First-order resolution for CTL

Lan Zhang, Ullrich Hustadt and Clare Dixon

Department of Computer Science, University of Liverpool

Liverpool, L69 3BX, UK

{Lan.Zhang, U.Hustadt, CLDixon}@liverpool.ac.uk

Abstract

In this paper, we present a resolution-based calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for Computational Tree Logic (CTL) as well as details about an implementation of that calculus in the theorem prover CTL-RP. The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ requires a transformation of an arbitrary CTL formula to an equi-satisfiable clausal normal form formulated in an extension of CTL with indexed formulae. The calculus itself consists of a set of resolution rules which can be used as the basis for an EXPTIME decision procedure for the satisfiability problem of CTL.

We give a formal semantics for the clausal normal form, provide proofs for the soundness and completeness of the resolution rules, discuss the complexity of decision procedure based on $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, and present an approach to implementing the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ using first-order techniques.

1 Introduction

Temporal logic is considered an important tool in many different areas of Artificial Intelligence and Computer Science, including for example the specification and verification of concurrent and distributed systems [10, 17, 23]. Computational Tree Logic (CTL) [9] is a branching-time temporal logic whose underlying model of time is a choice of possibilities branching into future. CTL uses the path quantifiers $\mathbf{A}$ (for all paths) and $\mathbf{E}$ (for some path) and each temporal operator, $\Box$ (always in the future), $\bigcirc$ (at the next moment in time), $\mathcal{U}$ (until), etc, must be paired with a path quantifier. For example, $\mathbf{A}\Box\varphi$ means on all paths φ always holds. There are many important applications that can be represented and verified in CTL such as digital circuit verification [10], analysis of real time and concurrent systems [23], etc. A range of model checking algorithms [8, 10, 18] as well as proof methods have been developed for CTL. The main proof methods for CTL are based on automata [15], resolution [5], and tableaux [2, 14, 25]. In this paper, we define a resolution-based calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL and its implementation in the theorem prover CTL-RP. The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ consists of a number of so-called *step* and *eventuality* resolution rules that operate on a clausal normal form for CTL.

$\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is designed in order to allow the use of classical first-order resolution techniques to implement the rules of the calculus. We take advantage of this fact in the development of our prover CTL-RP which uses the first-order theorem prover SPASS as a basis for the implementation of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$.

The rest of the paper is organised as follows. We first present the syntax and semantics of CTL in Section 2 and then the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is presented in Section 3, followed by a proof of its soundness and completeness and a discussion of termination and complexity. Section 4 discusses our approach to the implementation of the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ using first-order resolution techniques. In Section 5, we provide a comparison between a related resolution calculus for CTL [5] and our calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, and demonstrate the improvements of our calculus. Finally, conclusions are drawn in Section 7.

2 Syntax and semantics of CTL

In this paper, we use the syntax and semantics of CTL introduced by Clarke and Emerson in [13].

The language of CTL is based on the following symbols.

1. A set of atomic propositions P_{PL}.
2. Propositional constants, **true** and **false**, and boolean operators, $\wedge, \vee, \Rightarrow, \Leftrightarrow$ and $\neg$ ($\wedge$ and $\vee$ are associative and commutative).
3. Temporal operators $\Box$ (always in the future), $\bigcirc$ (at the next moment in time), $\Diamond$ (eventually in the future), $\mathcal{U}$ (until), and $\mathcal{W}$ (unless).
4. Path quantifiers **A** (for all future paths) and **E** (for some future path).
5. CTL operators $\mathbf{A}\bigcirc, \mathbf{E}\bigcirc, \mathbf{A}\Box, \mathbf{E}\Box, \mathbf{A}\Diamond, \mathbf{E}\Diamond, \mathbf{A}\mathcal{U}, \mathbf{E}\mathcal{U}, \mathbf{A}\mathcal{W}$ and $\mathbf{E}\mathcal{W}$.

The set of *formulae of CTL* is inductively defined as follows.

1. **true** and **false** are CTL formulae;
2. all atomic propositions in P_{PL} are CTL formulae;
3. if φ and ψ are CTL formulae, then so are $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$, $(\varphi \Leftrightarrow \psi)$, $\mathbf{A}\Box\varphi$, $\mathbf{A}\Diamond\varphi$, $\mathbf{A}\bigcirc\varphi$, $\mathbf{A}(\varphi\mathcal{U}\psi)$, $\mathbf{A}(\varphi\mathcal{W}\psi)$, $\mathbf{E}\Box\varphi$, $\mathbf{E}\Diamond\varphi$, $\mathbf{E}\bigcirc\varphi$, $\mathbf{E}(\varphi\mathcal{U}\psi)$, and $\mathbf{E}(\varphi\mathcal{W}\psi)$.

Formulae of CTL over P_{PL} are interpreted in *model structures*, $M = \langle S, R, L\rangle$, where

- S is a set of *states*;
- R is a total binary *accessibility relation* over S; and
- $L : S \rightarrow 2^{\mathsf{P}_{\mathsf{PL}}}$ is an *interpretation function* mapping each state to the set of atomic propositions true at that state.

M may be viewed as a labelled, directed graph with node set S, arc set R and nodes labels given by L.

An infinite path χ_{s_i} is an infinite sequence of states $s_i, s_{i+1}, s_{i+2}, \ldots$ such that for every $j, j \geqslant i, (s_j, s_{j+1}) \in R$. Informally, χ_{s_i} is an infinite path starting from state s_i.

We inductively define a satisfaction relation $\models$ between a pair consisting of a model structure M and a state s_i in S, and a CTL formula as follows:

$\langle M, s_i \rangle \models \mathbf{true}$

$\langle M, s_i \rangle \not\models \mathbf{false}$

$\langle M, s_i \rangle \models p$ iff $p \in L(s_i)$ for an atomic proposition $p \in \mathsf{P_{PL}}$

$\langle M, s_i \rangle \models \neg\varphi$ iff $\langle M, s_i \rangle \not\models \varphi$

$\langle M, s_i \rangle \models (\varphi \vee \psi)$ iff $\langle M, s_i \rangle \models \varphi$ or $\langle M, s_i \rangle \models \psi$

$\langle M, s_i \rangle \models (\varphi \wedge \psi)$ iff $\langle M, s_i \rangle \models \varphi$ and $\langle M, s_i \rangle \models \psi$

$\langle M, s_i \rangle \models (\varphi \Rightarrow \psi)$ iff $\langle M, s_i \rangle \not\models \varphi$ or $\langle M, s_i \rangle \models \psi$

$\langle M, s_i \rangle \models (\varphi \Leftrightarrow \psi)$ iff $\langle M, s_i \rangle \models (\varphi \Rightarrow \psi)$ and $\langle M, s_i \rangle \models (\psi \Rightarrow \varphi)$

$\langle M, s_i \rangle \models \mathbf{A}\bigcirc\psi$ iff for every path χ_{s_i}, $\langle M, s_{i+1} \rangle \models \psi$

$\langle M, s_i \rangle \models \mathbf{E}\bigcirc\psi$ iff there exists a path χ_{s_i}, $\langle M, s_{i+1} \rangle \models \psi$

$\langle M, s_i \rangle \models \mathbf{A}(\varphi \mathcal{U} \psi)$ iff for every path χ_{s_i}, there exists a state $s_j \in \chi_{s_i} \langle M, s_j \rangle \models \psi$
and for all states $s_k \in \chi_{s_i}$, if $i \leqslant k < j$ then $\langle M, s_k \rangle \models \varphi$

$\langle M, s_i \rangle \models \mathbf{E}(\varphi \mathcal{U} \psi)$ iff there exists a path χ_{s_i} such that there exists a state $s_j \in \chi_{s_i} \langle M, s_j \rangle \models \psi$
and for all states $s_k \in \chi_{s_i}$, if $i \leqslant k < j$ then $\langle M, s_k \rangle \models \varphi$

In addition, we use the following equivalences to define the semantics of other temporal operators.

$$\begin{array}{rr}
\mathbf{A}\Diamond\varphi \equiv \mathbf{A}(\mathbf{true}\,\mathcal{U}\,\varphi) & \mathbf{E}\Diamond\varphi \equiv \mathbf{E}(\mathbf{true}\,\mathcal{U}\,\varphi) \\
\mathbf{A}\Box\varphi \equiv \neg\mathbf{E}\Diamond\neg\varphi & \mathbf{E}\Box\varphi \equiv \neg\mathbf{A}\Diamond\neg\varphi \\
\mathbf{A}(\varphi\,\mathcal{W}\,\psi) \equiv \neg\mathbf{E}(\neg\psi\,\mathcal{U}\,(\neg\varphi \wedge \neg\psi)) & \mathbf{E}(\varphi\,\mathcal{W}\,\psi) \equiv \neg\mathbf{A}(\neg\psi\,\mathcal{U}\,(\neg\varphi \wedge \neg\psi))
\end{array}$$

A CTL formula φ is *valid*, written $\models \varphi$, iff for every model structure M and every state $s \in M$, $M, s \models \varphi$. A CTL formula φ is satisfiable, iff for some model structure M and some state $s \in M$, $M, s \models \varphi$, and *unsatisfiable* otherwise. A model structure M such that φ is true at some state $s \in M$ is called a *model* of φ.

