Annals of Mathematics and Artificial Intelligence 22 (1998) 87–115 

# Temporal resolution using a breadth-first search algorithm *

Clare Dixon

*Department of Computing and Mathematics, Manchester Metropolitan University, Manchester M1 5GD, UK*
E-mail: C.Dixon@doc.mmu.ac.uk

An approach to applying clausal resolution, a proof method well suited to mechanisation, to temporal logics has been developed by Fisher. The method involves translation to a normal form, classical style resolution within states, and temporal resolution between states. Not only has it been shown to be correct but as it consists of only one temporal resolution rule, it is particularly suitable as the basis of an automated temporal resolution theorem prover. As the application of the temporal resolution rule is the most costly part of the method, it is on this area that we focus. Detailed algorithms for a breadth-first search approach to the application of this rule are presented. Correctness is shown and complexity given. Analysis of the behaviour of the algorithms is carried out and we explain why this approach is an improvement to others suggested.

## 1. Introduction

Resolution was proposed as a proof procedure by Robinson in 1965 for propositional and first-order logics [43]. It was claimed to be "machine-oriented" as it was particularly suitable for proofs to be performed by computer, having only one rule of inference that may have to be applied many times. The resolution principle has been used to construct theorem provers for classical logic, for example, OTTER [33], as well as being the basis of the logic programming language Prolog, see, for example, [27,28].

Temporal logics are an extension of classical logic, having specific operators that deal with time, for example, $\Box$ (*always*) and $\Diamond$ (*eventually* or *sometime*). Temporal logics have been used in a variety of fields for example, the analysis of reactive systems [31,32], hardware verification [34], knowledge representation [14] and execution [4, 18]. Temporal logics extended with additional modalities to deal with knowledge or belief have also been used to analyse distributed and multi-agent systems [19,24]. Much work has been carried out into the specification and verification of properties of concurrent systems using temporal logics, see, for example, [3,5,23,29,31,35,38,39,41]. Important computational properties such as liveness, deadlock and mutual exclusion

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

can be expressed easily and simply in temporal logic making it useful for specification. Verifying that such properties hold for a program specified in temporal logic involves proofs within the logic itself. To perform such proofs, tableaux [46] or automata [44] based approaches have been the most popular, both a form of *model checking*. However, standard model checking approaches are limited as only finite state problems can be handled and, even then the number of states required soon becomes large due to the combinatorial explosion. Alternatively, one can adopt a resolution based approach which is not limited to finite state problems and has a significant body of work on heuristics to control search (see standard texts such as [10] for example). Decision procedures based on resolution have been developed for temporal logics [2,9,45], 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.

In this paper we present a breadth-first search style algorithm which enables practical implementation of the resolution method for temporal logics developed by Fisher [15]. Fisher's method has been shown correct [36], deals with the full range of past and future-time temporal operators and has only one temporal resolution rule making it suitable for mechanisation. The resolution procedure is characterised by translation to a normal form, the application of a classical style resolution rule to derive contradictions that occur at the same point in time (termed *step resolution*), together with a new resolution rule, which derives contradictions over temporal sequences (termed *temporal resolution*). As it is the latter that is the most expensive part of the algorithm, involving search through graphs, as well as the most novel, it is on the application of the temporal resolution rule that we concentrate. We suggest a breadth-first search approach to the application of the temporal resolution rule and through analysis of its operation and output, explain why it is an improvement on search mechanisms suggested previously [12].

This paper is structured as follows. In section 2, the syntax and semantics of propositional temporal logic is given and an outline of Fisher's temporal resolution method is given in section 3. The breadth-first search algorithm developed to implement the temporal resolution step is described in section 4 with examples to illustrate its use. A correctness argument and complexity results are given in section 5. The output of the breadth-first search algorithm and a description of its implementation are discussed in section 6. Finally we draw conclusions and consider related and future work in section 7.

## 2. Propositional temporal logic

Propositional temporal logic (PTL) is based on a linear, discrete model of time with finite past and infinite future [21]. This logic may be viewed as a classical propositional logic augmented with both future-time and past-time temporal connectives.[1]

[1] 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 or weak until*), while past-time connectives include '$\blacklozenge$' (*sometime in the past*), '$\blacksquare$' (*always in the past*), '$\boxdot$' (*in the previous moment in time or strong last*), '$\mathcal{S}$' (*since*), and '$\mathcal{Z}$' (*zince or weak since*). We also use an operator '**start**' meaning *at the beginning of time*.

### 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 $\boxdot$, $\blacklozenge$, $\blacksquare$, $\mathcal{S}$, and $\mathcal{Z}$.

The set of well-formed formulae of PTL,[2] 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

$$\begin{array}{lllll} \neg A, & A \vee B, & A \wedge B, & A \Rightarrow B, & A \Leftrightarrow B, \\ \Diamond A, & \Box A, & A\,\mathcal{U}\,B, & A\,\mathcal{W}\,B, & \bigcirc A, \\ \blacklozenge A, & \blacksquare A, & A\,\mathcal{S}\,B, & A\,\mathcal{Z}\,B, & \boxdot A. \end{array}$$

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 $\sigma$ at state index $i \in \mathbf{N}$. For any formula $A$, model $\sigma$ and state index $i \in \mathbf{N}$, then either $(\sigma, i) \models A$ holds

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

or $(\sigma, i) \models A$ does not hold, denoted by $(\sigma, i) \not\models A$. If there is some $\sigma$ such that $(\sigma, 0) \models A$, then $A$ is said to be *satisfiable*. If $(\sigma, 0) \models A$ for all models, $\sigma$, then $A$ is said to be *valid* and is written $\models A$.[3]

The semantics of a proposition is defined by the valuation given to it in the model

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

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:

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

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

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

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

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

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

$$\begin{array}{lll} (\sigma, i) \models \bigcirc\!\!\!\!\bullet\, A & \text{iff} & i > 0 \text{ and } (\sigma, i-1) \models A; \\ (\sigma, i) \models \blacklozenge A & \text{iff} & \text{there exists } h \text{ such that } 0 \leqslant h < i \text{ and } (\sigma, h) \models A; \\ (\sigma, i) \models \blacksquare A & \text{iff} & \text{for all } h \text{ such that } 0 \leqslant h < i,\ (\sigma, h) \models A. \end{array}$$

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

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

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:

$$\textbf{start} \equiv \neg\, \bigcirc\!\!\!\!\bullet\, \textbf{true}.$$

[3] Note that formulae here are interpreted at $s_0$. This is an alternative definition to the one usually used in temporal logics, but is equally expressive.

### 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 letters $I, R, S, T, U$ to represent sets of formulae.
- We use the uppercase letters $G, N, E$ to represent graphs, nodes and edges, respectively.

## 3. The temporal resolution method

The temporal resolution method may be divided into three main procedures:

- translation to a normal form;
- step resolution; and
- temporal resolution.

As Fisher's method is clausal, to ascertain the validity of a PTL formula $A$, we negate and translate $\neg A$ into a normal form, to which resolution, both step and temporal, may be applied. Step resolution consists of the application of a standard classical style resolution rule to formulae that represent constraints holding at a particular moment in time. Temporal resolution is applied between a $\diamondsuit$-formula, for example $\diamondsuit l$, and a derived formula meaning that $\neg l$ must always hold, i.e., $\Box\neg l$. The procedure terminates when either **false** has been derived, showing that $\neg A$ is unsatisfiable and therefore $A$ is valid or when no further resolution rules can be applied, showing $\neg A$ is satisfiable.

### 3.1. SNF

Formulae in PTL can be transformed to a normal form, called *separated normal form* (SNF), which is the basis of the resolution method described here. SNF was introduced first in [15] and has been extended both in [16,17]. The transformation of rules into SNF is beyond the scope of this paper although the interested reader may refer to [15–17] for more details. We note that the transformation into SNF preserves satisfiability and so any contradiction generated from the formula in SNF implies a contradiction in the original formula.

Formulae in SNF are of the general form

$$\Box \bigwedge_i A_i$$

where each $A_i$ is known as a *rule* and must be one of the following forms where each particular $k_a$, $k_b$, $l_c$, $l_d$ and $l$ represent literals:

$$\begin{array}{lll} \textbf{start} \Rightarrow \bigvee_c l_c & & \text{(an } \textit{initial } \Box\text{-rule)}, \\ \bigcirc\bigwedge_a k_a \Rightarrow \bigvee_d l_d & & \text{(a } \textit{global } \Box\text{-rule)}, \\ \textbf{start} \Rightarrow \Diamond l & & \text{(an } \textit{initial } \Diamond\text{-rule)}, \\ \bigcirc\bigwedge_b k_b \Rightarrow \Diamond l & & \text{(a } \textit{global } \Diamond\text{-rule)}. \end{array}$$

The outer '$\Box$' connective, 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 also define a variant on SNF called merged-SNF (SNF$_m$) [15], for combining rules. Given a set of rules in SNF, the relevant set of SNF$_m$ rules may be generated by repeatedly applying the following rule.

$$\frac{\bigcirc A \Rightarrow C \qquad \bigcirc B \Rightarrow D}{\bigcirc(A \wedge B) \Rightarrow C \wedge D}$$

The right hand side of this SNF$_m$ rule generated may have to be further translated into *disjunctive normal form* (DNF), if either $C$ or $D$ are disjunctive, to maintain the basic rule structure. Thus, SNF$_m$ represents all possible conjunctive combinations of SNF rules. We use rules in SNF$_m$ to apply the temporal resolution rule, see section 3.3.

