Resolution for Branching Time Temporal Logics: Applying the Temporal Resolution Rule

Alexander Bolotov and Clare Dixon
Department of Computing and Mathematics
Manchester Metropolitan University
Manchester M1 5GD, UK.
{*A.Bolotov,C.Dixon*}*@doc.mmu.ac.uk*

Abstract

In this paper we propose algorithms to implement a branching time temporal resolution theorem prover. The branching time temporal logic considered is Computation Tree Logic (CTL)*, often regarded as the simplest useful logic of this class. Unlike the majority of the research into temporal logic, we adopt a resolution-based approach. The method applies* step *and* temporal *resolution rules to the set of formulae in a normal form. Whilst step resolution is similar to the classical resolution rule, the temporal resolution rule resolves a formula, φ, that must eventually occur with a* set of formulae *that together imply that φ can never occur. Thus the method is dependent on the efficient detection of such sets of formulae. We present algorithms to search for these sets of formulae, give a correctness argument, and examples of their operation.*

1 Introduction

Since it was first proposed in [13] temporal logics have been used extensively in the specification and verification of properties of concurrent and distributed systems [9]. If in this application the ability to refer to a range of possible execution paths is important then the power of the language of branching-time logics is essential [9]. It has been observed that most correctness properties of concurrent programs (that do not deal with fairness) can be expressed in the family of branching time logics called Computation Tree Logics. The core logic we concentrate on is a Computation Tree Logic (CTL) [4] often regarded as the simplest useful logic of this family. There are several extensions of CTL of which CTL* is the most powerful [8].

Much of the research on the verification of concurrent and distributed systems has centered around the *model-checking* technique utilising CTL [9]. Here the satisfiability of a CTL formula is checked with respect to a model derived from a finite-state program [9]. The underlying research on decision procedures for branching time temporal logics has mostly involved tableau or automata methods [8] with an obvious lack of research into deductive proof methods.

For propositional *linear-time* temporal logics (PLTL) a clausal resolution method [10] has been developed. This resolution approach has been extended to CTL in [2, 3]. Its key elements consist of translation to a normal form and a variety of resolution rules. The normal form is a set of formulae which utilize only "next", "always" and "sometime" temporal operators (all other operators are subsumed within this representation). Two types of resolution rules are distinguished, namely, resolution within states known as *step* resolution, and resolution over states known as *temporal* resolution. The latter applies when a proposition l occurs at all future moments (known as a *loop* in l), and if l is also constrained to be false at some point in the future. The efficient search for such *loops* is crucial to the temporal resolution method. In the linear-time case, several algorithms for detecting loops have been developed [5, 6]. In this paper we select one of these loop detection algorithms and extend it to apply to the branching time logic CTL.

The remaining of the paper is organized as follows. In §2 we overview the logic CTL. In §3 a clausal resolution method is outlined. Algorithms to apply the temporal resolution rule and examples of their operation are given in §4. A correctness argument is outlined in §5. Finally, in §6 we consider related work and in §7 provide concluding remarks.

2 The logic CTL

Here we summarize the syntax and semantics of the core logic, CTL.

2.1. Syntax and semantics of CTL

In the language of CTL we utilize only future-time operators[1] $\Box$ (*always*), $\Diamond$ (*sometime*), $\bigcirc$ (*next time*), $\mathcal{U}$ (*until*) and $\mathcal{W}$ (*unless*). Additionally, we use path quantifiers **A** (*on all future paths*) and **E** (*on some future path*).

The syntax of CTL distinguishes *state* (S) and *path* (P) formulae. These are defined inductively as follows, where C is any well-formed formula of propositional logic, including classically defined constants **true** and **false**.

$$S ::= C \mid S \wedge S \mid S \vee S \mid S \Rightarrow S \mid \neg S \mid \mathbf{A}P \mid \mathbf{E}P$$
$$P ::= \Box S \mid \Diamond S \mid \bigcirc S \mid S\,\mathcal{U}\,S \mid S\,\mathcal{W}\,S$$

Well formed formulae of CTL are state formulae. Thus, each CTL formula has a structure where any temporal operator can only be followed by a path operator or a classical operator, while any path operator can only be followed by a temporal operator. An example of a CTL formula is $\mathbf{E}\Diamond p \wedge \mathbf{E}\Box p$ meaning 'there is a path on which p eventually holds and there is a path on which p always holds'. It follows that CTL is weaker than linear-time temporal logic in its expressive capabilities within a path, but is more expressive in that it can quantify over paths themselves. As an example of its restricted nature, note that no formula describing the property "both $\Diamond A$ and $\Box B$ are satisfied on the *same* specific path" can be constructed using CTL syntax. For the detailed description of CTL's theoretical properties, subsystems and extensions see [8].

Before continuing with the semantics of CTL we introduce some notation. We interpret a well-formed formula of CTL in a tree-like model structure $\mathcal{M} = \langle S, R, L\rangle$, where S is a set of states, $R \subset S \times S$ is a binary relation over S and L is an interpretation function mapping atomic propositional symbols to truth values at each state.

A *path*, χ_{s_i}, over R, is a sequence of states $s_i, s_{i+1}, s_{i+2}, \ldots$ such that for all $j \geq i$, $(s_j, s_{j+1}) \in R$. A path χ_{s_0} is called a *fullpath*.

Given a path χ_{s_i} and a state $s_j \in \chi_{s_i}$, $i < j$ we term a finite subsequence $s_i, s_{i+1}, \ldots, s_j$ a *prefix* of a path χ_{s_i} and an infinite sub-sequence of states $s_j, s_{j+1}, s_{j+2}, \ldots$ a *suffix* of a path χ_{s_i} abbreviating these respectively with $Pref(\chi_{s_i}, [s_i, s_j])$ (or simply as $[s_i, s_j]$ when it is clear which path this prefix belongs to) and $Suf(\chi_{s_i}, s_j)$.

We assume that a CTL model $\mathcal{M}$ satisfies the following conditions:

1. there is a designated state, $s_0 \in S$, a root of a structure (i.e. for all j, $\langle s_j, s_0\rangle \notin R$)),
2. every state belongs to some fullpath, i.e. a path starting at s_0,
3. tree structures are of at most countable branching,
4. every state should have a successor state, and,
5. every path is isomorphic to ω.