The satisfiability problem of CTL is known to be EXPTIME-complete [9, 13, 14].

In [13], Emerson provides a generalized semantics of CTL which defines the semantics relative to a more general structure $M' = \langle S, X, L \rangle$, where S and L are defined in Section 2, and $X \subseteq S^\omega$ is a family of infinite computation sequences (fullpaths) over S.

In general the set X can be arbitrary. However, Emerson also defines three properties of X which commonly hold if X is meant to model infinite computation sequences:

Suffix Closure If $s_0 s_1 s_2 \ldots \in X$, then the suffix $s_1 s_2 \ldots \in X$.

Fusion Closure If $x_1 s y_1,\ x_2 s y_2 \in X$, then $x_1 s y_2 \in X$ where x_1 and y_1 are prefixes of paths and y_1, $y_2 \in X$.

Limit Closure If $x_1 y_1,\ x_1 x_2 y_2,\ x_1 x_2 x_3 y_3, \ldots$ are all elements of X, where x_1, x_2, x_3 are prefixes of paths and $y_1, y_2, y_3 \in X$, then the infinite path $x_1 x_2 x_3 \ldots$, which is the limit of the prefixes $x_1,\ x_1 x_2,\ x_1 x_2 x_3, \ldots$ is also in X. In short, if it is possible to follow a path arbitrarily long, then it can be followed forever. (See an example in Figure 1)

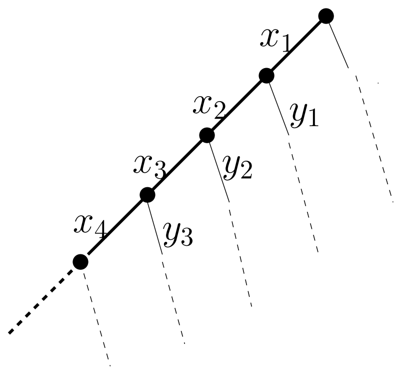


Figure 1: Limit Closure

As it turns out, if X is suffix, fusion, and limit closed, then the generalized semantics is identical to the semantics defined at the beginning of this section. Emerson defines a set X to be *R-generable* iff there exists a total binary relation R on S such that a sequence $s_0 s_1 s_2 \ldots \in X$ iff for every $i, (s_i, s_{i+1}) \in R$. Then X is R-generable iff it is limit closed, fusion closed and suffix closed.

3 $\mathsf{R}^{\succ,S}_{\text{CTL}}$: A refined resolution calculus for CTL

Our clausal resolution calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ for CTL is based on, but not identical to, the resolution calculus in [5, 7]. The calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ requires arbitrary CTL formulae to be transformed into a clausal normal form. The calculus itself consists of eight *step* resolution rules, two *eventuality* resolution rules and two *rewrite* rules. The calculus can be used as the basis for an EXPTIME decision procedure for the satisfiability problem of CTL.

3.1 Normal form

Our calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ operates on formulae in a clausal normal form called Separated Normal Form with Global Clauses for CTL, denoted by $\text{SNF}^{\text{g}}_{\text{CTL}}$. Separated Normal Form (SNF) was originally introduced for Propositional Linear Time Temporal Logic (PLTL) [16] and then extended to various other temporal logics including CTL [5].

The language of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is defined over an extension of CTL in which we label certain formulae with an index *ind* taken from a countably infinite index set Ind. To improve the readability of clauses, we introduce an operator precedence which allow us to reduce the number of parentheses required. We associate each operator with one of the following six precedence groups. Two operators in the same group have the same precedence. Higher precedence operators are applied before lower precedence operators. (i) is highest; (vi) is lowest. (i)$\mathbf{A}\bigcirc, \mathbf{E}\bigcirc, \mathbf{A}\Diamond, \mathbf{E}\Diamond, \mathbf{A}\Box, \mathbf{E}\Box, \mathbf{A}\,\mathcal{U}, \mathbf{E}\,\mathcal{U}, \mathbf{A}\,\mathcal{W}, \mathbf{E}\,\mathcal{W}, \mathbf{E}\bigcirc_{\langle ind\rangle}, \mathbf{E}\Diamond_{\langle LC(ind)\rangle}$, $\mathbf{E}\Box_{\langle LC(ind)\rangle}$, $\mathbf{E}\,\mathcal{U}_{\langle LC(ind)\rangle}$, $\mathbf{E}\,\mathcal{W}_{\langle LC(ind)\rangle}$; (ii)$\neg$; (iii)$\wedge$; (iv)$\vee$; (v)$\Rightarrow$ and (vi)$\Leftrightarrow$. Then the language of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses consists of formulae of the following forms.

$$\mathbf{A}\Box(\mathbf{start} \Rightarrow \textstyle\bigvee_{j=1}^{k} m_j) \qquad \text{(initial clause)}$$

$$\mathbf{A}\Box(\mathbf{true} \Rightarrow \textstyle\bigvee_{j=1}^{k} m_j) \qquad \text{(global clause)}$$

$$\mathbf{A}\Box(\textstyle\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{A}\bigcirc \textstyle\bigvee_{j=1}^{k} m_j) \qquad \text{(}\mathbf{A}\text{-step clause)}$$

$$\mathbf{A}\Box(\textstyle\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{E}\bigcirc \textstyle\bigvee_{j=1}^{k} m_{j\,\langle ind\rangle}) \qquad \text{(}\mathbf{E}\text{-step clause)}$$

$$\mathbf{A}\Box(\textstyle\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{A}\Diamond l) \qquad \text{(}\mathbf{A}\text{-sometime clause)}$$

$$\mathbf{A}\Box(\textstyle\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle}) \qquad \text{(}\mathbf{E}\text{-sometime clause)}$$

where $k \geq 0$, $n > 0$, **start** is a propositional constant, l_i $(1 \leq i \leq n)$, m_j $(1 \leq j \leq k)$ and l are literals, that is atomic propositions or their negation and *ind* is an element of Ind. A clause which is either an initial, a global, an **A**-step, or an **E**-step clause is also called a *determinate clause*. Note that the right-hand side of an **E**-sometime or an **A**-sometime clause only contains a single literal. This property simplifies the formulation of the eventuality resolution rules. As all clauses are of the form $\mathbf{A}\Box(P \Rightarrow D)$ we often simply write $P \Rightarrow D$ instead. The formula $\mathbf{A}\Diamond(\neg)l$ is called an **A**-*eventuality* and the formula $\mathbf{E}\Diamond(\neg)l_{\langle LC(ind)\rangle}$ is called an **E**-*eventuality*. We call a clause which is either an initial, a global, an **A**-step, or an **E**-step clause a *determinate clause*. In the remainder of this paper, two concepts, a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and a conjunction of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses, are interchangeable.

To provide a semantics for $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$, we extend model structures $\langle S, R, L\rangle$ to $\langle S, R, L, [\_], s_0\rangle$ where s_0 is an element of S and $[\_] : \mathsf{Ind} \rightarrow 2^{(S\times S)}$ maps every index $ind \in \mathsf{Ind}$ to a *successor function* $[ind]$ which is a total functional relation on S and a subset of R, that is, for every $s \in S$, there exists only one state $s' \in S, (s, s') \in [ind]$ and $(s, s') \in R$. We extend $[\_]$ to expressions of the form $LC(ind)$ by defining $[LC(ind)] = [ind]^*$, i.e. $[LC(ind)]$ is the reflexive and transitive closure of $[ind]$. The semantics of $\text{SNF}^{\text{g}}_{\text{CTL}}$ is then defined below as an extension of the semantics of CTL defined in Section 2.