### *3.2. Step resolution*

Once a formula has been transformed into SNF, both step resolution and temporal resolution can be applied. Step resolution effectively consists of the application of the standard classical resolution rule to formulae representing constraints at a particular moment in time, together with simplification rules, subsumption rules, and rules for transferring contradictions within states to constraints on previous states. The step resolution rule is a form of classical resolution applied between $\Box$-rules, representing constraints applying to the same moment in time. Pairs of initial $\Box$-rules, or global $\Box$-rules, may be resolved using the following (step resolution) rules.

$$\frac{\textbf{start} \Rightarrow A \vee p \qquad \textbf{start} \Rightarrow B \vee \neg p}{\textbf{start} \Rightarrow A \vee B}, \qquad \frac{\bigcirc C \Rightarrow A \vee p \qquad \bigcirc D \Rightarrow B \vee \neg p}{\bigcirc(C \wedge D) \Rightarrow A \vee B}$$

The following SNF rules can be removed during simplification as they represent valid subformulae and therefore cannot contribute to the generation of a contradiction:

$$\bigcirc\textbf{false} \Rightarrow A, \qquad A \Rightarrow \textbf{true}.$$

The first rule is valid as $\bigcirc$**false** can never be satisfied, and the second is valid as **true** is always satisfied. The following rewrite rule is used for rules which imply **false**:

$$\{\bigcirc A \Rightarrow \textbf{false}\} \longrightarrow \left\{ \begin{array}{c} \textbf{start} \Rightarrow \neg A \\ \bigcirc\textbf{true} \Rightarrow \neg A \end{array} \right\}$$

and states that if, by satisfying $A$ in the previous moment in time a contradiction is produced, then $A$ must never be satisfied. The new constraints therefore represent $\Box\neg A$.

Subsumption also forms a part of the step resolution process. Here, as in classical resolution, a rule may be removed from the rule-set if it is subsumed by another rule. Subsumption may be expressed as the following rewrite rule:

$$\left\{ \begin{array}{c} C \Rightarrow A \\ D \Rightarrow B \end{array} \right\} \xrightarrow{\vdash C \Rightarrow D \quad \vdash B \Rightarrow A} \{D \Rightarrow B\},$$

where the symbol '$\vdash$' means 'is provable' or 'is true'. The side conditions $\vdash C \Rightarrow D$ and $\vdash B \Rightarrow A$ must hold before this subsumption step can be applied and the rule $C \Rightarrow A$ can be deleted without losing information.

The step resolution process terminates when either no new resolvents are generated or **false** is derived by generating one of the following unsatisfiable formulae:

$$\textbf{start} \Rightarrow \textbf{false}, \qquad \bigcirc\textbf{true} \Rightarrow \textbf{false}.$$

### *3.3. Temporal resolution*

The temporal resolution process consists of (possibly multiple) applications of the temporal resolution rule resolving together formulae with the '$\Box$' and '$\Diamond$' connectives. However, the interaction between the '$\bigcirc$' and '$\Box$' connectives in PTL ensures that the application of such a rule is nontrivial. Further, as the translation to SNF restricts the rules to be of a certain form, the application of such a rule will be between a $\Diamond$-rule and the output of an algorithm that ensures an invariance, i.e., a literal will *always* hold, which will contradict the $\Diamond$-rule. Thus, given a set of rules in SNF, then for every eventuality, i.e., a formula $\Diamond l$ from the right hand side of a rule of the form

$$\textbf{start} \Rightarrow \Diamond l \quad \text{or} \quad \bigcirc C \Rightarrow \Diamond l,$$

temporal resolution may be applied between this $\Diamond$-rule and a formula which forces $\neg l$ always to be satisfied. Once the left hand side of the $\Diamond$-rule is satisfied (i.e., either **start** or $\bigcirc C$) then the eventuality $l$ can only be fulfilled if none of the conditions that force $\neg l$ always to hold are satisfied until $l$ is satisfied. To resolve with the above rule then, a DNF formula $\bigvee_{i=0}^{n} A_i$ must be identified such that

$$\bigcirc \bigvee_{i=0}^{n} A_i \Rightarrow \Box\neg l,$$

where each $A_i$ is a conjunction of literals. So, the general temporal resolution rule, written as an inference rule, becomes

$$\frac{\bigcirc\bigvee_{i=0}^{n} A_i \Rightarrow \Box\neg l \qquad D \Rightarrow \Diamond l}{D \Rightarrow \bigwedge_{i=0}^{n} (\neg A_i)\, \mathcal{W}\, l}$$

where the resolvent states that once $D$ (i.e., **start** or $\bigcirc C$) has occurred then $l$ must occur (i.e., the eventuality must be satisfied) before $\bigvee_{i=0}^{n} A_i$ can be satisfied.[4] The '$\mathcal{W}$' connective is used as we already have a rule guaranteeing that $\Diamond l$ *will* occur. The resolvent must be translated into SNF. As most of the formulae we consider will be in SNF, we translate this resolvent into SNF below for $i = 0, \ldots, n$ and $t$ a fresh proposition symbol:

$$D \Rightarrow \neg A_i \vee l, \qquad \bigcirc t \Rightarrow \neg A_i \vee l,$$
$$D \Rightarrow t \vee l, \qquad \bigcirc t \Rightarrow t \vee l.$$

We note that the first two resolvents depend on the particular $\Box$-formula detected but the last two do not.

For more details of the resolution method see [15].

### 3.4. Loops and loop-formulae

As for most of the remainder of the paper we are concerned with the notion of *loops* and *loop-formulae* we examine these in detail.

**Definition 1** (a loop-formula in $\neg l$). Let $R$ be a set of global $\Box$-rules and $\Diamond l$ an eventuality occurring on the right hand side of a $\Diamond$-rule. A *loop-formula in* $\neg l$ is a DNF formula

$$\bigvee_{i=0}^{n} A_i \quad \text{such that} \quad \bigcirc\bigvee_{i=0}^{n} A_i \Rightarrow \Box\neg l.$$

**Definition 2** (a loop in $\neg l$). Let $R$ be a set of global $\Box$-rules and $\Diamond l$ an eventuality occurring on the right hand side of a $\Diamond$-rule. A *loop in* $\neg l$ is a set of $\text{SNF}_m$ rules $\bigcirc A_i \Rightarrow B_i$ for $i = 0, \ldots, n$ such that

(1) $0 \leqslant i \leqslant n \vdash B_i \Rightarrow \neg l$,

(2) $\vdash B_i \Rightarrow \bigvee_{j=0}^{n} A_j$.

[4] In previous presentations two resolvents have been given. As the resolvent given here is sufficient for completeness, we omit the other for simplicity.

This definition ensures that each $\Box$-rule makes $\neg l$ true and the right hand side of each $\Box$-rule ensures that the left hand side of one of the $\Box$-rules will be satisfied.[5]

**Lemma 3.** Given a loop in $l$, $\bigcirc A_i \Rightarrow B_i$ for $i = 0, \ldots, n$, then $\bigvee_{i=0}^{n} A_i$ is a loop-formula for $l$.

*Proof.* Obvious from the definition of a loop in $l$. □

In previous presentations the temporal resolution rule is given as a $\Diamond$-rule resolved with a loop rather than a loop formula as a loop is more easily related to a set of rules. However, lemma 3 shows we can generate a loop-formula from any loop and apply the revised resolution rule.

Note, that as we have translated to a normal form we either have to perform an algorithm to detect a loop-formula directly or can first detect a loop and then obtain a loop-formula from a loop by disjoining the conjunctions of literals from the left hand sides of rules.

**Lemma 4.** Given two loop-formulae, $D_1$ and $D_2$ for $l$, then $D_1 \vee D_2$ is also a loop-formula for $l$.

*Proof.* Obvious. □

**Lemma 5.** Given two loops, $L_1$ and $L_2$ in $l$, then $L_1 \cup L_2$ is also a loop in $l$.

*Proof.* Obvious. □

**Lemma 6.** Given a loop in $l$, $L$, such that $\bigcirc A_0 \Rightarrow B_0 \in L$, $\bigcirc A_1 \Rightarrow B_1 \in L$ and $A_1 \Rightarrow A_0$ then $L - \{\bigcirc A_1 \Rightarrow B_1\}$ is an equivalent (but simpler) loop in $l$.

*Proof.* Let $L$ be the loop in $l$, $\bigcirc A_i \Rightarrow B_i$ for $i = 0, \ldots, n$ and $n \geqslant 1$. For $L - \{\bigcirc A_1 \Rightarrow B_1\}$ to be a loop the two conditions in definition 2 must hold. The first condition, each right hand side must imply $\neg l$, must still hold as we assumed that $L$ itself was a loop. The second condition must also hold as if $A_1 \Rightarrow A_0$ then

$$\bigvee_{i=0}^{n} A_i \Leftrightarrow A_0 \vee \bigvee_{i=2}^{n} A_i.$$ □

In summary, the union of two loops in $l$ is also a loop in $l$ (lemma 5) and therefore (modulo equivalence) there exists a unique maximal loop in $l$. From a loop in $l$ we can construct a loop-formula in $l$ (lemma 3). Similarly the disjunction of two loop-formulae in $l$ is also a loop-formulae in $l$ (lemma 4) and therefore there exists

[5] In its original presentation [15] there was a third side condition but this was found to make the method incomplete [36].

a unique maximal loop-formula (modulo equivalence) in $l$. It is this that we aim to detect, using the algorithm given in section 4, to apply the temporal resolution rule.