Below we define the satisfaction relation '$\models$' which evaluates well-formed CTL formulae at a state s_i in a model $\mathcal{M}$. Postulates s1-s7 define satisfaction relation for the CTL formulae at states while p1-p6 determine evaluation of their subformulae along paths.

s1 $\langle \mathcal{M}, s_i\rangle \models p$ iff $p \in L(s_i)$, for atomic p.

s2 $\langle \mathcal{M}, s_i\rangle \models \neg A$ iff $\langle \mathcal{M}, s_i\rangle \not\models A$.

s3 $\langle \mathcal{M}, s_i\rangle \models A \wedge B$ iff $\langle \mathcal{M}, s_i\rangle \models A$ and $\langle \mathcal{M}, s_i\rangle \models B$.

s4 $\langle \mathcal{M}, s_i\rangle \models A \vee B$ iff $\langle \mathcal{M}, s_i\rangle \models A$ or $\langle \mathcal{M}, s_i\rangle \models B$.

s5 $\langle \mathcal{M}, s_i\rangle \models A \Rightarrow B$ iff $\langle \mathcal{M}, s_i\rangle \not\models A$ or $\langle \mathcal{M}, s_i\rangle \models B$.

s6 $\langle \mathcal{M}, s_i\rangle \models \mathbf{A}B$ iff for all χ_{s_i}, $\langle \mathcal{M}, \chi_{s_i}\rangle \models B$.

s7 $\langle \mathcal{M}, s_i\rangle \models \mathbf{E}B$ iff there exists χ_{s_i} s.t. $\langle \mathcal{M}, \chi_{s_i}\rangle \models B$.

p1 $\langle \mathcal{M}, \chi_{s_i}\rangle \models A$ iff $\langle \mathcal{M}, s_i\rangle \models A$, for state formula A.

p2 $\langle \mathcal{M}, \chi_{s_i}\rangle \models \Box B$ iff for all $s_j \in \chi_{s_i}$ if $i \leq j$ then $\langle \mathcal{M}, Suf(\chi_{s_i}, s_j)\rangle \models B$.

p3 $\langle \mathcal{M}, \chi_{s_i}\rangle \models \Diamond B$ iff there exists $s_j \in \chi_{s_i}$ s.t. $i \leq j$ and $\langle \mathcal{M}, Suf(\chi_{s_i}, s_j)\rangle \models B$.

p4 $\langle \mathcal{M}, \chi_{s_i}\rangle \models \bigcirc B$ iff $\langle \mathcal{M}, Suf(\chi_{s_i}, s_{i+1})\rangle \models B$.

p5 $\langle \mathcal{M}, \chi_{s_i}\rangle \models A\,\mathcal{U}\,B$ iff there exists $s_j \in \chi_{s_i}$ s.t. $i \leq j$ and $\langle \mathcal{M}, Suf(\chi_{s_i}, s_j)\rangle \models B$ and for all $s_k \in \chi_{s_i}$ if $i \leq k < j$ then $\langle \mathcal{M}, Suf(\chi_{s_i}, s_k)\rangle \models A$.

p6 $\langle \mathcal{M}, \chi_{s_i}\rangle \models A\,\mathcal{W}\,B$ iff $\langle \mathcal{M}, \chi_{s_i}\rangle \models \Box A$ or $\langle \mathcal{M}, \chi_{s_i}\rangle \models A\,\mathcal{U}\,B$.

A well-formed formula, B, is *satisfiable* if, and only if, there exists a model $\mathcal{M}$ such that $\langle \mathcal{M}, s_0\rangle \models B$. A well-formed formula, B, is *valid* if, and only if, B is satisfied in every possible model, i.e. for each $\mathcal{M}$, $\langle \mathcal{M}, s_0\rangle \models B$.

Recall that well-formed CTL formulae are state formulae and due to the syntactic requirement (see above) when we evaluate a CTL formula, for example, $\mathbf{A}\bigcirc C$, we reduce the problem $\langle \mathcal{M}, s_i\rangle \models \mathbf{A}\bigcirc C$, following s6 above, to the evaluation of $\bigcirc C$ along each path χ_{s_i}. This, acording to p4, means $\langle \mathcal{M}, Suf(\chi_{s_i}, s_{i+1}\rangle \models C$. Since C must be a *state* formula, applying p1, we obtain from the latter $\langle \mathcal{M}, s_{i+1}\rangle \models C$. Now, if C contains another temporal operator, then this operator will again be preceded by a path operator indicating a specific path context. For example, if $C = \mathbf{E}\Box D$ then the problem $\langle \mathcal{M}, s_{i+1}\rangle \models \mathbf{E}\Box D$ will be reduced, according to s7, to evaluating $\Box D$ along some path $\pi_{s_{i+1}}$, etc.

[1] It is known that if linear-time temporal logic is interpreted over discrete linear models with finite past and infinite future then adding past-time operators does not give more expressiveness [11].

2.2. Closure properties of CTL models

When trees are considered as models for distributed systems, paths through a tree are viewed as computations. The natural requirements for such models would be **suffix** and **fusion closures**. The former means that every suffix of a path is itself a path. The latter requires that a system, following the prefix of a computation γ, at any point $s_j \in \gamma$, is able to follow any computation π_{s_j} originating from s_j. Finally, we might require that if a system follows a computation for an arbitrarily long time, then it can follow a computation forever. This corresponds to **limit closure** property, meaning that for any fullpath χ_{s_0} and any paths $\pi_{s_j}, \varphi_{s_k} \ldots$ $(0 < j < k < l)$ such that χ_{s_0} has the prefix $[s_0, s_j]$, π_{s_j} has the prefix $[s_j, s_k]$, φ_{s_k} has the prefix $[s_k, s_l]$ etc, the following holds (see Figure 1):

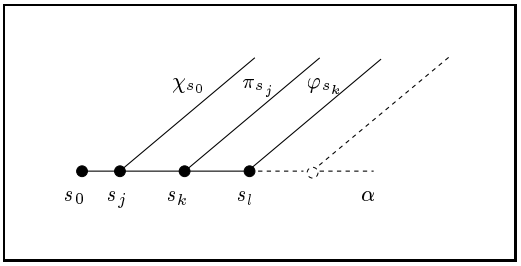

Figure 1. Limit closure

there exists an infinite path α that is a limit of the prefixes $[s_0, s_j]$, $[s_j, s_k]$, $[s_k, s_l]$ ....

In the following we assume that tree-like models of CTL are suffix, fusion and limit closed[2].

3. The temporal resolution method for CTL

Here we review the temporal resolution method for CTL [3, 2] whose main components are translation into the clausal normal form and application of resolution rules.

3.1. A normal form for CTL

The basic idea behind the normal form for CTL called Separated Normal Form ($\mathrm{SNF}_{\mathrm{CTL}}$) is to identify the core operators and generate formulae relevant to either the first state in a model, or to all subsequent states in a model. The transformation procedure uses fixpoint unwinding and subformula renaming in order to reduce an arbitrary formula to $\mathrm{SNF}_{\mathrm{CTL}}$. To preserve a specific path context *indices* are used.

[2] A variety of difficult problems concerning branching-time logics are due to the limit closure property, resulting, for example, in the case of CTL*, in the absence of a complete axiomatization.

The language for indices is based on the set of terms

$$\{\mathsf{IND}\} = \{\mathsf{f},\ \mathsf{g},\ \mathsf{h},\ \mathsf{LC(f)},\ \mathsf{LC(g)},\ \mathsf{LC(h)}\ \ldots\}$$

Thus, $\mathbf{E}A_{\langle \mathsf{f} \rangle}$ means that A holds on some path labelled as $\langle \mathsf{f} \rangle$. Indices of the type $\langle \mathsf{LC(ind)} \rangle$ aim to capture limit closure property. Once the translation to $\mathrm{SNF}_{\mathrm{CTL}}$ has been carried out, all $\mathrm{SNF}_{\mathrm{CTL}}$ clauses containing a '**E**' quantifier are labelled with some index (for more details see [2]).