$$
\begin{array}{ll}
\langle M, s_i\rangle \models \textbf{start} & \text{iff } s_i = s_0 \\
\langle M, s_i\rangle \models \mathbf{E}\bigcirc\psi_{\langle ind\rangle} & \text{iff there exists } s' \in S \text{ such that } (s_i, s') \in [ind] \text{ and } \langle M, s'\rangle \models \psi \\
\langle M, s_i\rangle \models \mathbf{E}\Diamond\psi_{\langle LC(ind)\rangle} & \text{iff there exists } s_j \in S \text{ such that } (s_i, s_j) \in [LC(ind)] \text{ and } \langle M, s_j\rangle \models \psi \\
\langle M, s_i\rangle \models \mathbf{E}\Box\psi_{\langle LC(ind)\rangle} & \text{iff for every } s_j \in S,\ (s_i, s_j) \in [LC(ind)] \text{ implies } \langle M, s_j\rangle \models \psi \\
\langle M, s_i\rangle \models \mathbf{E}(\varphi\,\mathcal{U}\,\psi)_{\langle LC(ind)\rangle} & \text{iff there exists } s_j \in S \text{ such that } (s_i, s_j) \in [LC(ind)] \text{ and } \langle M, s_j\rangle \models \psi \\
& \quad \text{and for every } s_k \in S \text{ with } s_k \neq s_j, \text{ if } (s_i, s_k) \in [LC(ind)] \text{ and} \\
& \quad (s_k, s_j) \in [LC(ind)] \text{ then } \langle M, s_k\rangle \models \varphi \\
\langle M, s_i\rangle \models \mathbf{E}(\varphi\,\mathcal{W}\,\psi)_{\langle LC(ind)\rangle} & \text{iff } \langle M, s_i\rangle \models \mathbf{E}\Box\varphi_{\langle LC(ind)\rangle} \text{ or } \langle M, s_i\rangle \models \mathbf{E}(\varphi\,\mathcal{U}\,\psi)_{\langle LC(ind)\rangle}
\end{array}
$$

The semantics of the remaining operators is analogous to that given previously but in the extended model structure $\langle S, R, L, [\_], s_0\rangle$.

Because the first step of our transformation from a CTL formula φ into a set T of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses defined later in Section 3.2 will add a CTL operator $\mathbf{A}\Box$ around φ and introduce a constant **start**, which is only true at the state s_0, into T, our definitions of a *valid* formula, a *satisfiable* formula and a *unsatisfiable* formula is slightly different from Emerson's definitions. A $\text{SNF}^{\text{g}}_{\text{CTL}}$ formula φ is *valid*, written $\models \varphi$, iff for every model structure $M = \langle S, R, L, [\_], s_0\rangle, M, s_0 \models \varphi$. A $\text{SNF}^{\text{g}}_{\text{CTL}}$ formula φ is *satisfiable*, iff for some model structure $M = \langle S, R, L, [\_], s_0\rangle, M, s_0 \models \varphi$, and *unsatisfiable* otherwise. A model structure $M = \langle S, R, L, [\_], s_0\rangle$ such that φ is true at the state $s_0 \in M$ is called a *model* of φ.

Figure 2 and Figure 3 depict the model structure satisfying the formulae $\mathbf{E}\bigcirc p_{\langle ind_1\rangle}$ and $\mathbf{E}\Diamond p_{\langle LC(ind_1)\rangle}$, respectively.

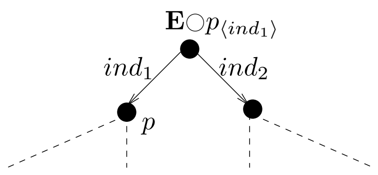


Figure 2: An *ind* example

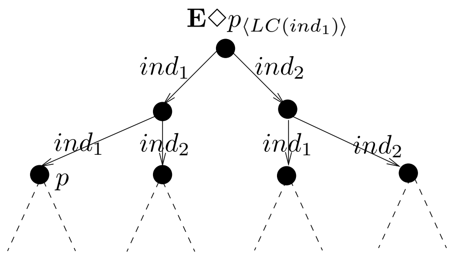


Figure 3: An $LC(ind)$ example

The last three formulae for which we have stated a semantics in the definition above, namely $\mathbf{E}\Box\psi_{\langle LC(ind)\rangle}$, $\mathbf{E}(\varphi\,\mathcal{U}\,\psi)_{\langle LC(ind)\rangle}$ and $\mathbf{E}(\varphi\,\mathcal{W}\,\psi)_{\langle LC(ind)\rangle}$, do not occur in $\mathrm{SNF}_{\mathrm{CTL}}$ clauses. We give their semantics because they appear in the process of transforming a CTL formula to a set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses which we explain in Section 3.2. Therefore, their semantics is required for the correctness proof of the transformation.

3.2 Transformation from CTL into $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$

We have defined a set of transformation rules which allows us to transform an arbitrary CTL formula into an equi-satisfiable set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. The transformation rules are similar to those in [5, 16], but modified to allow for global clauses.

Let *nnf* denote a method which transforms an arbitrary CTL formula into its negation normal form by pushing negations 'inwards'. Let *simp* be a method which simplifies an arbitrary CTL formula by exhaustive application of the following simplification rules,

$$\begin{array}{rr}
(\varphi \land \mathbf{true}) \longrightarrow \varphi & (\varphi \land \mathbf{false}) \longrightarrow \mathbf{false} \\
(\varphi \lor \mathbf{true}) \longrightarrow \mathbf{true} & (\varphi \lor \mathbf{false}) \longrightarrow \varphi \\
\neg\mathbf{true} \longrightarrow \mathbf{false} & \neg\mathbf{false} \longrightarrow \mathbf{true}
\end{array}$$

where φ is a CTL formula and $\lor$ and $\land$ are commutative and associative, plus the following rules which are based on the equivalences in [13]. $\mathbf{P} \in \{\mathbf{A}, \mathbf{E}\}$ and $* \in \{\bigcirc, \Box, \Diamond\}$.

$$\begin{array}{rr}
\mathbf{P} * \mathbf{false} \longrightarrow \mathbf{false} & \mathbf{P} * \mathbf{true} \longrightarrow \mathbf{true} \\
\mathbf{P}(\varphi\,\mathcal{U}\,\mathbf{false}) \longrightarrow \mathbf{false} & \mathbf{P}(\varphi\,\mathcal{U}\,\mathbf{true}) \longrightarrow \mathbf{true} \\
\mathbf{P}(\mathbf{false}\,\mathcal{U}\,\varphi) \longrightarrow \varphi & \mathbf{P}(\mathbf{true}\,\mathcal{U}\,\varphi) \longrightarrow \mathbf{P}\Diamond\varphi \\
\mathbf{P}(\varphi\,\mathcal{W}\,\mathbf{false}) \longrightarrow \mathbf{P}\Box\varphi & \mathbf{P}(\varphi\,\mathcal{W}\,\mathbf{true}) \longrightarrow \mathbf{true} \\
\mathbf{P}(\mathbf{false}\,\mathcal{W}\,\varphi) \longrightarrow \varphi & \mathbf{P}(\mathbf{true}\,\mathcal{W}\,\varphi) \longrightarrow \mathbf{true}
\end{array}$$

We start the transformation of an arbitrary CTL formula φ into the set $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}(\varphi)$ with the set $T_0 = \{\mathbf{A}\Box(\mathbf{start} \Rightarrow p), \mathbf{A}\Box(p \Rightarrow \mathit{simp}(\mathit{nnf}(\varphi)))\}$, where p is a new proposition that does not occur in φ. We then construct a sequence $T_0, T_1, \ldots, T_n$ of formulae such that for every $i, 0 \leq i < n, T_{i+1} = (T_i \setminus \{\psi\}) \cup R_i$, where ψ is a formula in T_i not in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ and R_i is the result of applying a matching transformation rule to ψ. Moreover, for every $i, 0 \leq i < n$, T_i contains at least one formula not in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ while all formulae in T_n are in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$. We assume that throughout the transformation formulae are kept in negation normal form.

Note that for each rule of *Trans* containing a proposition p, p represents a new proposition which does not occur in T_i while we apply the rule to $\psi \in T_i$. Let

- q, q_1, q_2, q_3 be atomic propositions,
- D be a disjunction of literals (possible consisting of a single literal), and
- φ, φ_1 and φ_2, be CTL formulae.

The definition of the rule set *Trans*:

- Index introduction rules:

$$\begin{array}{lll} \mathit{Trans}(1) & q \Rightarrow \mathbf{E}\bigcirc\varphi \longrightarrow q \Rightarrow \mathbf{E}\bigcirc\varphi_{\langle ind\rangle} & \\ \mathit{Trans}(2) & q \Rightarrow \mathbf{E}*\varphi \longrightarrow q \Rightarrow \mathbf{E}*\varphi_{\langle LC(ind)\rangle} & \text{where } * \in \{\Diamond, \Box\}. \\ \mathit{Trans}(3) & q \Rightarrow \mathbf{E}(\varphi_1 * \varphi_2) \longrightarrow q \Rightarrow \mathbf{E}(\varphi_1 * \varphi_2)_{\langle LC(ind)\rangle} & \text{where } * \in \{\mathcal{U}, \mathcal{W}\}. \end{array}$$

  ind is a new index.