### *3.5. An algorithm for the temporal resolution method*

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

1. Translate $A$ into SNF $A_s$.
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 loop-formulae for $\neg l$ and generate the resolvents.
4. If any new formulae have been generated, translate the resolvents into SNF and go to step 2.
5. Terminate declaring $A$ satisfiable.

## 4. Breadth-first search

Recall that to apply the temporal resolution rule given in section 3.3 a loop-formula must be detected for resolution with one of the $\Diamond$-rules. Specifically, given that we want to resolve with the eventuality $\Diamond l$ our aim is to detect a loop-formula

$$\bigvee_{i=0}^{m} A_i \quad \text{such that} \quad \bigcirc \bigvee_{i=0}^{m} A_i \Rightarrow \Box\neg l.$$

As this is the area that the most efficiency gains are to be made, we devote the following section to a detailed consideration of a breadth-first search approach to detecting a loop-formula. Having given a general description of the algorithm in section 4.1 the high level algorithm for creating the graph and its termination conditions is given. The process of stepping through the rule-set rule by rule and attempting to use or discard each rule to generate the next node is given in section 4.2 while section 4.3 explains how the rules may be combined efficiently. The simplification performed on nodes is described in section 4.4 and some examples illustrating the operation of the breadth-first search algorithm are given in section 4.5.

### *4.1. Overview*

The breadth-first search algorithm constructs a sequence of nodes, $H_i$ for $i \geqslant 0$, 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 rules or combinations of rules in the rule-set that satisfy certain properties. Assuming we are resolving with $\Diamond l$ each node $H_i$ satisfies $\bigcirc H_i \Rightarrow \neg l$ and given $H_i$ each new node $H_{i+1}$ satisfies $\bigcirc H_{i+1} \Rightarrow H_i$. When termination occurs we have $H_{i+1} \Leftrightarrow H_i$ so that $\bigcirc H_i \Rightarrow \Box\neg l$ for resolution with $\Diamond l$. We assume that all step resolution has been carried out.

**Breadth-first search algorithm**. For each eventuality $\Diamond l$ occurring on the right hand side of a $\Diamond$-rule do the following:

1. Search for all the rules of the form $\bigcirc C_k \Rightarrow \neg l$, for $k = 0$ to $b$ (called *start rules*), disjoin the left hand sides and make the *top node* $H_0$ equivalent to this, i.e.,

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

   Simplify $H_0$ (see section 4.4). If $\vdash H_0 \Leftrightarrow$ **true** we terminate having found a loop-formula (**true**).

2. Given node $H_i$, build node $H_{i+1}$ for $i = 0, 1, \ldots$ by looking for rules or combinations of rules of the form $\bigcirc A_j \Rightarrow B_j$, for $j = 0$ to $c$ where $\vdash B_j \Rightarrow H_i$ and $\vdash A_j \Rightarrow H_0$. Disjoin the left hand sides so that

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

   and simplify as previously.

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

Termination means that $H_{j+1} \Leftrightarrow H_j$ for some node $H_j$. As we have constructed each node $H_{i+1}$ we have checked that $\bigcirc H_{i+1} \Rightarrow H_i$ so that when we terminate $\bigcirc H_{j+1} \Rightarrow H_j$. Now as $H_j \Rightarrow H_{j+1}$ we have $\bigcirc H_j \Rightarrow H_j$. Nodes have been constructed from SNF rules and that there is an implicit $\Box$-formula surrounding SNF formulae, so $\Box(\bigcirc H_j \Rightarrow H_j)$. Each node, $H_i$, has been constructed so that $\bigcirc H_i \Rightarrow \neg l$ so on termination $\bigcirc H_j \Rightarrow \neg l$ and as $\Box(\bigcirc H_j \Rightarrow H_j)$ we have $\bigcirc H_j \Rightarrow \Box\neg l$, i.e., $H_j$ is a loop-formula for resolution with a $\Diamond$-rule with $\Diamond l$ on the right hand side.

### *4.2. Node creation*

As part of step 2 in section 4.1 we provide an algorithm to test whether to use an SNF rule directly, whether it needs to be combined into $\text{SNF}_m$ or can be discarded, so

as to satisfy the conditions in step 2 in section 4.1. Assume we are again resolving with a rule that has $\Diamond l$ on the right hand side and that node $H_i$ has just been constructed. Let steps (i), (ii), etc., mentioned below refer to the following node creation algorithm. If an SNF rule can be used without combining into SNF$_m$ that will ensure $\neg l$ holds and that the previous node, $H_i$, is satisfied then the left hand side of this rule is added directly as a disjunct of the new node (step (i) of the algorithm). Although we want to try and use as many of the rules as we can, there is no need to combine them unnecessarily as these combinations will be removed during the simplification of the node (see section 4.4). In step (ii) the rule does not ensure $\neg l$ holds. Thus the rule must be combined with others into SNF$_m$. The way to do this which makes the smallest number of combinations is by combining the rule with each start rule. In step (iii) any rules that may potentially be used to expand the graph but must be combined with other rules (that is other than start rules) into SNF$_m$ are added to a set for later combination. Step (iv) discards any rules that do not satisfy conditions (i)–(iii) and are not required for the creation of the new node. First the definition of a function, lits($C$), required in the algorithm is given.

The function lits($C$), extracts the set of literals from (simplified) DNF formulae $C$ with negations pushed through to propositions, as follows:

$$\begin{aligned}
\text{lits}(\textbf{true}) &= \{\,\},\\
\text{lits}(\textbf{false}) &= \{\,\},\\
\text{lits}(p) &= \{p\} \quad \text{if } p \in \mathcal{P},\\
\text{lits}(\neg p) &= \{\neg p\} \quad \text{if } p \in \mathcal{P},\\
\text{lits}(A \wedge B) &= \text{lits}(A) \cup \text{lits}(B),\\
\text{lits}(A \vee B) &= \text{lits}(A) \cup \text{lits}(B),
\end{aligned}$$

where $\mathcal{P}$ is the set of proposition symbols.

**Node creation algorithm**. Let set $R = \emptyset$. In order to create the new node, $H_{i+1}$, (step 2 of the algorithm in section 4.1) each rule, excluding the start rules, in the set of SNF global $\Box$-rules, $\bigcirc C \Rightarrow D$, is examined and

(i) if $\vdash D \Rightarrow H_i$ and $\vdash C \Rightarrow H_0$ then $C$ is added as a disjunct of $H_{i+1}$;

(ii) if $\vdash D \Rightarrow H_i$ but $\nvdash C \Rightarrow H_0$ then $\bigcirc C \Rightarrow D$ is combined with every start rule (into SNF$_m$) in turn and the literals on the left hand sides are added as new disjuncts to $H_{i+1}$;

(iii) if neither of the above hold, but lits($D$) $\subseteq$ lits($H_i$) then $\bigcirc C \Rightarrow D$ is added to set $R$ for combination into SNF$_m$ (see section 4.3);

(iv) if none of the above hold the rule is discarded for the purposes of this node.

### *4.3. Combining rules*

Here the remaining rules are combined into SNF$_m$. Rather than combine every rule with every other, rules can be combined in an efficient way by taking each disjunct

in node $H_i$ in turn and trying to extract rules from those left to combine that will imply the selected disjunct. Then the combined rule is checked to see whether its right hand side does actually imply $H_i$. If so the processing continues as above. Otherwise the combined rule is added to the set of rules available for combinations and may be combined further.

### 4.4. Simplification of nodes

Here we note the simplification rules used to manipulate the nodes in DNF that the breadth-first algorithm builds. Assume that the node is in DNF and that each disjunct is in its simplest form. This will have been carried out during the merging of rules into SNF$_m$. Let $C$ and $D$ be formulae in DNF (or **false**), $A$ and $B$ be conjunctions of literals (or **true**) and $l$ a literal. Note that ordering of disjuncts makes no difference, i.e., we can apply the rules $C \vee D \longrightarrow D \vee C$ without affecting the truth value of the formula. When a node is constructed apply the following simplification rules:

1. Subsumption: $C \vee A \vee B \longrightarrow C \vee A$ iff $\vdash B \Rightarrow A$.
2. Complementary literals in different disjuncts:

$$C \vee (A \wedge l) \vee (B \wedge \neg l) \longrightarrow C \vee (A \wedge l) \vee (B \wedge \neg l) \vee (A \wedge B)$$

where the conjunction $(A \wedge B)$ may need further simplification.

### 4.5. Examples

We give two examples to illustrate the use of the breadth-first search algorithm.