Further, an additional operator, $\mathbf{start}$, which effectively identifies the initial state, is introduced:

$$\langle \mathcal{M}, s_i \rangle \models \mathbf{start} \quad \text{iff} \quad i = 0$$

Definition 1 (Separated Normal Form for CTL) *Given a CTL formula F, the separated normal form for F,* $\mathrm{SNF}_{\mathrm{CTL}}$*(F), is a set of* clauses *of the form*

$$\mathbf{A}\Box \bigwedge_i (P_i \Rightarrow F_i)$$

where each of the "$P_i \Rightarrow F_i$" is further restricted as in Figure 2, where each α_i, β_j or γ is a literal, $\mathbf{true}$ *or* $\mathbf{false}$, *and $\langle \mathsf{ind} \in \mathsf{IND} \rangle$ is some index.*

$$\begin{array}{lcll}
\mathbf{start} & \Rightarrow & \bigvee_{j=1}^{k} \beta_j & \text{an initial clause} \\
\bigwedge_{i=1}^{l} \alpha_i & \Rightarrow & \mathbf{A}\bigcirc \left[\bigvee_{j=1}^{k} \beta_j \right] & \text{an } \mathbf{A} \text{ step clause} \\
\bigwedge_{i=1}^{l} \alpha_i & \Rightarrow & \mathbf{E}\bigcirc \left[\bigvee_{j=1}^{k} \beta_j \right]_{\langle \mathsf{ind} \rangle} & \text{a } \mathbf{E} \text{ step clause} \\
\bigwedge_{i=1}^{l} \alpha_i & \Rightarrow & \mathbf{A}\Diamond\gamma & \text{an } \mathbf{A} \text{ sometime clause} \\
\bigwedge_{i=1}^{l} \alpha_i & \Rightarrow & \mathbf{E}\Diamond\gamma_{\langle \mathsf{LC(ind)} \rangle} & \text{a } \mathbf{E} \text{ sometime clause}
\end{array}$$

Figure 2. Form of $\mathrm{SNF}_{\mathrm{CTL}}$ Clauses

The natural intuition here is that the initial clauses provide starting conditions while step and sometime clauses constrain the future behaviour. For example, a step clause $\mathbf{A}\Box(x \Rightarrow \mathbf{A}\bigcirc p)$ means

"for any fullpath χ and any state $s_i \in \chi$ $(0 \leq i)$, if x is satisfied at a state s_i then p must be satisfied at the moment, next to s_i, along each path which starts from s_i".

Similarly, interpreting $\mathbf{A}\Box(x \Rightarrow \mathbf{E}\bigcirc p_{\langle \mathsf{ind} \rangle})$, we use the information that '**E**' is associated with the index $\langle \mathsf{ind} \rangle$:

"for any fullpath χ and any state $s_i \in \chi$ $(0 \leq i)$, if x is satisfied at a state s_i then p must be satisfied at the moment, next to s_i, along some path associated with $\langle \mathsf{ind} \rangle$ which departs from s_i".

Finally, $\mathbf{A}\Box(x \Rightarrow \mathbf{E}\diamondsuit p_{\langle \mathsf{LC}(\mathsf{ind}) \rangle})$ means

"for any fullpath χ and any state $s_i \in \chi$ $(0 \leq i)$ if x is satisfied at a state s_i then p must be satisfied at some state, say s_j $(i \leq i)$, along some path α_{s_i} associated with the limit closure of $\langle \mathsf{ind} \rangle$ which departs from s_i".

All the other operators are subsumed within this representation. For example, the '$\mathbf{A}\Box$' operator is represented by a (possibly infinite) sequence of '$\mathbf{A}\bigcirc$' operations (see [3]). For convenience we will omit the outer '$\mathbf{A}\Box$' connective that surrounds the conjunction of clauses and drop the conjunction considering the set of clauses.

3.2. Resolution rules for CTL

Once a set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses, R, has been obtained, a resolution method is applied to R. Here we repeatedly apply 'step' and 'temporal' resolution rules, together with various simplification steps.

3.2.1 Step resolution. Step (classical) resolution can be used between formulae that refer to the initial moment of time or *same* next moment on some or all paths. The corresponding step resolution rules are given below

SRES 1

$$\frac{\begin{array}{rcl} \mathbf{start} & \Rightarrow & C \vee l \\ \mathbf{start} & \Rightarrow & D \vee \neg l \end{array}}{\begin{array}{rcl} \mathbf{start} & \Rightarrow & C \vee D \end{array}}$$

SRES 2

$$\frac{\begin{array}{rcl} P & \Rightarrow & \mathbf{A}\bigcirc(C \vee l) \\ Q & \Rightarrow & \mathbf{A}\bigcirc(D \vee \neg l) \end{array}}{\begin{array}{rcl} (P \wedge Q) & \Rightarrow & \mathbf{A}\bigcirc(C \vee D) \end{array}}$$

SRES 3

$$\frac{\begin{array}{rcl} P & \Rightarrow & \mathbf{A}\bigcirc(C \vee l) \\ Q & \Rightarrow & \mathbf{E}\bigcirc(D \vee \neg l)_{\langle \mathsf{ind} \rangle} \end{array}}{\begin{array}{rcl} (P \wedge Q) & \Rightarrow & \mathbf{E}\bigcirc(C \vee D)_{\langle \mathsf{ind} \rangle} \end{array}}$$

SRES 4

$$\frac{\begin{array}{rcl} P & \Rightarrow & \mathbf{E}\bigcirc(C \vee l)_{\langle \mathsf{ind} \rangle} \\ Q & \Rightarrow & \mathbf{E}\bigcirc(D \vee \neg l)_{\langle \mathsf{ind} \rangle} \end{array}}{\begin{array}{rcl} (P \wedge Q) & \Rightarrow & \mathbf{E}\bigcirc(C \vee D)_{\langle \mathsf{ind} \rangle} \end{array}}$$

where l is a literal and $\langle \mathsf{ind} \rangle$ is an index.

When an empty clause is generated on the right hand side of the conclusion of the resolution rule, we introduce a constant $\mathbf{false}$ to indicate this situation and, for example, the conclusion of SRES 2 rule, when resolving $P \Rightarrow \mathbf{A}\bigcirc q$ and $P \Rightarrow \mathbf{A}\bigcirc \neg q$, will be $P \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$.

Once a contradiction within a state(s) is found, as for example, $P \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$, then we simplify it applying the following rule:

$$\frac{\begin{array}{rcl} P & \Rightarrow & \mathbf{P}\bigcirc\mathbf{false} \end{array}}{\begin{array}{rcl} P & \Rightarrow & \mathbf{false} \end{array}}$$

where **P** is either of path quantifiers. The conculsion, $P \Rightarrow \mathbf{false}$, in turn, requires that P must never be satisfied in *any* moment in time. This is reflected in generating extra constraints by applying the following rule:

$$\frac{\begin{array}{rcl} P & \Rightarrow & \mathbf{false} \end{array}}{\begin{array}{rcl} \mathbf{start} & \Rightarrow & \neg P \\ \mathbf{true} & \Rightarrow & \mathbf{A}\bigcirc\neg P \end{array}}$$

The step resolution process terminates when either no new resolvents are derived, or $\mathbf{start} \Rightarrow \mathbf{false}$ is derived.