- Boolean rules:

$$\mathit{Trans}(4) \quad q \Rightarrow \varphi_1 \wedge \varphi_2 \longrightarrow \begin{cases} q \Rightarrow \varphi_1 \\ q \Rightarrow \varphi_2 \end{cases}$$

$$\mathit{Trans}(5) \quad q \Rightarrow \varphi_1 \vee \varphi_2 \longrightarrow \begin{cases} q \Rightarrow \varphi_1 \vee p \\ p \Rightarrow \varphi_2 \end{cases} \quad \text{if } \varphi_2 \text{ is not a literal or a disjunction of literals.}$$

$$\mathit{Trans}(6) \quad q \Rightarrow D \longrightarrow \mathbf{true} \Rightarrow \neg q \vee D$$

- Temporal operator rules:

$$\mathit{Trans}(7) \quad q \Rightarrow \mathbf{X}*\varphi \longrightarrow \begin{cases} q \Rightarrow \mathbf{X}*p \\ p \Rightarrow \varphi \end{cases} \quad \text{if } \varphi \text{ is not a literal or a disjunction of literals.}$$

$$\mathbf{X}* \in \{\mathbf{A}\bigcirc, \mathbf{E}\bigcirc_{\langle ind\rangle}\}$$

$$\mathit{Trans}(8) \quad q \Rightarrow \mathbf{X}*\varphi \longrightarrow \begin{cases} q \Rightarrow \mathbf{X}*p \\ p \Rightarrow \varphi \end{cases} \quad \text{if } \varphi \text{ is not a literal.}$$

$$\mathbf{X}* \in \{\mathbf{A}\Box, \mathbf{A}\Diamond, \mathbf{E}\Box_{\langle LC(ind)\rangle}, \mathbf{E}\Diamond_{\langle LC(ind)\rangle}\}$$

$$\textit{Trans}(9) \qquad q \Rightarrow \mathbf{X}(\varphi_1 * \varphi_2) \longrightarrow \begin{cases} q \Rightarrow \mathbf{X}(p * \varphi_2) \\ p \Rightarrow \varphi_1 \end{cases} \qquad \text{if } \varphi_1 \text{ is not a literal.}$$

$$\mathbf{X}* \in \{\mathbf{A}\mathcal{U}, \mathbf{A}\mathcal{W}, \mathbf{E}\mathcal{U}_{\langle LC(ind)\rangle}, \mathbf{E}\mathcal{W}_{\langle LC(ind)\rangle}\}$$

$$\textit{Trans}(10) \qquad q \Rightarrow \mathbf{X}(\varphi_1 * \varphi_2) \longrightarrow \begin{cases} q \Rightarrow \mathbf{X}(\varphi_1 * p) \\ p \Rightarrow \varphi_2 \end{cases} \qquad \text{if } \varphi_2 \text{ is not a literal.}$$

$$\mathbf{X}* \in \{\mathbf{A}\mathcal{U}, \mathbf{A}\mathcal{W}, \mathbf{E}\mathcal{U}_{\langle LC(ind)\rangle}, \mathbf{E}\mathcal{W}_{\langle LC(ind)\rangle}\}$$

$$\textit{Trans}(11) \qquad q_1 \Rightarrow \mathbf{A}\Box q_2 \longrightarrow \begin{cases} q_1 \Rightarrow q_2 \wedge p \\ p \Rightarrow \mathbf{A}\bigcirc(q_2 \wedge p) \end{cases}$$

$$\textit{Trans}(12) \qquad q_1 \Rightarrow \mathbf{E}\Box q_{2\langle LC(ind)\rangle} \longrightarrow \begin{cases} q_1 \Rightarrow q_2 \wedge p \\ p \Rightarrow \mathbf{E}\bigcirc(q_2 \wedge p)_{\langle ind\rangle} \end{cases}$$

$$\textit{Trans}(13) \qquad q_1 \Rightarrow \mathbf{A}(q_2 \mathcal{U} q_3) \longrightarrow \begin{cases} q_1 \Rightarrow q_3 \vee (q_2 \wedge p) \\ p \Rightarrow \mathbf{A}\bigcirc(q_3 \vee (q_2 \wedge p)) \\ q_1 \Rightarrow \mathbf{A}\Diamond q_3 \end{cases}$$

$$\textit{Trans}(14) \qquad q_1 \Rightarrow \mathbf{E}(q_2 \mathcal{U} q_3)_{\langle LC(ind)\rangle} \longrightarrow \begin{cases} q_1 \Rightarrow q_3 \vee (q_2 \wedge p) \\ p \Rightarrow \mathbf{E}\bigcirc(q_3 \vee (q_2 \wedge p))_{\langle ind\rangle} \\ q_1 \Rightarrow \mathbf{E}\Diamond q_{3\langle LC(ind)\rangle} \end{cases}$$

$$\textit{Trans}(15) \qquad q_1 \Rightarrow \mathbf{A}(q_2 \mathcal{W} q_3) \longrightarrow \begin{cases} q_1 \Rightarrow q_3 \vee (q_2 \wedge p) \\ p \Rightarrow \mathbf{A}\bigcirc(q_3 \vee (q_2 \wedge p)) \end{cases}$$

$$\textit{Trans}(16) \qquad q_1 \Rightarrow \mathbf{E}(q_2 \mathcal{W} q_3)_{\langle LC(ind)\rangle} \longrightarrow \begin{cases} q_1 \Rightarrow q_3 \vee (q_2 \wedge p) \\ p \Rightarrow \mathbf{E}\bigcirc(q_3 \vee (q_2 \wedge p))_{\langle ind\rangle} \end{cases}$$

Here we provide a proof to show our transformation is terminating, allows only a polynomial bounded number of rule applications and preserves satisfiability.

We start by showing that the first step of our transformation preserve satisfiability.

Lemma 1 *If a set of clauses T is satisfiable in a model $M = \langle S, R, L, [\_], s_0\rangle$, p is a proposition not occurring in T and a model $M' = \langle S, R, L', [\_], s_0\rangle$ is identical to M except that p occurs in L' with an arbitrary truth value assignment in each state of M', then T is also satisfiable in M'.*

Proof. By the inductive definition of the semantics of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$, the truth value assignments to propositions not occurring in T do not influence whether T is satisfiable in a model. Therefore, T is satisfiable in M'. $\square$

Lemma 2 *A CTL formula φ is satisfiable iff the set of formulae $\{\mathbf{A}\square(\mathbf{start} \Rightarrow p), \mathbf{A}\square(p \Rightarrow \varphi)\}$ is satisfiable.*

Proof. Assume $\{\mathbf{A}\square(\mathbf{start} \Rightarrow p), \mathbf{A}\square(p \Rightarrow \varphi)\}$ is satisfiable in a model $M = \langle S, R, L, [\_], s_0\rangle$ at the state s_0, i.e. $M, s_0 \models \mathbf{A}\square(\mathbf{start} \Rightarrow p) \wedge \mathbf{A}\square(p \Rightarrow \varphi)$. From the semantics of $\Rightarrow, \mathbf{A}\square, \wedge$, $M, s_0 \models (\mathbf{start} \Rightarrow p) \wedge (p \Rightarrow \varphi)$. From the semantics of $\Rightarrow, \wedge$, $M, s_0 \models \mathbf{start} \Rightarrow \varphi$. Because **start** holds at s_0, $M, s_0 \models \varphi$. Thus, if $\{\mathbf{A}\square(\mathbf{start} \Rightarrow p), \mathbf{A}\square(p \Rightarrow \varphi)\}$ is satisfiable, so is φ.

Assume φ is satisfiable in a model $M = \langle S, R, L, [\_], s_0\rangle$ at the state s_0, i.e. $M, s_0 \models \varphi$. Let $M' = \langle S, R, L', [\_], s_0\rangle$ be identical to M except that p holds only at s_0. From the semantics of $\mathbf{start}, \Rightarrow, \mathbf{A}\square$, $M', s_0 \models \mathbf{A}\square(\mathbf{start} \Rightarrow p)$. From Lemma 1, $M', s_0 \models \varphi$. From the semantics of $\Rightarrow, \mathbf{A}\square$, $M', s_0 \models \mathbf{A}\square(p \Rightarrow \varphi)$. From the semantics of $\wedge$, $M', s_0 \models \mathbf{A}\square(\mathbf{start} \Rightarrow p) \wedge \mathbf{A}\square(p \Rightarrow \varphi)$. Thus, if φ is satisfiable, so is $\{\mathbf{A}\square(\mathbf{start} \Rightarrow p), \mathbf{A}\square(p \Rightarrow \varphi)\}$. $\square$

The first transformation rule we consider is the rule *Trans*(1).