**Example 1.** Breadth-first search is used to detect the loop-formula in the set of rules given below. Assume we are trying to resolve with the eventuality $\Diamond l$ and the set of global $\Box$-rules is:

| | | | |
|---|---|---|---|
| 1. | $\bigcirc a \Rightarrow \neg l$ | 5. | $\bigcirc e \Rightarrow e$ |
| 2. | $\bigcirc b \Rightarrow \neg l$ | 6. | $\bigcirc(a \wedge e) \Rightarrow a$ |
| 3. | $\bigcirc c \Rightarrow \neg l$ | 7. | $\bigcirc b \Rightarrow b$ |
| 4. | $\bigcirc d \Rightarrow \neg l$ | 8. | $\bigcirc c \Rightarrow b$. |

The rules 1–4 have $\neg l$ on their right hand side. We disjoin their left hand sides and simplify (although in this case no simplification is necessary) to give the top node

$$H_0 = a \vee b \vee c \vee d.$$

To build the next node, $H_1$, we see that rules 6–8 satisfy the expansion criteria in step 2 of the breadth-first search algorithm (i.e., the right hand side and the literals on the left hand side of each rule implies $H_0$) but rule 5 does not. Note if we combine rule 5 with any of the other rules to produce an SNF$_m$ rule that satisfies the expansion

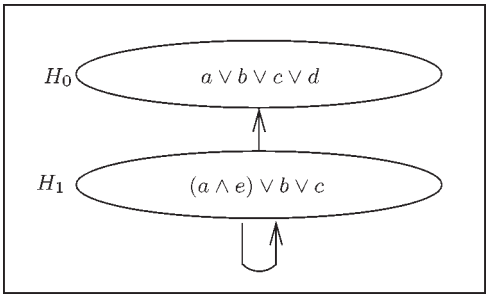


Figure 1. Breadth-first search for example 1.

criteria, its left hand side will be removed through simplification. So we disjoin the literals on the left hand sides of rules 6–8 to obtain node

$$H_1 = (a \wedge e) \vee b \vee c.$$

Rules 7 and 8 satisfy the expansion criteria and so do rules 5 and 6 when combined together to give the rule $\bigcirc(a \wedge e) \Rightarrow a \wedge e$. Thus node $H_2$ becomes

$$H_2 = (a \wedge e) \vee b \vee c.$$

As $H_2 \Leftrightarrow H_1$ we terminate having detected a loop-formula. The loop-formula output by breadth-first search is $(a \wedge e) \vee b \vee c$ and therefore $\bigcirc(a \wedge e) \vee b \vee c \Rightarrow \Box\neg l$. The graph constructed using breadth-first search for this set of rules is shown in figure 1.

**Example 2.** Assume we are trying to resolve the eventuality $\Diamond l$ with the following rules:

1. $\bigcirc x \Rightarrow \neg l$
2. $\bigcirc a \Rightarrow \neg l$
3. $\bigcirc b \Rightarrow c \vee d$
4. $\bigcirc c \Rightarrow a$
5. $\bigcirc d \Rightarrow a$
6. $\bigcirc a \Rightarrow x$
7. $\bigcirc x \Rightarrow b.$

The rules 1 and 2 both have $\neg l$ on their right hand side. We disjoin their left hand sides to give the top node

$$H_0 = x \vee a$$

and note that lits($H_0$) $= \{x, a\}$. To build the next node, $H_1$, we examine each rule (other than rules 1 and 2) in turn using the algorithm given in section 4.2.

- As $c \vee d \not\Rightarrow H_0$ and lits($c \vee d$) $\not\subseteq$ lits($H_0$) rule 3 is discarded.
- As $a \Rightarrow H_0$ but $c \not\Rightarrow H_0$ we combine rule 4 with rules 1 and 2 to give

  $1 + 4.$ $\bigcirc(x \wedge c) \Rightarrow a \wedge \neg l$
  $2 + 4.$ $\bigcirc(a \wedge c) \Rightarrow a \wedge \neg l.$

  Now $(a \wedge \neg l) \Rightarrow H_0$ and both $(x \wedge c) \Rightarrow H_0$ and $(a \wedge c) \Rightarrow H_0$ so the left hand sides are disjoined and added to the new node to give $H_1 = (x \wedge c) \vee (a \wedge c)$.

- Similarly, we must combine rule 5 with rules 1 and 2 and add the left hand sides to $H_1$ giving $H_1 = (x \wedge c) \vee (a \wedge c) \vee (x \wedge d) \vee (a \wedge d)$.
- Rule 6 can be used directly so $a$ is disjoined to $H_1$ and simplified giving $H_1 = (x \wedge c) \vee (x \wedge d) \vee a$.
- Rule 7 is similar to rule 3 as $b \not\Rightarrow H_0$ and lits($b$) $\not\subseteq$ lits($H_0$), so rule 7 is discarded.

We have

$$H_1 = (x \wedge c) \vee (x \wedge d) \vee a$$

and note that lits($H_1$) $= \{a, c, d, x\}$ We now start trying to build node $H_2$. We discard rule 7 as $b \not\Rightarrow H_1$ and lits($b$) $\not\subseteq$ lits($H_1$). Rules 4 and 5 satisfy the expansion criteria after combining each rule with one of the start rules so we obtain

$$H_2 = (x \wedge c) \vee (a \wedge c) \vee (x \wedge d) \vee (a \wedge d).$$

Rules 3 and 6 satisfy step (iii) of the node creation algorithm so we combine them obtaining the rule

$$3 + 6. \quad \bigcirc (a \wedge b) \Rightarrow (c \wedge x) \vee (d \wedge x)$$

which now satisfies the expansion criteria. We add $(a \wedge b)$ as a disjunct to $H_2$ and simplify to give

$$H_2 = (x \wedge c) \vee (a \wedge c) \vee (x \wedge d) \vee (a \wedge d) \vee (a \wedge b)$$

and note that lits($H_2$) $= \{a, b, c, d, x\}$. We start trying to build the next node, $H_3$. For each of rules 3–7 of the form $\bigcirc C \Rightarrow D$, $D \not\Rightarrow H_2$ and lits($D$) $\subseteq$ lits($H_2$), so all are used for combinations. Combining rules 3 and 6, 4 and 7, and 5 and 7 gives

$$\begin{array}{ll} 3 + 6. & \bigcirc(a \wedge b) \Rightarrow (c \wedge x) \vee (d \wedge x) \\ 4 + 7. & \bigcirc(c \wedge x) \Rightarrow (a \wedge b) \\ 5 + 7. & \bigcirc(d \wedge x) \Rightarrow (a \wedge b) \end{array}$$

and we can use the left hand sides of these rules as disjuncts in $H_3$ to give

$$H_3 = (a \wedge b) \vee (c \wedge x) \vee (d \wedge x).$$

Similarly we can combine rule 3 with either 4 or 5 giving

$$\begin{array}{ll} 3 + 4. & \bigcirc(b \wedge c) \Rightarrow (a \wedge c) \vee (a \wedge d) \\ 3 + 5. & \bigcirc(b \wedge d) \Rightarrow (a \wedge c) \vee (a \wedge d). \end{array}$$

For these two rules we must further combine with rules 1 and 2 and add the combined left hand sides to the new node and simplify to give

$$H_3 = (a \wedge b) \vee (c \wedge x) \vee (d \wedge x).$$

We note lits($H_3$) $= \{a, b, c, d, x\}$. We now try to build node $H_4$. Again we must use all rules for combinations. As above, combining the rules 3 and 6, 4 and 7, and 5 and

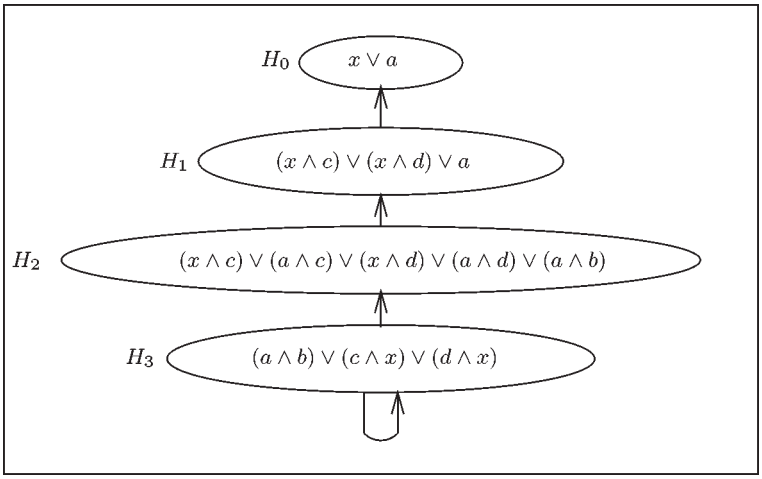


Figure 2. A breadth-first search on example 2.

7 give new rules that can be used to expand node from $H_3$. Disjoining the left hand sides of these rules and simplifying we obtain

$$H_4 = (c \wedge x) \vee (d \wedge x) \vee (a \wedge b).$$

As $H_4 \Leftrightarrow H_3$ we terminate having detected a loop-formula. The loop-formula output from breadth-first search is $(c \wedge x) \vee (d \wedge x) \vee (a \wedge b)$ such that

$$\bigcirc\big((c \wedge x) \vee (d \wedge x) \vee (a \wedge b)\big) \Rightarrow \Box\neg l.$$

The graph constructed using breadth-first search for this set of rules is shown in figure 2.