Once step resolution has been applied, the temporal resolution rule can be invoked. The basic idea here is to resolve a set of formulae containing a loop in l, i.e. a situation when l occurs at all future moments along some (**E**-loop in l) or every path (an **A**-loop in l) from a particular point in a CTL model, with the formula containing a $\diamondsuit\neg l$, provided that both refer to the *same* path. Thus, identification of loops within given set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses, similar to PLTL, is the crucial part of the temporal resolution method in CTL. As in PLTL, some loops might be given directly as a set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses; in other cases loops might be more difficult to detect. One of the benefits of the normal form is that it allows us to identify 'hidden' loops within some set of clauses.

3.2.2. Loops in CTL. Loops in CTL are defined on sets of merged clauses, which are, in turn, generated from step clauses.

Merging

$$\frac{\begin{array}{rcl} P & \Rightarrow & \mathbf{A}\bigcirc C \\ Q & \Rightarrow & \mathbf{A}\bigcirc D \end{array}}{\begin{array}{rcl} (P \wedge Q) & \Rightarrow & \mathbf{A}\bigcirc(C \wedge D) \end{array}}$$

$$\frac{\begin{array}{rcl} P & \Rightarrow & \mathbf{A}\bigcirc C \\ Q & \Rightarrow & \mathbf{E}\bigcirc D_{\langle \mathsf{ind} \rangle} \end{array}}{\begin{array}{rcl} (P \wedge Q) & \Rightarrow & \mathbf{E}\bigcirc(C \wedge D)_{\langle \mathsf{ind} \rangle} \end{array}}$$

$$\frac{\begin{array}{rcl} P & \Rightarrow & \mathbf{E}\bigcirc C_{\langle \mathsf{ind} \rangle} \\ Q & \Rightarrow & \mathbf{E}\bigcirc D_{\langle \mathsf{ind} \rangle} \end{array}}{\begin{array}{rcl} (P \wedge Q) & \Rightarrow & \mathbf{E}\bigcirc(C \wedge D)_{\langle \mathsf{ind} \rangle} \end{array}}$$

Note that, similar to SRES 4, we allow merging of two **E** step clauses if both existential path quantifiers refer to the same path.

Definition 2 (Loop in CTL) *A loop in l is a set of merged clauses (possibly labelled) of the form*

$$\begin{array}{lcl} B_0 & \Rightarrow & \mathbf{P}_0 \bigcirc C_{0_{\langle \mathsf{ind}_0 \rangle}} \\ \ldots & & \ldots \\ B_n & \Rightarrow & \mathbf{P}_n \bigcirc C_{n_{\langle \mathsf{ind}_n \rangle}} \end{array}$$

where $\models C_i \Rightarrow l$ *and* $\models C_i \Rightarrow \bigvee_{j=0}^{n} B_j$, *for all* $0 \leq i \leq n$.

We will abbreviate such loop by $(B_0 \vee \ldots \vee B_n) \Rightarrow \mathbf{P}\bigcirc\mathbf{P}\Box l_{\langle \mathsf{ind} \rangle}$, where

- if for all i, $(0 \leq i \leq n)$, $\mathbf{P}_i$ is the '**A**' path quantifier then $\mathbf{P} = \mathbf{A}$, $\langle \mathsf{ind} \rangle$ is empty, and we have an **A**-loop in l,
- if for all i $(0 \leq i \leq n)$, $\mathbf{P}_i$ only involves one '**E**' quantifier labelled by $\langle \mathsf{ind}_i \rangle$ or every $\mathbf{P}_i$ which involves the '**E**' quantifier has the same label $\langle \mathsf{ind}_i \rangle$ then $\langle \mathsf{ind} \rangle = \langle \mathsf{LC}(\mathsf{ind}_i) \rangle$ and we have a **E**-loop in l on the path $\langle \mathsf{LC}(\mathsf{ind}_i) \rangle$, otherwise
- we have indicated a **E**-loop in l on the path $\langle \mathsf{LC}(\mathsf{ind}) \rangle$, where $\langle \mathsf{ind} \rangle$ is a new index.

For this set of merged clauses, each right hand side implies one or more left hand sides from the side condition on loops. Each right hand side implies l. Hence, once one of the left hand sides is satisfied, a literal l holds at all future moments on some or all paths (dependent on the type of the path quantifier).

As a simple example consider the following set of clauses (where the first clause on its own gives us a loop in x).

$$\mathbf{true} \Rightarrow \mathbf{A}\bigcirc x, \quad x \wedge r \Rightarrow \mathbf{E}\bigcirc r_{\langle \mathsf{ind} \rangle} \tag{1}$$

By merging both clauses, we obtain

$$x \wedge r \Rightarrow \mathbf{E}\bigcirc (x \wedge r)_{\langle \mathsf{ind} \rangle} \tag{2}$$

which gives us additionally a **E**-loop, in r, which is linked to the limit closure property of CTL. Consider a model $\mathcal{M}$ given in Figure 3, where we arbitrarily chose a fullpath χ and let $s_i \in \chi$ be the first moment along χ which satisfies $x \wedge r$. Therefore, there is a path π_{s_i} associated with $\langle \mathsf{ind} \rangle$ such that s_j, the successor of s_i on this path, satisfies $x \wedge r$. Formula (2) means "for any fullpath φ and any state $s_k \in \varphi\,(0 \leq k)$, if $x \wedge r$ is satisfied at s_k then $\mathbf{E}\bigcirc(x \wedge r)_{\langle \mathsf{ind} \rangle}$ is satisfied at s_k". Due to the fusion closure property, there is a fullpath $[s_0, s_i] \circ \pi_{s_i}$ (a concatenation of $[s_0, s_i]$ and π_{s_i}). Thus, setting $\varphi = [s_0, s_i] \circ \pi_{s_i}$ and $k = j$, we conclude that $\langle \mathcal{M}, s_j \rangle \models \mathbf{E}\bigcirc(x \wedge r)_{\langle \mathsf{ind} \rangle}$. Therefore, there is a path φ_{s_j} associated with $\langle \mathsf{ind} \rangle$ such that there is a state, next to s_j, say s_m, on this path which satisfies $x \wedge r$, etc. Hence, according to the limit closure property, the sequence

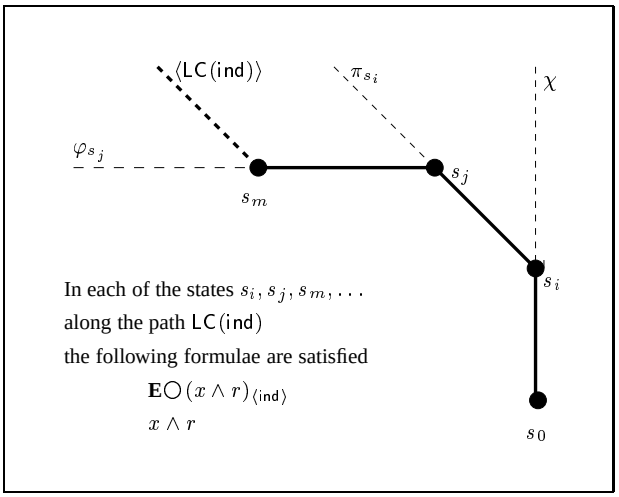

Figure 3. Effects of limit closure in CTL: E-loop

$s_i, s_j, s_m \ldots$ is a path, and each state along this path satisfies $x \wedge r$, which gives us the desired **E**-loop in r.