Lemma 3 *If a set of clauses T is satisfiable in a model $M = \langle S, R, L, [\_], s_0\rangle$, the index ind is not in the set of indices $\mathrm{Ind}(T)$ occurring in T and a model $M' = \langle S, R, L, [\_]', s_0\rangle$ is identical to M except that $[ind]$ is an arbitrary function for each state of M', then T is also satisfiable in M'.*

Proof. By the inductive definition of the semantics of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$, the successor function $[ind]$ such that $ind \notin \mathrm{Ind}(T)$, does not influence whether T is satisfiable in a model. Therefore, T is satisfiable in M'.$\square$

Lemma 4 *If $\psi \to R_i$ by an application of Trans(1), then $T_i = \Delta \cup \{\psi\}$ is satisfiable iff $T_{i+1} = \Delta \cup R_i$ is satisfiable.*

Proof. Let $\mathrm{Ind}(T_i)$ be the set of indices occurring in T_i. Let ψ be $\mathbf{A}\square(q \Rightarrow \mathbf{E}\bigcirc\varphi)$, then R_i is $\mathbf{A}\square(q \Rightarrow \mathbf{E}\bigcirc\varphi_{\langle ind\rangle})$, where $ind \notin \mathrm{Ind}(T_i)$.

Assume a model $M = \langle S, R, L, [\_], s_0\rangle$ satisfies T_{i+1} at the state $s_0 \in S$, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\square(q \Rightarrow \mathbf{E}\bigcirc\varphi_{\langle ind\rangle})$. From the semantics of $\wedge$, $M, s_0 \models \Delta$ and $M, s_0 \models \mathbf{A}\square(q \Rightarrow \mathbf{E}\bigcirc\varphi_{\langle ind\rangle})$. From the semantics of $\mathbf{A}\square$, $M, s_0 \models \Delta$ and for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M, s_j \models q \Rightarrow \mathbf{E}\bigcirc\varphi_{\langle ind\rangle}$. From the semantics of $\Rightarrow, \vee$ and $\mathbf{E}\bigcirc_{\langle ind\rangle}$, $M, s_0 \models \Delta$ and for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M, s_j \models \neg q$ or there exists a state s' such that $(s_j, s') \in [ind]$ and $M, s' \models \varphi$. From the semantics of $\mathbf{E}\bigcirc$, $M, s_0 \models \Delta$ and for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M, s_j \models \neg q$ or $M, s_j \models \mathbf{E}\bigcirc\varphi$. Therefore, from the semantics of $\vee, \Rightarrow, \wedge, \mathbf{A}\square$, we obtain $M, s_0 \models \Delta \wedge \mathbf{A}\square(q \Rightarrow \mathbf{E}\bigcirc\varphi)$. Thus, if T_{i+1} is satisfiable, then so is T_i.

Next we prove the 'if' part. Assume a model $M = \langle S, R, L, [\_], s_0\rangle$ satisfies T_i at the state $s_0 \in S$, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\square(q \Rightarrow \mathbf{E}\bigcirc\varphi)$. We can obtain that $M, s_0 \models \Delta$ and for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M, s_j \models \neg q$ or there exists a path χ_{s_j}, there exists a state $s' \in \chi_{s_j}$ such that $(s_j, s') \in R$ and $M, s' \models \varphi$ using a similar approach to that in the paragraph above. A model $M' = \langle S, R, L, [\_]', s_0 \rangle$ is identical to M except $(s_j, s') \in [ind]$. From the semantics of $\mathbf{E}\bigcirc_{\langle ind \rangle}$, for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M', s_j \models \neg q$ or $M', s_j \models \mathbf{E}\bigcirc\varphi_{\langle ind \rangle}$. From the semantics of $\vee, \Rightarrow, \mathbf{A}\Box$, $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{E}\bigcirc\varphi_{\langle ind \rangle})$. Moreover, Lemma 3 shows that $M', s_0 \models \Delta$. Therefore, from the semantics of $\wedge$, $M', s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{E}\bigcirc\varphi_{\langle ind \rangle})$. Thus, if T_i is satisfiable, then so is T_{i+1}. □

The next rule we consider is the rule *Trans*(4).

Lemma 5 *If $\psi \rightarrow R_i$ by an application of Trans(4), then $T_i = \Delta \cup \{\psi\}$ is satisfiable iff $T_{i+1} = \Delta \cup R_i$ is satisfiable.*

Proof. Assume $T_i = \Delta \wedge \mathbf{A}\Box(q \Rightarrow \varphi_1 \wedge \varphi_2)$ is satisfiable in a model structure $M = \langle S, R, L, [\_], s_0 \rangle$ at the state s_0 in M. Based on the semantics of the various operators, we have that

$\langle M, s_0 \rangle \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \varphi_1 \wedge \varphi_2)$

iff $\langle M, s_0 \rangle \models \Delta \wedge \mathbf{A}\Box((q \Rightarrow \varphi_1) \wedge (q \Rightarrow \varphi_2))$

iff $\langle M, s_0 \rangle \models \Delta$ and for each future path χ_{s_0}, for each $s_j \in \chi_{s_0}$, $\langle M, s_j \rangle \models q \Rightarrow \varphi_1$ and $\langle M, s_j \rangle \models q \Rightarrow \varphi_2$

iff $\langle M, s_0 \rangle \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \varphi_1) \wedge \mathbf{A}\Box(q \Rightarrow \varphi_2)$ □

Now the rule we consider is the rule *Trans*(6), which is not one of the transformation rules that can be found in [5, 7].

Lemma 6 *If $\psi \rightarrow R_i$ by an application of Trans(6), then $T_i = \Delta \cup \{\psi\}$ is satisfiable iff $T_{i+1} = \Delta \cup R_i$ is satisfiable.*

Proof. $\mathbf{A}\Box(q \Rightarrow D)$ is obviously equivalent to $\mathbf{A}\Box(\mathbf{true} \Rightarrow \neg q \vee D)$ as $q \Rightarrow D$ is propositionally equivalent to $\mathbf{true} \Rightarrow \neg q \vee D$. Therefore, T_i is actually equivalent to T_{i+1}. □

The next rule we consider is the rule *Trans*(7).

Lemma 7 *If $\psi \rightarrow R_i$ by an application of Trans(7), then $T_i = \Delta \cup \{\psi\}$ is satisfiable iff $T_{i+1} = \Delta \cup R_i$ is satisfiable.*

Proof. We only prove the lemma for $\psi = q \Rightarrow \mathbf{A}\bigcirc\varphi$ as the proof for $\psi = q \Rightarrow \mathbf{E}\bigcirc\varphi_{\langle ind \rangle}$ is analogous.

We first show the 'if' part. Assume T_{i+1} is satisfiable in a model structure $M = \langle S, R, L, [\_], s_0 \rangle$, i.e.$\langle M, s_0 \rangle \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc p) \wedge \mathbf{A}\Box(p \Rightarrow \varphi)$ iff (a) $M, s_0 \models \Delta$ and (b) for each path χ_{s_0}, for each $s_j \in \chi_{s_0} \langle M, s_j \rangle \models \neg q$ or for each path χ_{s_j}, $\langle M, s_{j+1} \rangle \models p$ and (c) for each path χ_{s_0}, for each $s_k \in \chi_{s_0}$ $\langle M, s_k \rangle \models p \Rightarrow \varphi$.

According to (b), if q holds at the state s_j, then $\mathbf{A}\bigcirc p$ must hold at the state s_j and for each path χ_{s_j}, p must hold at the state s_{j+1} with $(s_j, s_{j+1}) \in R$. Furthermore, by (c) we know φ must hold at the state s_{j+1} and therefore $\mathbf{A}\bigcirc\varphi$ holds at the state s_j and so does $q \Rightarrow \mathbf{A}\bigcirc\varphi$. From the semantics of $\mathbf{A}\Box$ and (a), we obtain $\langle M, s_0\rangle \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$. Therefore, if T_{i+1} is satisfiable, so is T_i.