## 5. Correctness and complexity

### 5.1. *Correctness*

To show the correctness of the breadth-first search algorithm we construct a graph of the set of global $\Box$-rules and relate sets of nodes in the graph to the DNF formula constructed at each step in the breadth-first search algorithm. We omit the proofs of many of the subsidiary results and refer the interested reader to [11] for more detail. We begin with some definitions relating to loops. Recall a loop, $L$, is a set of $\text{SNF}_m$ rules, combined from the set of global $\Box$-rules satisfying the side conditions of section 3.4. As we claim that breadth-first search detects the maximal loop-formula the following definitions try and capture the notion of a maximal loop in the set of global $\Box$-rules.

**Definition 7** (a proper loop in $l$). Let $T$ be a set of SNF rules, $R$ the set of global $\Box$-rules of $T$ and $L$ a loop in $l$ made by combining rules in $R$. For each rule $r \in L$ of the form $\bigcirc A \Rightarrow B$ find the set

$$L_r = \{\bigcirc C \Rightarrow D \mid \bigcirc C \Rightarrow D \in R \text{ and } \vdash A \Rightarrow C\}.$$

For each $r$ in $L$ combine all the rules in $L_r$ to give the set of SNF$_m$ rules $L'$. If $L'$ contains no rules with **false** on the right hand side then $L'$ is a *proper loop in* $l$.

**Definition 8** (maximal proper loop in $l$). Let $T$ be a set of SNF rules and $R$ the set of global $\Box$-rules of $T$. $L$, a proper loop in $l$, formed from combining rules in $R$ is said to be a *maximal proper loop in* $l$ if and only if for all proper loops $L'$ in $l$, from combining rules in $R$, $L' \subseteq L$.

Note there is not always a maximal proper loop. For example the set of rules $\{\bigcirc a \Rightarrow a, \bigcirc a \Rightarrow l, \bigcirc a \Rightarrow \neg a\}$ contains the loop in $l$ $\bigcirc a \Rightarrow (a \wedge l)$ but no proper loop (as we obtain $\bigcirc a \Rightarrow$ **false**). However if a maximal proper loop does exist it will be unique.

Next we show how to construct a graph of the set of global $\Box$-rules known as the *reduced behaviour graph* and identify the parts of the graph that represent a loop. Paths through the graph represent models of the set of global $\Box$-rules. The approach is based on a suggestion in [15] where a directed graph is built to represent the global $\Box$-rules by combining them into SNF$_m$. We build the reduced behaviour graph for the set of global $\Box$-rules following the construction presented in [36].

**Building the reduced behaviour graph**. Given a set $T$ of SNF rules let

- $I \subseteq T$ be the set of initial $\Box$-rules,
- $R \subseteq T$ be the set of global $\Box$-rules, and
- $S \subseteq T$ be the set of $\Diamond$-rules.

Given $T$, we construct a finite directed graph $G = (N, E)$, for $R$ the set of global $\Box$-rules, where $N$ is the set of nodes and $E$ is the set of edges, as follows. A node, $n$, in $G$ is a valuation of all the propositions occurring in $T$. Let $R_n \subseteq R$ be the set of rules such that $n$ satisfies the conjunction of literals on the left hand side of each rule. Let $U$ be the set of clauses on the right hand side of $R_n$. For each valuation $n'$ that satisfies $U$ there is an edge in $G$ from $n$ to $n'$, and these are the only edges originating from $n$. Note, for node $n$, if there are no (explicit) rules whose left hand sides are satisfied by $n$, then there is an edge from $n$ to every other node, including itself, representing the use of the valid rule $\bigcirc$**true** $\Rightarrow$ **true** to construct edges. However, if the rules in $R_n$ are combined and produce an SNF rule of the form $\bigcirc A \Rightarrow$ **false** then no edges lead out of the node $n$. $G$ is known as the *behaviour graph* of $R$.

To obtain the *reduced behaviour graph*, given a behaviour graph for $R$, delete any node with no successors (and all edges into that node), until no more deletions are possible. We call the graph we obtain having performed all possible deletions the *reduced behaviour graph* for $R$.

**Definition 9** (terminal subgraph where $l$ holds). Let $T$ be a set of SNF rules, and $G_R = (N_R, E_R)$ the reduced behaviour graph for $R$, the set of global $\Box$-rules in $T$. For $l$, a literal in $T$, $G_T = (N_T, E_T)$ is a *terminal subgraph where* $l$ *holds* if and only if

(1) $G_T \subseteq G_R$ and $G_T \neq \emptyset$, i.e., $N_T \subseteq N_R$ and $E_T \subseteq E_R$ such that for all $e = (n, n') \in E_T$, $n, n' \in N_T$,

(2) if $n \in N_T$ then $n \models l$, i.e., $l$ holds at each node, and

(3) if $n \in N_T$ then $\nexists e \in E_R$ such that $e = (n, n')$ and $n' \notin N_T$, i.e., $G_T$ is terminal.

**Definition 10** (maximal terminal subgraph where $l$ holds). Let $T$ be a set of SNF rules, $R$ the set of global $\Box$-rules of $T$, and $G_R$ the reduced behaviour graph for $R$. A terminal subgraph $G_T = (N_T, E_T)$ of $G_R$ is said to be a *maximal terminal subgraph where $l$ holds* if and only if for all terminal subgraphs $G'_T = (N'_T, E'_T)$ of $G_R$ where $l$ holds $N'_T \subseteq N_T$ and $E'_T \subseteq E_T$.

The union of two terminal subgraphs is also a terminal subgraph (even if the two subgraphs may have no nodes in common). Thus the maximal terminal subgraph where $l$ holds will be unique. Relating back to the set of rules $\{\bigcirc a \Rightarrow a, \bigcirc a \Rightarrow l, \bigcirc a \Rightarrow \neg a\}$ discussed earlier that contains a loop but not a proper loop, the reduced behaviour graph for this set of rules does not contain a terminal subgraph where $l$ holds. The aim is to show that maximal proper loops correspond to maximal terminal subgraphs.

Next we establish a link between rules and graphs.

**Lemma 11.** Let $T$ be a set of SNF rules, $R$ the set of global $\Box$-rules of $T$, and $G_R$ the reduced behaviour graph for $R$. If $R$ contains rules that may be combined to form $L$ a maximal proper loop in $l$, of the form $\bigcirc A_i \Rightarrow B_i$, for $i = 0$ to $m$, then $G_R$ contains a maximal terminal subgraph $G_M = (N_M, E_M)$ where $l$ holds and

$$n \in N_M \quad \text{iff} \quad n \models \bigvee_{i=0}^{m} A_i \wedge l.$$

The operation of the breadth-first search algorithm is then related to sets of nodes in the reduced behaviour graph. Given a node, $H_i$, in breadth-first search the corresponding set of nodes in the reduced behaviour graph, $N_{H_i}$, is those nodes $n$ such that $n \models H_i$. To construct $N_{H_{i+1}}$ using the reduced behaviour graph alone we remove any nodes from $N_{H_i}$ that have edges leading to nodes that are not members of $N_{H_i}$. Termination occurs when either the set of nodes becomes empty (equivalent to not been able to construct a next node in breadth-first search) or when we have a non-empty set of nodes and cannot delete any more, i.e., the set is a terminal subgraph (corresponding to the equivalence of the last two nodes in breadth-first search). As each set in the reduced behaviour graph is a subset of the previous one the algorithm will terminate. The following three lemmata were originally shown [11] relating steps in the breadth-first search algorithm to steps in an algorithm on the reduced behaviour graph itself. Here, we use the result that the algorithm on the graph itself detects the maximum terminal subgraph where $l$ holds and present the results relating to $N_B$ the set of nodes representing the output of breadth-first search.

**Lemma 12.** Let $T$ be a set of SNF rules and $G_R = (N_R, E_R)$ the reduced behaviour graph for $R$ the set of global $\Box$-rules in $T$. Let $N_B = \{n \mid n \in N_R \text{ and } n \models D\}$ where $D$ is the DNF formula output by the breadth-first search algorithm. If $G_R$ contains a maximal terminal subgraph where $l$ holds, $G_M = (N_M, E_M)$, then

$$(n \in N_B \wedge n \models l) \Rightarrow n \in N_M.$$

**Lemma 13.** Let $T$ be a set of SNF rules and $G_R = (N_R, E_R)$ the reduced behaviour graph for $R$ the set of global $\Box$-rules in $T$. Let $N_B = \{n \mid n \in N_R \text{ and } n \models D\}$ where $D$ is the DNF formula output by the breadth-first search algorithm. If $G_R$ contains a maximal terminal subgraph where $l$ holds, $G_M = (N_M, E_M)$, then

$$\big(n \in N_B \wedge \big(n, n'\big) \in E_R\big) \Rightarrow n' \in N_M.$$

**Lemma 14.** Let $T$ be a set of SNF rules and $G_R = (N_R, E_R)$ the reduced behaviour graph for $R$ the set of global $\Box$-rules in $T$. If $G_R$ contains a maximal terminal subgraph where $l$ holds, $G_M = (N_M, E_M)$, then the breadth-first search algorithm yields a DNF formula $D$ such that if $N_B = \{n \mid n \in N_R \text{ and } n \models D\}$ then $N_M \subseteq N_B$.

