leftrightarrow (G_j \vee K_j) \wedge [(\bigvee_{j=1}^{m} \neg F_{j+1}) \vee (\bigvee_{k=1}^{p} \neg C_{k+1})] \\
&\Leftrightarrow [G_j \wedge (\bigvee_{j=1}^{m} \neg F_{j+1})] \vee [K_j \wedge (\bigvee_{j=1}^{m} \neg F_{j+1})] \vee [(G_j \vee K_j) \wedge (\bigvee_{j=1}^{m} \neg C_{k+1})] \\
&\Leftrightarrow G_{j+1} \vee K_{j+1}
\end{aligned}$$

As $K_j \wedge (\bigvee_{j=1}^{m} \neg F_{j+1})$ is subsumed by $(G_j \vee K_j) \wedge (\bigvee_{j=1}^{m} \neg C_{k+1})$. □

6.4 Completeness of the Repeated Step Loop Search Algorithm

Theorem 7. *Algorithm 2 is complete.*

Proof. The proof follows Theorem 5 and Theorem 6 and completeness of the Step Loop Search Algorithm. □

Termination

Lemma 2. *For all i, $K_{i+1} \Rightarrow K_i \vee G_i$.*

Proof. Let $T_{-1}, T_0, ..., T_{i+1}$ be the guesses from applying Algorithm 1 to the clauses $X \cup Y$.
By Theorem 1 and definition of Breadth-First Search algorithm, $T_{i+1} \Rightarrow T_i$.
By Theorem 6 $T_i \Leftrightarrow G_i \vee K_i$. Then $K_{i+1} \vee G_{i+1} \Rightarrow K_i \vee G_i$. Therefore, $K_{i+1} \Rightarrow K_i \vee G_i$. □

Theorem 8. *The algorithm for searching for K_i terminates.*

Proof. We know that the search for T_i must terminate because of termination of the algorithm described in section 4.2 when $T_i \Leftrightarrow T_{i+1}$. By Lemma 2 we know that $K_{i+1} \Rightarrow K_i \vee G_i$ if $K_i \not\Leftrightarrow$ **false**. K_i has the property that $T_i \Leftrightarrow G_i \vee K_i$. Then $G_i \vee K_i \Leftrightarrow G_{i+1} \vee K_{i+1}$. Therefore $K_i \Rightarrow G_{i+1} \vee K_{i+1}$. □

7 Some Advantages of Algorithm 2

Let X be a set of SNF-clauses with an eventuality $\diamondsuit\neg l$ when we apply Algorithm 1 to X, we obtain a sequence of guesses $G_{-1}, G_0, ..., G_n$.
Now we assume that some new set of SNF clauses, Y, have been added and we intend to search for a loop in the set of clauses $X \cup Y$.
As we have shown in section 6.3, if a new set of clauses is added the new guess has the property $T_i \Leftrightarrow G_i \vee K_i$, where K_i is given by Algorithm 2. Applying the algorithm 1 to $X \cup Y$ for every new guess T_i, the clauses that we must add for generating the next guess are

$$s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg G_i)$$
$$s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$$

Thus, all step resolution amongst clauses $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$ and the set of clauses X will be produced again, as they were produced when Algorithm 1 was applied just to X.

If we apply Algorithm 2 instead, we can save all those resolution steps as we only add clauses of the form $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$.

8 Conclusions and Future Work

In this paper we have presented two algorithms for searching for loops based upon step resolution.

The first algorithm uses outputs from the previous guess to guide the choice for the next guess and its correctness is shown with respect to an existing loop search algorithm. The second algorithm is based on the first one and allows us search for a second loop without having to carry out a whole search again, since it uses clauses generated during previous searches.

In the future we intend to apply these results to the development of strategies for temporal resolution that allows us to reduce the search space. In particular, we are interested in incorporating the set of support strategy [17, 6]. In [7] the set of support is defined for the case without eventualities. For full temporal resolution the situation is more complex and we intend to achieve it combining and extending the results presented in [7] and the results in this paper.
Even though the algorithms presented are used in order to search for loops for the application of the temporal resolution operation, we can still guide this search. Thus, our intention is to define a set of support strategy for temporal resolution which involves a set of support for the search for loops process and then combines it with the set of support for step resolution. Thus, if we consider the example in Section 6.2 the Set of Support (SOS) for searching for a loop will include clauses of the form $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$.

The practical efficiency of the algorithms is part of current work. It is expected to be examined while updating an existing implementation for temporal resolution where the search for a loop process is substituted with the algorithms presented here.

We also intend to investigate the detailed complexity of this approach.

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 *Lecture Notes in Computer Science*, 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, December 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 International Workshop on Temporal Representation and Reasoning*, Sanibel Island, Florida, May 1998. IEEE Computer Society Press.

7. M.C. Fernández-Gago. Efficient Control of Temporal Reasoning. Transfer Report, Manchester Metropolitan University, 2000.
8. M. Fisher. A Resolution Method for Temporal Logic. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 99–104, Sydney, Australia, August 1991. Morgan Kaufman.
9. 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, August 1997.
10. M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. In *Transactions on Computational Logic 2(1)*, January 2001.
11. 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.
12. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.
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, M. Vardi, and P. Wolper. The Complementation Problem for Buchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49:217–237, 1987.
15. G. Venkatesh. A Decision Method for Temporal Logic based on Resolution. *Lecture Notes in Computer Science*, 206:272–289, 1986.
16. P. Wolper. The Tableau Method for Temporal Logic: An overview. *Logique et Analyse*, 110–111:119–136, June-Sept 1985.
17. 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.