3.2.3. Temporal resolution rules. Now, using the expressions $P \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l$ and $P \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$ as abbreviations for sets of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses which together represent these formulae, temporal resolution rules for CTL are defined as follows.

TRES 1

$$\frac{\begin{array}{lcl} P & \Rightarrow & \mathbf{A}\bigcirc\mathbf{A}\Box l \\ Q & \Rightarrow & \mathbf{A}\Diamond\neg l \end{array}}{Q \Rightarrow \mathbf{A}(\neg P\, \mathcal{W}\, \neg l)}$$

TRES 2

$$\frac{\begin{array}{lcl} P & \Rightarrow & \mathbf{A}\bigcirc\mathbf{A}\Box l \\ Q & \Rightarrow & \mathbf{E}\Diamond\neg l_{\langle \mathsf{LC}(\mathsf{ind}) \rangle} \end{array}}{Q \Rightarrow \mathbf{E}(\neg P\, \mathcal{W}\, \neg l)_{\langle \mathsf{LC}(\mathsf{ind}) \rangle}}$$

TRES 3

$$\frac{\begin{array}{lcl} P & \Rightarrow & \mathbf{E}\bigcirc\mathbf{E}\Box l_{\langle \mathsf{LC}(\mathsf{ind}) \rangle} \\ Q & \Rightarrow & \mathbf{A}\Diamond\neg l \end{array}}{Q \Rightarrow \mathbf{A}(\neg P\, \mathcal{W}\, \neg l)}$$

TRES 4

$$\frac{\begin{array}{lcl} P & \Rightarrow & \mathbf{E}\bigcirc\mathbf{E}\Box l_{\langle \mathsf{LC}(\mathsf{ind}) \rangle} \\ Q & \Rightarrow & \mathbf{E}\Diamond\neg l_{\langle \mathsf{LC}(\mathsf{ind}) \rangle} \end{array}}{Q \Rightarrow \mathbf{E}(\neg P\, \mathcal{W}\, \neg l)_{\langle \mathsf{LC}(\mathsf{ind}) \rangle}}$$

In each case the resolvent ensures that once Q has been satisfied, the conditions, P, for triggering a $\Box$-formula are

not allowed to occur, i.e., P must be false, until the eventuality ($\neg l$) has been satisfied. Although it might be surprising that resolving an $\mathbf{A}\Diamond\neg l$ with a **E**-loop in l results in an **A**-formula, if the premises of temporal resolution in this case are satisfiable, the satisfiability of the conclusion of TRES 3 is guaranteed by the limit closure property (the corresponding proof is given in [2]).

This system of resolution rules is a complete deductive method for the logic CTL ([2, 3]).

4. Applying CTL Temporal Resolution

The clausal resolution approach to linear-time logic has been shown to be particularly amenable to efficient implementation [5, 6]. Here we concentrate on one of these algorithms, the Breadth-First Search approach [6], and modify it for use in our branching time setting.

4.1. Overview of the Loop Detection Method in CTL

While in PLTL we have only one temporal resolution rule, in the branching-time framework, a variety of such rules are defined. Depending on the type of a path quantifier in the 'sometime' $\mathrm{SNF}_{\mathrm{CTL}}$ clause, we look for different types of loops. In particular, given $\mathbf{A}\Diamond\neg l$, we search for a set of clauses that together imply either $\mathbf{A}\bigcirc\mathbf{A}\Box l$ or $\mathbf{E}\bigcirc\mathbf{E}\Box l$ (labelled by any $\langle\mathsf{ind}\rangle$), which can be used to apply TRES 1 and TRES 3. When we are resolving with a 'sometime' formula containing $\mathbf{E}\Diamond\neg l$ labelled by $\langle\mathsf{LC(ind)}\rangle$, then we search for a set of clauses that together imply either $\mathbf{A}\bigcirc\mathbf{A}\Box l$ or $\mathbf{E}\bigcirc\mathbf{E}\Box l_{\langle\mathsf{ind}\rangle}$ to apply TRES 2 or TRES 4. In the latter case use of indices is crucial.

The Breadth-First Search Algorithm constructs a sequence of nodes that are labelled with formulae in Disjunctive Normal Form. This represents the left hand sides of clauses used to expand the previous node which have been disjointed and simplified. Clauses are selected for use in the algorithm if they generate the required literal at the next moment in time and their right hand side implies the previous node. The former ensures the required literal holds and the latter gives the looping required so that the literal *always* holds. If we build a new node that is equivalent to the previous one, using this approach, then we have detected a loop. However, if we cannot create a new node then we terminate without having found a loop.

Not surprisingly, the detection of a **A**-loop is almost identical to that of Breadth-First Search in linear-time temporal logic. However, with the detection of an **E**-loop we must take care when combining clauses (see below).

4.2. Breadth-first A-loop search algorithm

Given a set, R, of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses we develop an algorithm to detect an **A**-loop in l by constructing a set of nodes $H_0, H_1, \ldots, H_n$ labelled by formulae $L_0, \ldots, L_n$, where each L_i $(0 \leq i \leq n)$ is in DNF and the label, L_n, of the terminating node H_n satisfies the following condition: $L_n \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l$.

Thus, to detect an **A**-loop in l follow the steps below.
(1) Search for all $\mathrm{SNF}_{\mathrm{CTL}}$ clauses of the form $X_k \Rightarrow \mathbf{A}\bigcirc l$, for $k = 0$ to b, disjoin the left hand sides and make the label L_0, of the *top node* H_0 equivalent to this, i.e.[3]

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

Simplify H_0. If $\vdash H_0 \equiv \mathbf{true}$ we have found a loop.

(2) Given a node H_i, build node H_{i+1} for $i = 0, 1, \ldots$ by looking for clauses or combinations of clauses of the form $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$, for $j = 0$ to m where $\vdash B_j \Rightarrow H_i$. Disjoin the left hand sides so that

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

and simplify as previously.

(3) Repeat (2) until one of the conditions (a)-(c) holds:

(a) $\vdash H_i \equiv \mathbf{true}$. We have found a **A**-loop and return the label of the terminating node, $\mathbf{true}$.

(b) $\vdash H_i \equiv H_{i+1}$. We have found a **A**-loop and return the label of the terminating node, DNF formula H_i.

(c) H_{i+1} is empty. Terminate - no loop has been found.

4.3. Breadth-first E-loop search algorithm

To detect a **E**-loop in l do the following.
(1) Search for all the clauses of the form $X_k \Rightarrow \mathbf{A}\bigcirc l$, or $X_k \Rightarrow \mathbf{E}\bigcirc l_{\langle\mathsf{ind_n}\rangle}$, for $k = 0$ to b, disjoin the left hand sides, make the *top node* H_0 equivalent to this and label H_0 with Ind, i.e.

$$H_0 = \left[\bigvee_{k=0}^{b} X_k\right]\mathsf{Ind}$$

where Ind is a set of all indices $\langle\mathsf{ind_n}\rangle$ occurring within $X_k \Rightarrow \mathbf{E}\bigcirc l_{\langle\mathsf{ind_n}\rangle}$. Simplify H_0. If $\vdash H_0 \equiv \mathbf{true}$ we terminate having found a loop.