Next, we prove the 'only if' part. Assume that T_i is satisfiable in a model structure $M = \langle S, R, L, [\_], s_0\rangle$ at the state s_0, i.e.$M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$. We define a model structure M' to be a model structure identical to M except that p is true at a state s_i iff φ is true at s_i. By the definition of M', we have that $\mathbf{A}\Box(p \Leftrightarrow \varphi)$ holds in M', that is, $\langle M', s_0\rangle \models \mathbf{A}\Box(p \Leftrightarrow \varphi)$. Furthermore, as $q \Rightarrow \mathbf{A}\bigcirc\varphi$ is true at a state s_i in M' iff $q \Rightarrow \mathbf{A}\bigcirc\varphi$ is true at s_i in M, $\mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$ is satisfiable in M', that is, $\langle M', s_0\rangle \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$. From $\langle M', s_0\rangle \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$ and the semantics of $\Rightarrow$, $\vee$ and $\mathbf{A}\Box$, for each future path χ_{s_0}, for each state $s_j \in \chi_{s_0}, \langle M', s_j\rangle \models \neg q$ or $\langle M', s_j\rangle \models \mathbf{A}\bigcirc\varphi$. From the semantics of $\mathbf{A}\bigcirc$, for each future path χ_{s_0}, for each state $s_j \in \chi_{s_0}$, $\langle M', s_j\rangle \models \neg q$ or for each future path χ_{s_j}, $\langle M', s_{j+1}\rangle \models \varphi$. From $\langle M', s_{j+1}\rangle \models \varphi$ and $\langle M', s_0\rangle \models \mathbf{A}\Box(p \Leftrightarrow \varphi)$, we obtain $\langle M', s_{j+1}\rangle \models p$. So, for each future path χ_{s_0}, for each state $s_j \in \chi_{s_0}, \langle \mathcal{M}', s_j\rangle \models \neg q$ or $\langle M', s_j\rangle \models \mathbf{A}\bigcirc p$. Therefore, from the semantics of $\vee$, $\Rightarrow$ and $\mathbf{A}\Box$, $\langle M', s_0\rangle \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc p)$ and from Lemma 1, $M', s_0 \models \Delta$. Thus, if T_i is satisfiable, so is T_{i+1}. □

Lemma 8 *If $\psi \to R_i$ by an application of Trans(11), then $T_i = \Delta \cup \{\psi\}$ is satisfiable iff $T_{i+1} = \Delta \cup R_i$ is satisfiable.*

Proof. Assume T_i is satisfiable in $M = \langle S, R, L, [\_], s_0\rangle$ at the state s_0, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q_1 \Rightarrow \mathbf{A}\Box q_2)$. Let $M' = \langle S, R, L', [\_], s_0\rangle$ be identical to M except that $M', s \models p$ iff $M', s \models \mathbf{A}\Box q_2$, for every state s in S. From Lemma 1, $M', s_0 \models \Delta \wedge \mathbf{A}\Box(q_1 \Rightarrow \mathbf{A}\Box q_2)$. From the semantics of $\mathbf{A}\Box, \Rightarrow, \wedge$, we obtain $M', s_0 \models \mathbf{A}\Box(q_1 \Rightarrow q_2 \wedge p) \wedge \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(q_2 \wedge p))$. From the semantics of $\wedge$, $M', s_0 \models \Delta \wedge \mathbf{A}\Box(q_1 \Rightarrow q_2 \wedge p) \wedge \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(q_2 \wedge p))$. We proved that if T_i is satisfiable, so is T_{i+1}.

Next we prove the 'if' part. Assume T_{i+1} is satisfiable in $M = \langle S, R, L, [\_], s_0\rangle$ at the state s_0, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q_1 \Rightarrow q_2 \wedge p) \wedge \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(q_2 \wedge p))$. From the semantics of $\mathbf{A}\Box, \Rightarrow, \mathbf{A}\bigcirc$ and $M, s_0 \models \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(q_2 \wedge p))$, we obtain $M, s_0 \models \mathbf{A}\Box(p \Rightarrow \mathbf{A}\Box q_2)$. From the semantics of $\Rightarrow, \mathbf{A}\Box$ and $M, s_0 \models \mathbf{A}\Box(q_1 \Rightarrow q_2 \wedge p)$, we obtain $M, s_0 \models \mathbf{A}\Box(q_1 \Rightarrow \mathbf{A}\Box q_2)$. From the semantics of $\wedge$ and $M, s_0 \models \Delta$, we obtain $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q_1 \Rightarrow \mathbf{A}\Box q_2)$. Thus, if T_{i+1} is satisfiable, so is T_i. □

Corollary 1 *If T_{i+1} is obtained by an application of a transformation rule $\psi \to R_i$, then T_i is satisfiable iff T_{i+1} is satisfiable.*

Proof. We have proved that *Trans*$(1, 4, 6, 7, 11)$ preserve satisfiability in Lemma 4, 5, 6, 7, and 8, respectively.

The proofs for *Trans*$(2, 3)$ are analogous to Lemma 4; the proof for *Trans*(5) is analogous to Lemma 5; the proofs for *Trans*$(8-10)$ are analogous to Lemma 7; the proofs for *Trans*$(12-16)$ are analogous to Lemma 8. □

Theorem 1 *Let φ be an arbitrary CTL formula and T_n be a set of SNF$^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained from $T_0 = \{\mathbf{A}\Box(\mathbf{start} \Rightarrow p), \mathbf{A}\Box(p \Rightarrow simp(nnf(\varphi)))\}$ by a sequence of applications of our transformation rules. Then φ is satisfiable iff T_n is satisfiable.*

Proof. We prove this by induction over the length of the sequence $T_0, T_1, \ldots, T_n$ constructed by a sequence of applications of the transformation rules.

For the base case, in [13] the author shows the functions *simp* and *nnf* preserve equivalence. By Lemma 2, φ is satisfiable iff T_0 is satisfiable.

For the induction step, by Corollary 1, T_i is satisfiable iff T_{i+1} is satisfiable. Therefore, φ is satisfiable iff T_n is satisfiable. $\Box$

Theorem 2 *Let φ be an arbitrary CTL formula and T_n be a set of SNF$^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained from $T_0 = \{\mathbf{A}\Box(\mathbf{start} \Rightarrow p), \mathbf{A}\Box(p \Rightarrow simp(nnf(\varphi)))\}$ by a sequence of applications of our transformation rules. Then the set of indices* $\mathrm{Ind}(T_n)$ *occurring in T_n is finite.*

Proof. We observe that the number of $\mathbf{E}$ path quantifiers in φ and T_0 is finite and the only three transformation rules which introduce new indices are $\mathit{Trans}(1-3)$. Each rule can be applied exactly once to a particular occurrence of an unindexed $\mathbf{E}$ path quantifier. Thus, there exists a one-to-one mapping between $\mathbf{E}$ path quantifiers in T_0 and indices in T_n and the set $\mathrm{Ind}(T_n)$ is finite. $\Box$

Definition 1 [*Length of a CTL formula*]

The length of a CTL formula φ is the number of path quantifiers plus the number of the occurrences of propositions and constants in φ.

Definition 2 [*Size of a set of formulae (clauses)*]

The size of a set T of formulae (clauses) is the sum of the length of each formula (clause) in T.

Lemma 9 *A transformation of an arbitrary CTL formula φ into a set T_n of* SNF$^{\mathrm{g}}_{\mathrm{CTL}}$ *clauses can be done by a linearly bounded number of applications of our transformation rules.*

Proof. From the definition of the transformation procedure, a transformation of a CTL formula φ into a set T_n of SNF$^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is a sequence $T_0, T_1, \ldots, T_n$ of formulae such that

- $T_0 = \{\mathbf{A}\Box(\mathbf{start} \Rightarrow p), \mathbf{A}\Box(p \Rightarrow simp(nnf(\varphi)))\}$, where p is a new proposition, and
- for every $i, 0 \leq i < n, T_{i+1} = (T_i \setminus \psi) \cup R_i$ such that $\psi \Rightarrow R_i$ by application of one of the transformation rules and
- for every $i, 0 \leq i < n, T_i$ contains at least one formula not in SNF$^{\mathrm{g}}_{\mathrm{CTL}}$ and all the formulae in T_n are in SNF$^{\mathrm{g}}_{\mathrm{CTL}}$.

Let the size of T_0 be m.

For $\mathit{Trans}(1-3)$, each rule can be applied exactly once to a particular occurrence of an unindexed $\mathbf{E}$ path quantifier in T_0. Thus, the number of applications of each rule is bounded by m.

Each application of $\mathit{Trans}(11)$ removes one $\mathbf{A}\Box$ operator at a time and no other rules generate $\mathbf{A}\Box$ operators. Obviously, the number of applications of $\mathit{Trans}(11)$ is bounded by m. Also the number of applications of $\mathit{Trans}(12-16)$ is bounded by m for the same reason.

Next, consider $\mathit{Trans}(7)$. Each application of $\mathit{Trans}(7)$ removes an occurrence of $\mathbf{A}\bigcirc\varphi_1$ or $\mathbf{E}\bigcirc\varphi_{2\langle ind\rangle}$, where φ_1 and φ_2 are not disjunctions of literals. But this case is different from the above, $\mathit{Trans}(11-16)$ generate formulae containing $\mathbf{A}\bigcirc\varphi_1$ or $\mathbf{E}\bigcirc\varphi_{2\langle ind\rangle}$. Since the number of applications of $\mathit{Trans}(11-16)$ is bounded by m, therefore, the number of applications of $\mathit{Trans}(7)$ is bounded by $6m+m=7m$, where $6m$ is the contribution from $\mathit{Trans}(11-16)$ and m is the contribution from T_0.