**Theorem 15** (soundness for the breadth-first search algorithm). If, given a set of SNF rules, $T$, the breadth-first search algorithm terminates detecting a DNF-formula $D$ for resolution with a $\Diamond$-rule in $T$ with the eventuality $\Diamond\neg l$ on the right hand side then

$$\text{◎}D \Rightarrow \Box l.$$

*Proof.* Let $G_R = (N_R, E_R)$ be the reduced behaviour graph for $R$ the set of global $\Box$-rules of $T$. Let $N_B = \{n \mid n \in N_R \text{ and } n \models D\}$ and $N_M$ is the maximal terminal subgraph where $l$ holds then, from lemma 14, $N_M \subseteq N_B$.

Now as $N_M$ is terminal and from lemma 13 we have

$$\big(n \in N_B \wedge \big(n, n'\big) \in E_R\big) \Rightarrow n' \in N_M,$$

i.e., every edge out of every node in $N_B$ leads to a node that is in $N_M$ so $N_B$ is also terminal. From the definition of the maximal terminal subgraph where $l$ holds if $n \in N_M$ then $n \models l$ and as $n \in N_B$ iff $n \models D$, from lemma 13

$$\big(n \models D \wedge \big(n, n'\big) \in E_R\big) \Rightarrow n' \in N_M$$

then $D \Rightarrow \bigcirc l$. So as $N_B$ is terminal and $l$ holds at each next node in $N_B$ then $D \Rightarrow \bigcirc\Box l$ or $\text{◎}D \Rightarrow \Box l$ as required. □

Given a maximal proper loop $\text{◎}A_i \Rightarrow B_i$, for $i = 0$ to $m$ to apply the temporal resolution rule we require $\bigwedge_{i=0}^{m} \neg A_i$. Thus it is sufficient for completeness to ensure the output from the breadth-first search algorithm, $D$, is equivalent to $\bigvee_{i=0}^{m} A_i$. Actually the completeness proof shows that, assuming we are looking for a loop in $l$, the two formulae are equivalent if $l$ holds. In terms of the graph adding the resolvents from

$\bigcirc\!\!\!\!\bigcirc A_i \Rightarrow B_i$ means that any edge into any node where $\bigvee_{i=0}^{m} A_i$ holds and $l$ holds, i.e., $\neg l$ is an outstanding eventuality is removed [36]. Likewise adding the resolvents from the DNF formula output by breadth-first search, $D$, any edge into any node where $D$ and $l$ holds is removed. So ensuring that

$$\models l \Rightarrow \left(D \Leftrightarrow \bigvee_{i=0}^{m} A_i\right)$$

means the resolvents from either $\bigcirc\!\!\!\!\bigcirc A_i \Rightarrow B_i$ or $D$ will remove the same edges from the reduced behaviour graph and is therefore sufficient for completeness.

**Theorem 16** (sufficient completeness for the breadth-first search algorithm). Given a set of SNF rules $T$ where $R$ the set of global $\Box$-rules of $T$ that contains rules that may be combined together to form a maximal proper loop in $l$, $L$, of the form $\bigcirc\!\!\!\!\bigcirc A_i \Rightarrow B_i$ for $i = 0$ to $m$ then by applying the breadth-first search algorithm we can find a DNF formula $D$ such that

$$\models l \Rightarrow \left(D \Leftrightarrow \bigvee_{i=0}^{m} A_i\right).$$

*Proof.* As $R$ contains a maximal proper loop in $l$, from lemma 11, the behaviour graph for $R$, $G_R = (N_R, E_R)$, contains a maximal terminal subgraph where $l$ holds, $G_M = (N_M, E_M)$. The breadth-first search algorithm outputs $D$ so let $N_B = \{n \mid n \in N_R \text{ and } n \models D\}$. We have the following:

- $n \in N_M$ iff $n \models \bigvee_{i=0}^{m} A_i \wedge l$ (lemma 11);
- $N_M \subseteq N_B$ (lemma 14);
- $(n \in N_B \wedge n \models l) \Rightarrow n \in N_M$, from lemma 12; and
- $n \in N_M \Rightarrow n \models l$ from the definition of the maximal terminal subgraph where $l$ holds.

So we have

$$(n \in N_B \wedge n \models l) \Leftrightarrow n \in N_M,$$

i.e., the set of nodes in $N_B$ where $l$ also holds is identical to the set of nodes $N_M$. Recall each node in the reduced behaviour graph is merely a valuation, so, for all valuations $n$,

$$n \models D \wedge l \text{ iff } n \models \bigvee_{i=0}^{m} A_i \wedge l, \quad \text{so} \quad \models D \wedge l \Leftrightarrow \bigvee_{i=0}^{m} A_i \wedge l$$

and

$$\models l \Rightarrow \left(D \Leftrightarrow \bigvee_{i=0}^{m} A_i\right)$$

as required. □

### 5.2. Complexity

Given a set of rules $R$ let

- $n$ be the number of propositions; and
- $m$ be the number of rules.

The breadth-first search algorithm first looks for rules of the form $\bigcirc A \Rightarrow l$, assuming we are trying to find a loop-formula in $l$. Then, to construct the next node we must find the rules or combinations of rules of the form $\bigcirc A \Rightarrow B$ where $\vdash B \Rightarrow H_i$ and $\vdash A \Rightarrow H_0$ where $H_i$ is the previous node constructed. Now in the worst case we will have to take all the combinations of all $m$ rules which generates O($2^m$) rules. For each rule we must check the above implications hold. Now for each implication test there can be at most $n$ propositions so we can check each implication holds by generating the truth table which will have $2^n$ lines. So the worst case complexity for building a node is $2^m(2^n + 2^n)$, i.e., O($2^{m+n}$).

Now we must consider how many nodes we can generate. Recall the relationship of breadth-first search to the behaviour graph. As we construct a new node in breadth-first search we delete any nodes that have edges taking us out of the current set of nodes in the behaviour graph. The worst case we could have is if the nodes in the behaviour graph are in a sequence, the first pointing to the second, the second to the third, etc. At most we may have $2^n$ nodes and only half of them can contain the required literal so in the worst case it will take $2^{n-1}$ rounds of breadth-first search to terminate. We must also add the cost of the termination test but this will be dominated by the other factor, i.e., complexity of breadth-first search is O($2^{m+2n}$).

## 6. Analysis and implementation

In this section the characteristics of loop-formulae detected by breadth-first search are discussed and the details of an implementation given.

### 6.1. Loop search characteristics

We examine the operation of the breadth-first search algorithm and the loop-formulae it detects. The breadth-first search algorithm finds the most general loop-formula for a particular $\Diamond$-rule. Using example 1 in section 4.5 we first consider loops in the set of rules and relate to output from the breadth-first search algorithm. Considering example 1 it is clear that the three SNF$_m$ rules,

$$\begin{array}{ll} M1 & \bigcirc(a \wedge e) \Rightarrow (a \wedge e \wedge \neg l) \\ M2 & \bigcirc b \Rightarrow (b \wedge \neg l) \\ M3 & \bigcirc c \Rightarrow (b \wedge \neg l) \end{array}$$

satisfy the criteria for a loop together. We could combine some of these SNF$_m$ rules further to give other SNF$_m$ rules; for example, combining rules $M1$ and $M2$ we obtain

$$M4 \quad \bigcirc(a \wedge b \wedge e) \Rightarrow (a \wedge b \wedge e \wedge \neg l),$$

so that rules $M1$–$M4$ together form a loop. However this loop provides no more information than the loop with just rules $M1$–$M3$. By lemma 6 the loop from rules $M1$–$M3$ is an equivalent but simpler loop than that formed from rules $M1$–$M4$. Performing breadth-first search we return the loop-formula $(a \wedge e) \vee b \vee c$ as shown previously (because of simplification we will never return $(a \wedge e) \vee b \vee c \vee (a \wedge b \wedge e)$, for example).

Breadth-first search finds disjuncts representing the 'lead into the loop'. An example of this is the disjunct $c$ in the loop-formula output in example 1 representing the use of the SNF$_m$ rule $\bigcirc c \Rightarrow (b \wedge \neg l)$ ($M3$). This rule represents the path into the loop (there are no rules that make this rule 'fire'). The loop represented by the rules $M1$, $M2$ and $M3$ above still satisfy the side conditions imposed on loops given in section 3.4. However if the loop just relating to rules $M1$ and $M2$ (i.e., without the lead into the loop) was used for resolution instead of the $M1$–$M3$ loop the system would still be complete, however we would require additional applications of the step resolution rule to generate rules equivalent to the complete set of resolvents from breadth-first search.

Breadth-first search returns the maximal loop-formula. Again considering example 1 $M1$ is a loop on its own as is $M2$ or $M2$ and $M3$ together. Breadth-first search returns the maximal loop-formula (modulo equivalence) in this case $(a \wedge e) \vee b \vee c$ rather than just $a \wedge e$, $b$, or $b \vee c$, respectively.

Usually, a disadvantage of breadth-first search algorithms in general is the amount of memory required to construct the search space. The breadth-first search algorithm described here does attempt to use *all* the rules or combinations of rules to construct the next node but is different from general breadth-first search style algorithms in the following ways. Firstly, the breadth-first search algorithm only requires the storage of the top node, $H_0$, and the last node constructed, $H_i$, to enable the construction of node $H_{i+1}$ and to test for termination. Secondly, each node constructed is a DNF formula represented in a simple form so any redundant information is removed from the node at an early stage. Finally, although the number of rules in which we search for loop-formulae may be large and each loop-formulae may contain many propositions, hopefully, the loop-formulae found will relate to only a few of these rules containing a subset of the total number of propositions.