[3] To simplify the presentation below, since a label L_i of a node H_i uniquely identifies this node, instead of saying 'a label $L_i = (X_1 \vee \ldots \vee X_r)$ of a node H_i' we will use the expression $H_i = (X_1 \vee \ldots \vee X_r)$.

(2) Given a node H_i $\langle \mathsf{Ind}' \rangle$, build a node H_{i+1} $\langle \mathsf{Ind}'' \rangle$ for $i = 0, 1, \ldots$ by looking for combinations of clauses of the form $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}\bigcirc(B_j \wedge l)_{\langle \mathsf{ind_s} \rangle}$ for $j = 0$ to m where $\vdash B_j \Rightarrow H_i$. Disjoin the left hand sides so that

$$H_{i+1} = \left[\bigvee_{j=0}^{m} A_j \right] \mathsf{Ind}''$$

where Ind" is a set of all indices $\langle \mathsf{ind_s} \rangle$ occurring within $A_j \Rightarrow \mathbf{E}\bigcirc(B_j \wedge l)_{\langle \mathsf{ind_s} \rangle}$. Simplify H_{i+1} as previously.

(3) Repeat (2) until one of the conditions (a)-(c) holds:

(a) $\vdash H_i \equiv \mathbf{true}$. We terminate having found a **E**-loop on path $\langle \mathsf{ind} \rangle$ and return the label of the terminating node, $\mathbf{true}$, provided that the following condition (†) is satisfied

- if $\langle \mathsf{ind} \rangle$ is the only element of Ind then we have found a **E**-loop in l on the path $\langle \mathsf{LC(ind)} \rangle$, else
- we have found a **E**-loop in l on the path $\langle \mathsf{LC(ind}') \rangle$, where $\langle \mathsf{ind}' \rangle$ is a new index.

(b) $\vdash H_i \equiv H_{i+1}$. We terminate having found a **E**-loop and return the label of the terminating node, DNF formula H_i provided that condition † above is satisfied;

(c) H_{i+1} is empty. Terminate - no loop has been found.

In a particular case of the **E**-loop detection algorithm, given a sometime clause labelled by $\langle \mathsf{LC(ind)} \rangle$ we search for a loop on the path $\langle \mathsf{LC(ind)} \rangle$. Thus, we only apply merging to those step clauses containing '**E**' quantifier which are labelled by $\langle \mathsf{ind} \rangle$. This will guarantee that if the algorithm terminates by finding a **E**-loop in l, this loop will occur on the desired path $\langle \mathsf{LC(ind)} \rangle$.

4.4. Examples

Example 1. Consider the following set of $\mathrm{SNF_{CTL}}$ clauses in which we are looking for a loop:

1.	a	$\Rightarrow$	$\mathbf{E}\bigcirc l_{\langle \mathsf{ind_1} \rangle}$	5.	e	$\Rightarrow$	$\mathbf{E}\bigcirc e_{\langle \mathsf{ind_2} \rangle}$
2.	b	$\Rightarrow$	$\mathbf{A}\bigcirc l$	6.	$(a \wedge e)$	$\Rightarrow$	$\mathbf{A}\bigcirc a$
3.	c	$\Rightarrow$	$\mathbf{A}\bigcirc l$	7.	b	$\Rightarrow$	$\mathbf{A}\bigcirc b$
4.	d	$\Rightarrow$	$\mathbf{A}\bigcirc l$	8.	c	$\Rightarrow$	$\mathbf{E}\bigcirc b_{\langle \mathsf{ind_3} \rangle}$

Noting that here **A**-loop searching algorithm detects a loop $b \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\,\Box l$, we will construct a **E**-loop.

1. The clauses 1–4 have either $\mathbf{A}\bigcirc l$ or $\mathbf{E}\bigcirc l$ on their right hand side. We disjoin their left hand sides and simplify to give the top node

$$H_0 = a \vee b \vee c \vee d_{\{\langle \mathsf{ind_1} \rangle\}}.$$

2. To build the next node, H_1, we see that the merged clauses 6+1, 7+2 and 8+3 satisfy the expansion criteria in step (2) of the **E**-loop search algorithm (i.e. all the pairs of clauses can be merged and the right hand side contains l as a conjunct and also implies H_0). So we disjoin the literals on the left hand sides of the merged clauses 6+1, 7+2 and 8+3 to obtain node

$$H_1 = (a \wedge e) \vee b \vee c_{\{\langle \mathsf{ind_1} \rangle, \langle \mathsf{ind_3} \rangle\}}.$$

Note that merged clause 7+1 would be removed via simplification.

3. Clauses 7+2 and 8+3 still satisfy the expansion criteria so we can add $b \vee c$ to our new node. However if we merge clauses 5 and 6 to give

$$(a \wedge e) \Rightarrow \mathbf{E}\bigcirc(a \wedge e)_{\langle \mathsf{ind_2} \rangle}$$

we cannot now combine it with clause 1 as it has a different path index. We can, however, merge it with either clause 2, 3 or 4 to give

$$\begin{array}{rcl} (a \wedge e \wedge b) & \Rightarrow & \mathbf{E}\bigcirc(a \wedge e \wedge l)_{\langle \mathsf{ind_2} \rangle} \\ (a \wedge e \wedge c) & \Rightarrow & \mathbf{E}\bigcirc(a \wedge e \wedge l)_{\langle \mathsf{ind_2} \rangle} \\ (a \wedge e \wedge d) & \Rightarrow & \mathbf{E}\bigcirc(a \wedge e \wedge l)_{\langle \mathsf{ind_2} \rangle} \end{array}$$

The first two left hand sides will be removed via simplification to leave node H_2 as

$$H_2 = (a \wedge e \wedge d) \vee b \vee c_{\{\langle \mathsf{ind_2} \rangle, \langle \mathsf{ind_3} \rangle\}}.$$

4. Now only clauses 7+2 and 8+3 satisfy the expansion criteria so the new node is

$$H_3 = b \vee c_{\{\langle \mathsf{ind_3} \rangle\}}.$$

5. The same thing happens when we construct the next node so we obtain

$$H_4 = b \vee c_{\{\langle \mathsf{ind_3} \rangle\}}.$$

and terminate with the loop $(b \vee c) \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\,\Box l_{\{\langle \mathsf{ind_3} \rangle\}}$.

Example 2. Consider the following set of $\mathrm{SNF_{CTL}}$ clauses

1.	p	$\Rightarrow$	$\mathbf{E}\bigcirc l_{\langle \mathsf{ind_1} \rangle}$	3.	p	$\Rightarrow$	$\mathbf{A}\bigcirc q$
2.	q	$\Rightarrow$	$\mathbf{A}\bigcirc l$	4.	q	$\Rightarrow$	$\mathbf{E}\bigcirc p_{\langle \mathsf{ind_2} \rangle}$

It is not immediately obvious that this set of $\mathrm{SNF_{CTL}}$ clauses contains a **E**-loop in l, namely, $p \vee q \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\,\Box l$.