The proofs for $\mathit{Trans}(4-6,8-10)$ are analogous to the proof $\mathit{Trans}(7)$. The number of application of each rule is bounded by a linear number.

To transform any formula not in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ in $T_0, T_1, \ldots, T_{n-1}$, we need to consider all the possible forms which are not in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$, for example, $q \Rightarrow \mathbf{A}\Box\varphi_3, q \Rightarrow \varphi_4 \wedge \varphi_5, \ldots$, where $\varphi_3, \varphi_4, \varphi_5$ are arbitrary CTL formulae. The following table shows all the possible forms and the corresponding rules handling them. (In the table below $\psi_1, \psi_2, \ldots, \psi_n$ are arbitrary CTL formulae, q is a proposition, and l is a literal.)

The possible form	rule	The possible form	rule
$q \Rightarrow \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_n$	4	$q \Rightarrow \mathbf{E}\bigcirc\psi_{1\langle ind\rangle}$	7
$q \Rightarrow \psi_1 \vee \psi_2 \vee \ldots \vee \psi_n$	5, 6	$q \Rightarrow \mathbf{A}\Box\psi_1$	8, 11
$q \Rightarrow (\neg)l$	6	$q \Rightarrow \mathbf{E}\Box\psi_{1\langle LC(ind)\rangle}$	8, 12
$q \Rightarrow \mathbf{E}\bigcirc\psi_1$	1	$q \Rightarrow \mathbf{A}\Diamond\psi_1$	8
$q \Rightarrow \mathbf{E}\Diamond\psi_1$	2	$q \Rightarrow \mathbf{E}\Diamond\psi_{1\langle LC(ind)\rangle}$	8
$q \Rightarrow \mathbf{E}\Box\psi_1$	2	$q \Rightarrow \mathbf{A}(\psi_1 \,\mathcal{U}\, \psi_2)$	9, 10, 13
$q \Rightarrow \mathbf{E}(\psi_1 \,\mathcal{U}\, \psi_2)$	3	$q \Rightarrow \mathbf{E}(\psi_1 \,\mathcal{U}\, \psi_2)_{\langle LC(ind)\rangle}$	9, 10, 14
$q \Rightarrow \mathbf{E}(\psi_1 \,\mathcal{W}\, \psi_2)$	3	$q \Rightarrow \mathbf{A}(\psi_1 \,\mathcal{W}\, \psi_2)$	9, 10, 15
$q \Rightarrow \mathbf{A}\bigcirc\psi_1$	7	$q \Rightarrow \mathbf{E}(\psi_1 \,\mathcal{W}\, \psi_2)_{\langle LC(ind)\rangle}$	9, 10, 16

As the table above shows our transformation rules cover all the possibilities and the number of applications of each rules is bounded by a linear number, the whole transformation to φ can be done by a linear bounded number of applications. $\Box$

Corollary 2 *Let φ be an arbitrary CTL formula and T_n be the set of $\mathit{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained from $T_0 = \{\mathbf{A}\Box(\mathbf{start} \Rightarrow p), \mathbf{A}\Box(p \Rightarrow \mathit{simp}(\mathit{nnf}(\varphi)))\}$ by applications of our transformation rules. Then our transformation procedure terminates.*

Proof. From Lemma 9, we know the number of applications of our transformation has an upper bound, then our transformation of an arbitrary CTL formula into a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses terminates. □

Theorem 3 *Let φ_0 be an arbitrary CTL formula and T_n be the set of $SNF^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained from $T_0 = \{\mathbf{A}\Box(\mathbf{start} \Rightarrow p), \mathbf{A}\Box(p \Rightarrow simp(nnf(\varphi_0)))\}$ by applications of our transformation rules. Then the set T_n can be computed in polynomial time.*

Proof. We use tree structures for the representation of CTL formulae. Then the cost of $Trans(1-3)$ is obviously constant time.

For the rules $Trans(4-16)$, the proofs are very similar. So we only provide one, namely $Trans(7)$ as an example. In $Trans(7)$ we first replace φ with p, then create the structure "$p \Rightarrow$" and attach φ to $p \Rightarrow$. The manipulation of formulae in $Trans(7)$ shows that it is a procedure requiring constant steps. Similarly, the cost of the rest rules are constant time as well.

However, to determine which rule to be applied, we need to scan the current set of clauses T_i. Suppose we use BFS method, then the cost is linear time in the size of the current set T_i. From all the rules, we know for each application transforming from T_i to T_{i+1}, size of the set grows in a constant number. Thus, the current set T_i is linear in the size of T_0. Then we conclude that each rule application needs linear time in the size of T_0.

From Lemma 9, T_n can be obtained in polynomial time. □

3.3 The clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$

The resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ consists of two types of resolution rules, the *step* resolution rules, SRES1 to SRES8, and the *eventuality* resolution rules, ERES1 and ERES2, as well as two rewrite rules, RW1 and RW2.

Motivated by refinements of propositional and first-order resolution [4], we restrict the applicability of step resolution rules by means of an atom ordering and a selection function.

An *atom ordering* for $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is a well-founded and total ordering $\succ$ on the set P_{PL}. The ordering $\succ$ is extended to literals by identifying each positive literal p with the singleton multiset $\{p\}$ and each negative literal $\neg p$ with the multiset $\{p, p\}$ and comparing such multisets of atoms by using the multiset extension of $\succ$. Doing so, $\neg p$ is greater than p, but smaller than any literal q or $\neg q$ with $q \succ p$.

In this section, we assume that conjunctions and disjunctions of propositional literals do not contain duplicate occurrences of the same literal, that is, the operators $\vee$ and $\wedge$ are idempotent. We use **false** to denote the empty disjunction and **true** to denote the empty conjunction. A literal l is (*strictly*) *maximal* with respect to a propositional disjunction C iff for every literal l' in C, $l' \not\succ l$ ($l' \not\succeq l$).

A *selection function* is a function S mapping every propositional disjunction C to a possibly empty subset $S(C)$ of the negative literals occurring in C. If $l \in S(C)$ for a disjunction C, then we say that l is *selected* in C.

In the following presentation of the rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, *ind* is an index, P and Q are conjunctions of literals, C and D are disjunctions of literals, and l is a literal. If l is a negative literal $\neg p$, then $\neg l$ denotes the atomic proposition p.

SRES1

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

SRES2

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

SRES3

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

SRES4

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

SRES5

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

SRES6

$$\frac{\begin{array}{c} \mathbf{true} \Rightarrow C \vee l \\ Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l) \end{array}}{Q \Rightarrow \mathbf{A}\bigcirc(C \vee D)}$$

SRES7

$$\frac{\begin{array}{c} \mathbf{true} \Rightarrow C \vee l \\ Q \Rightarrow \mathbf{E}\bigcirc(D \vee \neg l)_{\langle ind \rangle} \end{array}}{Q \Rightarrow \mathbf{E}\bigcirc(C \vee D)_{\langle ind \rangle}}$$

SRES8

$$\frac{\begin{array}{c} \mathbf{true} \Rightarrow C \vee l \\ \mathbf{true} \Rightarrow D \vee \neg l \end{array}}{\mathbf{true} \Rightarrow C \vee D}$$

A step resolution rule, SRES1 to SRES8, is only applicable if one of the following two conditions is satisfied:

(C1) if l is a positive literal, then l must be strictly maximal with respect to C and no literal is selected in $C \vee l$, and $\neg l$ must be selected in $D \vee \neg l$ or no literal is selected in $D \vee \neg l$ and $\neg l$ is maximal with respect to D; or

(C2) if l is a negative literal, then l must be selected in $C \vee l$ or no literal is selected in $C \vee l$ and l is maximal with respect to C, and $\neg l$ must be strictly maximal with respect to D and no literal is selected in $D \vee \neg l$.

Note that these two conditions are identical modulo the polarity of l. If l in $C \vee l$ and $\neg l$ in $D \vee \neg l$ satisfy condition (C1) or condition (C2), then way say that l is *eligible* in $C \vee l$ and $\neg l$ is *eligible* in $D \vee \neg l$.