### *6.2. Implementation*

A prototype implementation performing the temporal resolution method has been built. The programs are written in SICStus Prolog [8] running under UNIX. The algorithm has been tested on a set of valid temporal formulae taken from [30] chosen as it is a reasonably sized collection of small problems to be proved valid. Larger

examples have also been tackled, for example Peterson's algorithm [37,40]. Timings relating breadth-first search to a previously described algorithm are given in [12]. The algorithm is based on a depth-first search approach, i.e., nodes are expanded using one $\text{SNF}_m$ rule rather than as many as possible. Its description is beyond the scope of this paper but for more details see [11]. Results from timings show that breadth-first search performs better in significantly more examples than depth-first search. The percentage of examples where breadth-first search is quicker than depth-first search increases if low timings are progressively excluded, suggesting that breadth-first search performs better on larger examples. Further, the raw data shows that timings for depth-first search have a larger range of values than breadth-first search. This is what we would expect from depth-first search style algorithms because if the correct search path is chosen first the solution *can* be detected more quickly than breadth-first search type algorithms. However, if a large amount of time is spent exploring fruitless paths then the overall time may be far greater than that for breadth-first search.

## 7. Conclusions, related and future work

### *7.1. Related work*

Here we consider other resolution systems for PTL, some of the theorem provers available for PTL and mention general breadth-first search algorithms. Other methods for applying resolution to PTL have been developed in [2,9,45].

In [1,2] a non-clausal resolution method is described for propositional and first-order linear time temporal logics. As it is non-clausal many inference rules are necessary leading to problems with proof direction. Rules are defined for simplification, distribution, cut, resolution, modality and induction. The simplification and distribution rules are similar to those used in Fisher's system so that rules are kept in their simplest form. The cut rule allows the introduction of rules of the form $C \vee \neg C$ and is not necessary for the completeness of the propositional system (but is necessary for the first-order system). The resolution rule is of the form

$$A\langle C, \ldots, C\rangle, B\langle C, \ldots, C\rangle \longrightarrow A\langle \textbf{true}\rangle \vee B\langle \textbf{false}\rangle$$

where $A\langle C, \ldots, C\rangle$ denotes that the subformula $C$ occurs one or more times in $A$. Here occurrences of $C$ in $A$ and $B$ are replaced with **true** and **false** respectively. To ensure the rule is sound each $C$ that is replaced must be in the scope of the same number of $\bigcirc$-operators, and must not be in the scope of any other modal operator in $A$ or $B$, i.e., they must apply to the same moment in time. This is similar to Fisher's step resolution rules except through writing to a normal form we are sure that rules refer to the same moment in time. The modality rules apply to formulae in the scope of the temporal operators. For example, the $\Box$-rule allows any formula $\Box C$ to be rewritten as $C \wedge \bigcirc \Box C$. This type of rule is applied in Fisher's system during the

translation to the normal form. The induction rule deals with the interaction between $\bigcirc$ and $\Box$ and is of the form

$$D, \Diamond C \longrightarrow \Diamond\big(\neg C \wedge \bigcirc(C \wedge \neg D)\big) \quad \text{if } \vdash \neg(D \wedge C).$$

Informally this means that if $D$ and $C$ cannot both hold at the same time and if $D$ and $\Diamond C$ hold now then there must be a moment in time (now or) in the future when $C$ does not hold and at the next moment in time $C$ holds and $D$ does not. A proof editor has been developed for the propositional system with the $\bigcirc$, $\Box$ and $\Diamond$ operators. The resolution system is then extended to allow for the operators $\mathcal{W}$ and $\mathsf{P}$ (where $\mathsf{P}$ is known as *precedes* and $C\,\mathsf{P}\,D = \neg((\neg C)\,\mathcal{W}\,D)$) also. A decision procedure based on tableau is given and completeness for the resolution systems are shown by proving that if a formula $\neg C$ is found unsatisfiable by the tableau decision procedure then there is a refutation for $\neg C$.

Although a proof editor has been developed for the restricted propositional system it seems unlikely that Abadi's system lends itself to a fully automatic implementation. This is because of the large number of rules that may be applied. The modality rules for example deal with subformulae in the scope of modal operators, required as the system is non-clausal. Further, the induction rule requires a proof as a side condition to its usage which will make automatic proofs difficult.

The system described in [9] rewrites formulae to a rather complex normal form. Resolution rules are of the form 'that $\Box A$ and $\Diamond B$ can be resolved if $A$ and $B$ are resolveable and the resolvent will be the resolvent of $A$ and $B$ with a $\Diamond$-operator in front'. There are three types of inference rules of the form

$$\frac{C_1 \vee C \quad C_2 \vee C'}{R(C_1, C_2) \vee C \vee C'}$$

if $C_1$ and $C_2$ are resolvable. Formulae are refuted by translation to clausal form and repeated application of the inference rules. Resolution only takes place between clauses in the context of certain operators outlined in the resolution rules.

The method is only similar to Fisher's method as it uses translation to a clause form, although the normal form is much more complicated, and attempts to resolve $\Box$ and $\Diamond$-rules by using the above rules. As $\Box$-formulae are allowed in the clausal form no loop search algorithms similar to that described in this paper are necessary. The range of temporal operators is more limited than that used here, in particular it does not include the $\mathcal{U}$-operator making the logic less expressive [21,26]. Further, it is not clear whether the completeness proof for the method would easily extend to the more general case with $\mathcal{U}$. Note, that formulae containing $\mathcal{U}$ are allowed in Fisher's method even though the translation to SNF rewrites such formulae as a series of global $\Box$-rules and a $\Diamond$-formula, by the introduction of new proposition symbols. The set of formulae in SNF can be used to test for unsatisfiability as detecting unsatisfiability in the rules in SNF implies unsatisfiability of the original formula [15–17].

Venkatesh describes a resolution method for PTL in [45]. Again a clausal method, formulae are *unwound* into those that hold now and in the next moment in time. In

this way reasoning is done from the start state forwards in time. This compares to Fisher's method where reasoning is carried out backwards and contradictions within states generate constraints on previous states until a contradiction in the initial state is obtained.

DP and DPP [22] are decision procedures for propositional linear time temporal logic. DP has a model of time with a finite past (as here) and the same set of future-time temporal operators whilst DPP has both infinite past and future and the full range of future and past time temporal operators. It is based on the tableau method and is similar, although not identical, to the system presented by Wolper in [46].

A decision procedure for propositional linear-time temporal logic is available as a module of the Logics Workbench (LWB) [25]. Using this tool formulae can be shown to be valid or satisfiable. Models are constructed for satisfiable formulae and countermodels can be constructed for formulae that are shown not to be valid. Again the method is tableau based with a check for unsatisfied eventualities.

STeP [6] is system that combines both model checking and deductive methods. The model checking component allows the automatic verification of properties against their input systems. The deductive element allows the verification of systems where model checking is infeasible or impossible. The general approach here is one of model checking rather than theorem proving, that is a property is verified against a model of the behaviour of the input system. The model checking component allows automatic verification of properties however in the deductive component user interaction is often required. Decision procedures are incorporated to check the validity of a large class of formulae automatically.

Standard breadth-first search algorithms are described in AI texts such as [42] for controlling search in a state space. Breadth-first search algorithms are such that the expansion of each node in a particular level of the breadth-first search tree occurs before moving on to the expansion of nodes at the following level. The advantages are that the search algorithm won't get stuck down *blind alleys*, i.e., we keep expanding a path that contains no solution, and any solution discovered will be minimal. The disadvantages are that the complete search tree constructed so far must be kept in memory and we lose the chance of obtaining a quick solution if one exits that may be discovered using depth-first algorithms. Although the search algorithm presented in this paper may be called breadth-first search in that we attempt to use *all* possible rules or combinations of rules (without the duplication of information) to expand from a particular DNF formula it differs from the standard breadth-first search in a number of ways. Firstly we do not need to keep the complete search space in memory as we only require the top node and previous node to apply the breadth-first search algorithm. Secondly the information from applying each rule or combination of rules is not really a separate move in a search tree. Rather we are looking for all the rules that satisfy certain criteria and we disjoin and simplify their left hand sides and consider the resulting DNF formula "one" piece of information. Finally for termination we require the last two DNF formulae to be equivalent and return the resulting DNF formula rather than just one solution or search path.

### 7.2. Conclusions

We have described the approach to clausal resolution in PTL as suggested by Fisher and given an algorithm with which to implement the temporal resolution rule. Given an eventuality, $\Diamond l$, the temporal resolution rule requires that we find a DNF formula such that once this formula is satisfied then $\neg l$ always holds. The breadth-first search algorithm presented attempts to construct the loop-formulae piecemeal only using rules that both generate the required literal and preserve looping whilst keeping the structure constructed in its simplest form. This is done by constructing a series of DNF formulae such that if the DNF formula is satisfied then $\neg l$ holds at the next moment in time, terminating when the last formula in the series is equivalent to its predecessor. Soundness and completeness are proved for the breadth-first search algorithm and show that the maximal, most general loop-formulae are detected. The complexity of the algorithm has been analysed.