1. The clauses 1 and 2 have either $\mathbf{A}\bigcirc l$ or $\mathbf{E}\bigcirc l$ on their right hand side, hence, we disjoin their left hand sides and simplify to give the top node

$$H_0 = p \vee q_{\{\langle \mathsf{ind_1} \rangle\}}.$$

2. To build the next node, H_1, we derive merged clauses 1+3 and 2+4, to obtain

$$\begin{array}{rcl} p & \Rightarrow & \mathbf{E}\bigcirc(q \wedge l)_{\langle \mathsf{ind_1} \rangle} \\ q & \Rightarrow & \mathbf{E}\bigcirc(p \wedge l)_{\langle \mathsf{ind_2} \rangle}. \end{array}$$

These satisfy the expansion criteria, so we add $p \vee q$ to the new node. Note that while we can not merge clauses 1 and 4 as they are labelled by different indices, we can, additionally to the merging 1+3 and 2+4, obtain a merged clause 2+3. However, its left hand side will be removed via simplification to leave node H_1 as

$$H_1 = (p \vee q)_{\{\langle \mathsf{ind}_1 \rangle, \langle \mathsf{ind}_2 \rangle\}}.$$

3. Now, as $H_1 = H_0$ we terminate the searching with the loop $(p \vee q) \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l_{\langle \mathsf{LC}(\mathsf{ind}_3) \rangle}$, where $\langle \mathsf{ind}_3 \rangle$ is a new index.

The model, $\mathcal{M}$, (Figure 4), which satisfies the set of clauses given in Example 2, in particular, represents this loop. Pick a fullpath χ and a state $s_i \in \chi$ that is the first state satisfying $p \vee q$.

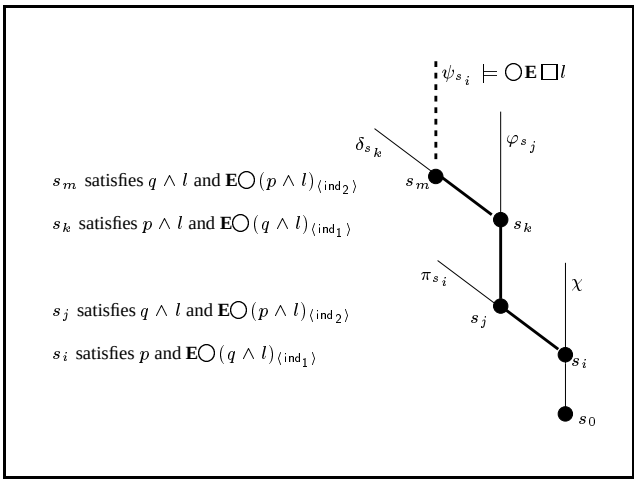

Figure 4. E-loop in a combination of step clauses with different labels

Suppose that $\langle \mathcal{M}, s_i \rangle \models p$. Thus, according to merged clause 1+3, $\langle \mathcal{M}, s_i \rangle \models \mathbf{E}\bigcirc(q \wedge l)_{\langle \mathsf{ind}_1 \rangle}$, i.e.there must be a path π_{s_i} associated with $\langle \mathsf{ind}_1 \rangle$, such that $s_j \in \pi_{s_i}$, the successor of s_i along π_{s_i}, satisfies $q \wedge l$. Hence, $\langle \mathcal{M}, s_j \rangle \models q$ and therefore, according to merged clause 2+4, $\langle \mathcal{M}, s_j \rangle \models \mathbf{E}\bigcirc(p \wedge l)_{\langle \mathsf{ind}_2 \rangle}$. Therefore, we must have in the model a path φ_{s_j} (possibly different from $Suf(\pi_{s_i}, s_j)$), associated with $\langle \mathsf{ind}_2 \rangle$ such that there is a state $s_k \in \varphi_{s_j}$, the successor of s_j, which satisfies $p \wedge l$. Now, considering s_k, we apply the same reasoning as in the case of s_i, etc. Thus, there is a path ψ_{s_i}, the limit closure of the prefixes $[s_i, s_j], [s_j, s_k], [s_k, s_m], \ldots$ such that $\langle \mathcal{M}, \psi_{s_i} \rangle \models \bigcirc\mathbf{E}\Box l$. Hence, $\langle \mathcal{M}, s_i \rangle \models \mathbf{E}\bigcirc\mathbf{E}\Box l$ as desired.

5. Correctness of the loop searching for CTL

Here we outline the soundness and completeness of the loop detection algorithms. The following lemma is useful.

Lemma 1 *Given a series of nodes H_i, for $i \geq 0$, output by a Breadth-First* **A***-Search for a loop in l then*

$$\models \mathbf{A}\Box(H_{i+1} \Rightarrow \mathbf{A}\bigcirc(H_i \wedge l))$$

PROOF:

For $i > 0$ in step (2) of the algorithm, given node H_i, we select clauses $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$, for $j = 0$ to m, where $\models B_j \Rightarrow H_i$ and the new node H_{i+1} is the disjunction of the A_js. Therefore, for each A_j, $\models A_j \Rightarrow \mathbf{A}\bigcirc H_i$ and $\models A_j \Rightarrow \mathbf{A}\bigcirc l$. Hence, we obtain

$$\models \mathbf{A}\Box(H_{i+1} \Rightarrow \mathbf{A}\bigcirc(H_i \wedge l)$$

as required. (END)

Lemma 2 *Given a series of nodes H_i, for $i \geq 0$, output by a Breadth-First* **E***-Search for a loop in l then*

$$\models \mathbf{A}\Box(H_{i+1} \Rightarrow \mathbf{E}\bigcirc(H_i \wedge l)).$$

PROOF: Similar to the **A**-loop (END)

Theorem 1 *[(Soundness)] Let R be a set of step clauses and l be the literal we are searching for a loop in. For any DNF formula X output by a Breadth-First* **A***-Search (respectively a Breadth-First* **E***-search), on R,*

$$\models X \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l \quad (\textit{respectively } X \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l.)$$

PROOF: We prove this for the **A**-Search and the **E**-Search is similar. From the termination conditions of the algorithm given in §4.2 and §4.3 we know that either $X \equiv \mathbf{true}$ from step (1) or step (3a), or $H_{i+1} = H_i$ from step (3b). For the former either the clause $\mathbf{true} \Rightarrow \mathbf{A}\bigcirc l$ is in the clause-set or there are clauses in the clause-set that together imply this. Again, due to the implicit $\mathbf{A}\Box$-operators surrounding $\mathrm{SNF}_{\mathrm{CTL}}$ clauses, the clause $\mathbf{true} \Rightarrow \mathbf{A}\bigcirc l$ holds in all states on all paths. Therefore, $\models \mathbf{true} \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l$ holds as required. Otherwise $H_{i+1} = H_i$ and from Lemma 1 we have $\models \mathbf{A}\Box(H_{i+1} \Rightarrow \mathbf{A}\bigcirc(H_i \wedge l))$. Thus, if we terminate with X then $\models \mathbf{A}\Box(X \Rightarrow \mathbf{A}\bigcirc(X \wedge l))$ and $\models X \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l$ as required. (END)

Theorem 2 *[(Completeness)] Let R be a set of step clauses and l be the literal we are searching for a loop in. For any set of clauses $R' \subseteq R$ that together imply an* **A***-loop, i.e. $\models X \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l$, (respectively a* **E***-loop, i.e. $\models X \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$) the Breadth-First* **A***-Search (respectively* **E***-Search) applied to R outputs a DNF formula X' and $X \Rightarrow X'$.*

PROOF: As we only use **A** step clauses for the **A**-loop, completeness for the **A**-loop can be shown as in the linear-time case [6]. Extract all **A** step clauses from R, delete the path quantifier and let the set of (linear-time) clauses be R'. Note also that the Breadth-First **A**-Search algorithm is identical to that of Breadth-First Search algorithm for linear-time temporal logic if the path quantifiers are deleted. To show completeness in the linear-time case a graph is constructed from the clauses in R', whose nodes are valuations of all the propositions in R'. Paths through the graph represent all possible models of R'. It is shown that if a loop exists in R', X, then this is represented in the graph of R' by a terminal subgraph where the required literal l holds at each node. More specifically X is satisfied by each node (a valuation) in the subgraph. The completeness proof shows that each step in the Breadth-First Search algorithm corresponds to an operation on the graph. If the Breadth-First Search algorithm detects a loop returning the DNF formula X' then this corresponds to the terminal subgraph representing the largest most comprehensive loop in R'. For more details see [5].

The proof for **E**-loops is similar but we must construct a graph as described above for each 'path' through the branching tree structure. (END)

6. Related work

Automata-theoretic methods for CTL extend those developed for PLTL [1, 8]. To test formulae of PLTL finite automata on infinite strings are used, the appropriate type of automata in the branching-time setting are finite automata on infinite trees, i.e. when automaton visits a state it reads an input tree rather then an input word. Given a CTL formula φ, a run of the automaton constructed for φ is considered successful if it meets certain requirements known as "acceptance conditions". This is also known as checking automaton for (non)emptiness. If the acceptance condition is satisfied then a state structure of a successful run is not empty and it gives a model for φ. Alternatively, a run is unsuccessful. If the automaton does not have a successful run then φ is not satisfiable. Note that although the structure of the normal form described in this work is close to alternating tree automata used in [1], there is no direct method of testing these automata for (non)emptiness. On the contrary, clausal resolution method is effectively applied to a set of clauses of normal form.

Tableau-based method for CTL are outlined in [8]. Using tableau methods, to show that a formula φ is valid, we negate φ and apply tableau algorithm. The algorithm systematically constructs a structure from which a model can be generated. If the structure is empty then no model can be constructed, the negated formula is unsatisfiable and the original formula is valid. The incremental method of the construction of the tableau [8] has been essentially used in showing completeness of the resolution method for CTL [2].

Proof methods for particular modal logics are given in several papers, for example [12, 14]. Ohlbach takes a resolution based approach removing modal operators and replacing them with *world path* arguments to predicate and function symbols, i.e. a modal diamond (possibility) is replaced by a meaning there is an accessible world a from here. Similarly here we annotate **E** rules with an index to denote which path we are referring to. Ohlbach requires sophisticated unification algorithms dependant upon the properties of the paths concerned. In the CTL resolution system matching indices is trivial.

Temporal logics are hard to reason about due to the interaction between the $\Box$ and $\bigcirc$ operators encoding, in case of linear-time temporal logic, a simple form of induction i.e.

$$(\varphi \wedge \Box(\varphi \Rightarrow \bigcirc\varphi)) \Rightarrow \Box\varphi.$$

The formulation of induction extended to branching-time temporal logic, which can be found in the axiomatization of CTL [8], is given by a set of formulae of a complex structure, for example,

$$\mathbf{A}\Box(r \Rightarrow (\neg q \wedge \mathbf{E}\bigcirc r)) \Rightarrow (r \Rightarrow \neg\mathbf{A}\Diamond q).$$

The complex resolution rule, and search method described in this paper is required to deal with the above induction principle. Note that induction in branching-time logic is additionally complicated by the limit closure property of the underlying tree models and in case of the full branching-time logic, CTL*, represents a main difficulty in axiomatizing the latter.

One of the benefits of the clausal resolution technique is the possibility of invoking a variety of well-developed methods and refinements used in the framework of classical logic. For example an initial investigation into the development of the set of support for classical logics [15] to that for linear-time temporal logics has been made in [7]. The refinement of this strategy is ongoing work and could potentially be adapted to the branching framework.

7 Conclusions

As we have already mentioned, most of the research on the proof methods for branching-time logics has been concentrated around the tableau or automata methods [8]. In this paper we have investigated the application of the clausal resolution method for CTL [2, 3]. The authors know of no other clausal resolution methods developed for branching-time logics. Searching for a loop is the crucial part of the

resolution technique. We have described breadth-first algorithms of identifying **A**-loops and **E**-loops and sketched the correctness argument. This, together with the algorithm for the clausal resolution for CTL [2], makes the resolution method developed practically suitable for implementation. However, ways to improve the application of the temporal resolution rule must be further investigated. Here, although we have formulated one method of searching for a loop, this is only the starting point. The work to be done in this direction concerns, in particular, development of the strategies of the preferred loops to find first, reduction of the search graphs and loop subsumption algorithms. Again, we expect here to incorporate the related results obtained for PLTL [5, 6]. Also, taking into account that in branching-time logics we deal with path quantifiers, it might be useful to investigate parallels between loop searching techniques and (rather simple in this case) unification algorithms used in the resolution technique for predicate logic.

Acknowledgements This work has been partially supported by funding from HEFCE, under a PhD studentship and EPSRC, under research grant GR/L87491. Both authors would like to thank Michael Fisher for his advice and comments on this work. We are also grateful to anonymous referees for their useful suggestions on improving the presentation.

References

[1] O. Bernholtz, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification. Proceedings of 6th International Workshop*, volume 818 of Lecture Notes in Computer Science. Springer-Verlag, 1994.

[2] A. Bolotov. *Clausal Resolution for Branching-Time Temporal Logic*. PhD thesis, The Manchester Metropolitan University, 1999, (submitted).

[3] A. Bolotov and M. Fisher. A clausal resolution method for ctl branching time temporal logic. *Journal of experimental and theoretical artificial intelligence*, (11):77–93, 1999.

[4] E. M. Clarke and E. A. Emerson. Using Branching Time Temporal Logic to Synthesise Synchronisation Skeletons. *Science of Computer Programming*, pages 241–266, 1982.

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

[6] C. Dixon. Temporal resolution using a breadth-first search algorithm. *Annals of Mathematics and Artificial Intelligence*, 22, 1998.

[7] 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.

[8] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics.*, pages 996–1072. Elsevier, 1990.

[9] E. A. Emerson. Automated reasoning about reactive systems.. In *Logics for Concurrency: Structures Versus Automata, Proc. of International Workshop*, volume 1043 of Lecture Notes in Computer Science. Springer-Verlag, 1996.

[10] M. Fisher. A Resolution Method for Temporal Logic. In *Proc. of the XII International Joint Conference on Artificial Intelligence (IJCAI)*, 1991.

[11] D. Gabbay, A. Phueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of 7th ACM Symposium on Principles of Programming Languages*, 1980.

[12] H.-J. Ohlbach. A Resolution Calculus for Modal Logics. *Lecture Notes in Computer Science*, 310:500–516, May 1988.

[13] A. Pnueli. The Temporal Logic of Programs. In *Proc. of the Eighteenth Symposium on the Foundations of Computer Science*, 1977.

[14] L. A. Wallen. Matrix Proof Methods for Modal Logics. In *Proc. IJCAI-87*, pages 917–923, Milan, Aug. 1987.

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