A prototype temporal resolution theorem prover using the breadth-first algorithm has been implemented and tested on examples. The algorithm is an improvement on those suggested previously [12] for the following reasons. Firstly, it attempts to construct the search structure on the fly using only the relevant rules rather than to construct a complex structure representing the global $\Box$-rules first and *then* search for loops (or loop-formulae). In this way the worst case complexity is avoided in many cases. Secondly, the algorithm detects the maximal loop-formula, avoiding the necessity to explore parts of the search graph already explored to detect a new loop that the breadth-first search algorithm would have found in one attempt. The information required for the search is kept to a minimum as we require only the top node and the previous node to construct the next node and test for termination and each of these nodes is kept in a simple form. Other searches suggested require complex structures to be kept in memory.

Our approach is resolution rather than tableau based so we are not limited to finite state problems and can use strategies to guide the application of the resolution rules. Although other temporal resolution systems are outlined in section 7.1 we feel our system has the following advantages. It is applicable to 'full' PTL with the usual range of temporal operators, i.e., not just to $\bigcirc$, $\Diamond$ and $\Box$ as [9] is whilst remaining complete. Also it is suitable for automatic implementation as there is just one temporal resolution rule and does not suffer from problems with proof direction due to the number of resolution rules as [1,2].

### 7.3. Future work

The current Prolog system has been useful for experimenting with the loop search algorithms we have developed. However an efficient implementation for the full theorem prover in a *fast* language with a user friendly interface is desirable. Then a range of more challenging examples can be tried and a comparison with other temporal resolution theorem provers may be realistically attempted.

We have been looking at applying Fisher's method to logics other than PTL. We have already extended to method to deal with temporal logics of knowledge and belief [20] and the branching time logic CTL [7] and are currently working on further extensions.

Finally we are looking at heuristics and strategies for the temporal resolution method. In [13] we provide an algorithm to remove rules from the set of global $\Box$-rules that can never form part of the loop prior to loop search. The development of a range of strategies to reduce the search space is essential in order to tackle larger problems efficiently.

## Acknowledgements

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

## References

[1] M. Abadi, Temporal-logic theorem proving, Ph.D. thesis, Department of Computer Science, Stanford University (1987).

[2] M. Abadi and Z. Manna, Nonclausal deduction in first-order temporal logic, Journal of ACM 37(2) (1990) 279–317.

[3] H. Barringer, Using temporal logic in the compositional specification of concurrent systems, in: *Temporal Logics and their Applications*, ed. A.P. Galton, Chapter 2 (Academic Press, London, 1987) pp. 53–90.

[4] H. Barringer, M. Fisher, D. Gabbay, R. Owens and M. Reynolds, eds., *The Imperative Future* (Research Studies Press, 1996).

[5] H. Barringer, R. Kuiper and A. Pnueli, Now you may compose temporal logic specifications, in: *Proc. Sixteenth ACM Symp. Theory of Computing* (1984).

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

[7] A. Bolotov and M. Fisher, A resolution method for CTL branching-time temporal logic, in: *Proc. TIME '97 the Fourth International Workshop on Temporal Representation and Reasoning*, Daytona Beach, FL (IEEE Computer Society Press, 1997).

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

[9] A. Cavalli and L. Fariñas del Cerro, A decision method for linear temporal logic, in: *Proc. 7th International Conference on Automated Deduction* (*CADE*), ed. R.E. Shostak, Lecture Notes in Computer Science 170 (Springer, 1984) pp. 113–127.

[10] C.L. Chang and R.C.T. Lee, *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, 1973).

[11] C. Dixon, Strategies for temporal resolution, Ph.D. thesis, Department of Computer Science, University of Manchester (1995).

[12] C. Dixon, Search strategies for resolution in temporal logics, in: *Proc. Thirteenth International Conference on Automated Deduction (CADE)*, New Brunswick, NJ, eds. M.A. McRobbie and J.K. Slaney, Lecture Notes in Artificial Intelligence 1104 (Springer, 1996) pp. 672–687.
[13] C. Dixon, Temporal resolution: removing irrelevant information, in: *Proc. TIME '97 the Fourth International Workshop on Temporal Representation and Reasoning*, Daytona Beach, FL (IEEE Computer Society Press, 1997).
[14] R. Fagin, J.Y. Halpern, Y. Moses and M.Y. Vardi, *Reasoning About Knowledge* (MIT Press, 1995).
[15] M. Fisher, A resolution method for temporal logic, in: *Proc. Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, Sydney, Australia (Morgan Kaufmann, 1991).
[16] M. Fisher, A normal form for first-order temporal formulae, in: *Proc. Eleventh International Conference on Automated Deduction (CADE)*, Saratoga Springs, NY, Lecture Notes in Computer Science 607 (Springer, 1992).
[17] M. Fisher and P. Noël, Transformation and synthesis in METATEM – Part I: Propositional METATEM. Technical Report UMCS-92-2-1, Department of Computer Science, University of Manchester, Manchester, UK (1992).
[18] M. Fisher and R. Owens, eds., *Executable Modal and Temporal Logics*, Lecture Notes in Artificial Intelligence 897 (Springer, 1995).
[19] M. Fisher and M. Wooldridge, On the formal specification and verification of multi-agent systems, International Journal of Cooperative Information Systems 6(1) (1997).
[20] M. Fisher, M. Wooldridge and C. Dixon, A resolution-based proof method for temporal logics of knowledge and belief, in: *Proc. International Conference on Formal and Applied Practical Reasoning (FAPR '96)*, Bonn, Germany (1996).
[21] D. Gabbay, A. Pnueli, S. Shelah and J. Stavi, The temporal analysis of fairness, in: *Proc. Seventh ACM Symp. Principles of Programming Languages*, Las Vegas, NE (1980) pp. 163–173.
[22] G.D. Gough, Decision procedures for temporal logic, Master's thesis, Department of Computer Science, University of Manchester, October 1984. Also University of Manchester, Department of Computer Science, Technical Report UMCS-89-10-1.
[23] B.T. Hailpern, *Verifying Concurrent Processes Using Temporal Logic*, Lecture Notes in Computer Science 129 (Springer, 1982).
[24] J.Y. Halpern, Using reasoning about knowledge to analyze distributed systems, Annual Review of Computer Science 2 (1987).
[25] G. Jaeger, A. Heuerding, S. Schwendimann, F. Achermann, P. Balsiger, P. Brambilla, H. Zimmermann, M. Bianchi, K. Guggisberg and W. Heinle, LWB – The Logics Workbench 1.0, University of Berne, Switzerland, http://lwbwww.unibe.ch:8080/LWBinfo.html.
[26] J.A.W. Kamp, Tense logic and the theory of linear order, Ph.D. thesis, University of California (1968).
[27] R. Kowalski, Predicate logic as a programming language, in: *Proc. International Federation for Information Processing*, ed. J.L. Rosenfield (North-Holland, 1974) pp. 569–574.
[28] R. Kowalski, *Logic for Problem Solving* (North-Holland, 1979).
[29] L. Lamport, Specifying concurrent program modules, ACM Transactions on Programming Languages and Systems 5(2) (1983) 190–222.
[30] Z. Manna and A. Pnueli, Verification of concurrent programs: The temporal framework, in: *The Correctness Problem in Computer Science*, eds. R.S. Boyer and J.S. Moore (Academic Press, London, 1981) pp. 215–273.
[31] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification* (Springer, New York, 1992).
[32] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety* (Springer, New York, 1995).
[33] W.W. McCune, *OTTER 2.0 Users Guide*, ANL-90/9, Argonne National Laboratory, Argonne, IL (1990).

[34] B. Moszkowski, Reasoning about digital circuits, Ph.D. thesis, Department of Computer Science, Stanford University (1983).
[35] S. Owicki and L. Lamport, Proving liveness properties of concurrent programs, ACM Transactions on Programming Languages and Systems 4(3) (1982) 455–495.
[36] M. Peim, Propositional temporal resolution over labelled transition systems, Unpublished Technical Note, Department of Computer Science, University of Manchester (1994).
[37] G.L. Peterson, Myths about the mutual exclusion problem, Information Processing Letters 12(3) (1981) 115–116.
[38] A. Pnueli, The temporal logic of programs, in: *Proc. Eighteenth Symp. Foundations of Computer Science*, Providence (1977).
[39] A. Pnueli, The temporal semantics of concurrent programs, Theoretical Computer Science 13 (1981) 45–60.
[40] A. Pnueli, In transition from global to modular temporal reasoning about programs, in: *Logics and Models of Concurrent Systems*, ed. K. Apt, La Colle-sur-Loup, France (NATO, Springer, 1984) pp. 123–144.
[41] A. Pnueli, Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends, in: *Current Trends in Concurrency*, eds. J.W. de Bakker, W.P. de Roever and G. Rozenberg, Lecture Notes in Computer Science 224 (Springer, 1986).
[42] E. Rich and K. Knight, *Artificial Intelligence* (McGraw-Hill, 1991).
[43] J.A. Robinson, A machine-oriented logic based on the resolution principle, Journal of ACM 12(1) (1965) 23–41.
[44] M.Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification, in: *Proc. IEEE Symp. Logic in Computer Science*, Cambridge (1986) pp. 332–344.
[45] G. Venkatesh, A decision method for temporal logic based on resolution, Lecture Notes in Computer Science 206 (1986) pp. 272–289.
[46] P. Wolper, The tableau method for temporal logic: An overview, Logique et Analyse 110–111 (1985) 119–136.