The rewrite rules RW1 and RW2 are defined as follows:

$$\textbf{RW1} \qquad \bigwedge_{i=1}^{n} m_i \Rightarrow \mathbf{A}\bigcirc\mathbf{false} \longrightarrow \mathbf{true} \Rightarrow \bigvee_{i=1}^{n} \neg m_i$$

$$\textbf{RW2} \qquad \bigwedge_{i=1}^{n} m_i \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind\rangle} \longrightarrow \mathbf{true} \Rightarrow \bigvee_{i=1}^{n} \neg m_i$$

where $n \geq 1$ and each m_i, $1 \leq i \leq n$, is a literal.

The intuition of the eventuality resolution rule ERES1 below is to resolve an eventuality $\mathbf{A}\Diamond\neg l$, which states that $\Diamond\neg l$ is true on all paths, with a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses which together, provided that their combined left-hand sides were true, imply that $\Box l$ holds on (at least) one path.

ERES1

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

where $P^{\dagger} \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$ represents a set $\Lambda_{\mathbf{E}\Box}$ of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses

$$\begin{array}{ccc} P^1_1 \Rightarrow *C^1_1 & & P^n_1 \Rightarrow *C^n_1 \\ \vdots & & \vdots \\ P^1_{m_1} \Rightarrow *C^1_{m_1} & \cdots & P^n_{m_n} \Rightarrow *C^n_{m_n} \end{array}$$

with each $*$ either being empty or being an operator in $\{\mathbf{A}\bigcirc\} \cup \{\mathbf{E}\bigcirc_{\langle ind\rangle} \mid ind \in \mathsf{Ind}\}$ and for every i, $1 \leq i \leq n$,

$$(\bigwedge_{j=1}^{m_i} C^i_j) \Rightarrow l \qquad (1)$$

and

$$(\bigwedge_{j=1}^{m_i} C^i_j) \Rightarrow (\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} P^i_j) \qquad (2)$$

are provable. Furthermore, $P^{\dagger} = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} P^i_j$.

Conditions (1) and (2) ensure that the set $\Lambda_{\mathbf{E}\Box}$ of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses implies $P^{\dagger} \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$.

Note that the conclusion of ERES1 is not stated in normal form. To present the conclusion of ERES1 in normal form, we use a new atomic proposition $w^{\mathbf{A}}_{\neg l}$ uniquely associated with the eventuality $\mathbf{A}\Diamond\neg l$. Then the conclusion of ERES1 can be represented by the following set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses:

$$\begin{array}{c} \{w^{\mathbf{A}}_{\neg l} \Rightarrow \mathbf{A}\bigcirc(\neg l \vee \neg(\bigwedge_{j=1}^{m_i} P^i_j) \mid 1 \leq i \leq n\} \\ \cup\, \{\mathbf{true} \Rightarrow \neg Q \vee \neg l \vee \neg(\bigwedge_{j=1}^{m_i} P^i_j) \mid 1 \leq i \leq n\} \\ \cup\, \{\mathbf{true} \Rightarrow \neg Q \vee \neg l \vee w^{\mathbf{A}}_{\neg l}, w^{\mathbf{A}}_{\neg l} \Rightarrow \mathbf{A}\bigcirc(\neg l \vee w^{\mathbf{A}}_{\neg l})\}. \end{array}$$

The use of a proposition $w^{\mathbf{A}}_{\neg l}$ uniquely associated with the eventuality $\mathbf{A}\Diamond\neg l$ is important for the termination of our procedure. If we were to use a new proposition each time we need to transform the conclusion of an application of ERES1 into normal form, then even the repeated application of ERES1 to the same set of premises would lead to distinct sets of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses as conclusion (which would also not be equivalent to each other or subsume each other). This in turn would mean that we could generate an infinite number of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses during a derivation. In contrast, the use of atomic propositions uniquely associated with **A**-eventualities allows us to represent any resolvents by ERES1 using a fixed set of propositions depending only on the initial set of clauses, i.e., n different **A**-eventualities in the initial set of clauses require at most n new atomic propositions to represent resolvents by ERES1.

Similar to ERES1, the intuition underlying the ERES2 rule is to resolve an eventuality $\mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle}$, which states that $\Diamond\neg l$ is true on the path given by $[LC(ind)]$, with a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses which together, provided that their combined left-hand sides were true, imply that $\Box l$ holds on the same path.

ERES2

$$\begin{array}{c} P^\dagger \Rightarrow \mathbf{E}\bigcirc(\mathbf{E}\Box l_{\langle LC(ind)\rangle})_{\langle ind\rangle} \\ Q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle} \\ \hline Q \Rightarrow \mathbf{E}(\neg(P^\dagger)\,\mathcal{W}\,\neg l)_{\langle LC(ind)\rangle} \end{array}$$

where $P^\dagger \Rightarrow \mathbf{E}\bigcirc(\mathbf{E}\Box l_{\langle LC(ind)\rangle})_{\langle ind\rangle}$ represents a set $\Lambda^{ind}_{\mathbf{E}\Box}$ of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses

$$\begin{array}{ccc} P^1_1 \Rightarrow *C^1_1 & & P^n_1 \Rightarrow *C^n_1 \\ \vdots & & \vdots \\ P^1_{m_1} \Rightarrow *C^1_{m_1} & \cdots & P^n_{m_n} \Rightarrow *C^n_{m_n} \end{array}$$

with each $*$ either being empty or being an operator in $\{\mathbf{A}\bigcirc, \mathbf{E}\bigcirc_{\langle ind\rangle}\}$ and for every i, $1 \leq i \leq n$,

$$(\textstyle\bigwedge_{j=1}^{m_i} C^i_j) \Rightarrow l \tag{3}$$

and

$$(\textstyle\bigwedge_{j=1}^{m_i} C^i_j) \Rightarrow (\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} P^i_j) \tag{4}$$

are provable. Furthermore, $P^\dagger = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} P^i_j$.

Again, conditions (3) and (4) ensure that the set $\Lambda^{ind}_{\mathbf{E}\Box}$ of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses implies the formula $P^\dagger \Rightarrow \mathbf{E}\bigcirc(\mathbf{E}\Box l_{\langle LC(ind)\rangle})_{\langle ind\rangle}$.

Also, as for ERES1, the conclusion of ERES2 is not in normal form. This time we use an atomic proposition $w^{ind}_{\neg l}$ uniquely associated with $\Diamond\neg l_{\langle LC(ind)\rangle}$ to represent the resolvent of ERES2 as the following set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses:

$$\begin{array}{l} \{w^{ind}_{\neg l} \Rightarrow \mathbf{E}\bigcirc(\neg l \vee \neg(\bigwedge_{j=1}^{m_i} P^i_j)_{\langle ind\rangle} \mid 1 \leq i \leq n\} \\ \cup\,\{\mathbf{true} \Rightarrow \neg Q \vee \neg l \vee \neg(\bigwedge_{j=1}^{m_i} P^i_j) \mid 1 \leq i \leq n\} \\ \cup\,\{\mathbf{true} \Rightarrow \neg Q \vee \neg l \vee w^{ind}_{\neg l}, \\ \quad w^{ind}_{\neg l} \Rightarrow \mathbf{E}\bigcirc(\neg l \vee w^{ind}_{\neg l})_{\langle ind\rangle}\}. \end{array}$$

As for ERES1, the use of atomic propositions uniquely associated with **E**-eventualities allows us to represent any resolvents by ERES2 using a fixed set of atomic propositions depending only on the initial set of clauses.

The expensive part of applying ERES1 and ERES2 is finding sets of step and global clauses which can serve as premises for these rules, that is, for a given literal l stemming from some eventuality, to find sets of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses $\Lambda_{\mathbf{E}\Box}$, satisfying conditions (1) and (2), and $\Lambda^{ind}_{\mathbf{E}\Box}$, satisfying conditions (3) and (4). Such sets of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses are also called **E**-*loops in* l and the formula $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} P^i_j$ is called a *loop formula*. Algorithms to find loops were first presented by Bolotov and Dixon in [6]. They define two loop search algorithms, called **A**-loop search algorithm and **E**-loop search algorithm. An **A**-loop search algorithm is not required for our calculus as an **E**-loop search algorithm is sufficient to find the premises for both ERES1 and ERES2. Therefore, we only present an **E**-loop search algorithm here. In Section 5 we will discuss in more detail why an **A**-loop search algorithm is not required in our setting, while in Section 4 we present in more detail how the **E**-loop search algorithm can be implemented.

The **E**-loop search algorithm makes use of the notion of *merged clauses* which is inductively defined as follows. Any global clause, **A**-step clause, and **E**-step clause is a merged clause. If $A_1 \Rightarrow B_1$, $A_2 \Rightarrow B_2$, $A_3 \Rightarrow \mathbf{A}\bigcirc B_3$, $A_4 \Rightarrow \mathbf{A}\bigcirc B_4$, $A_5 \Rightarrow \mathbf{E}\bigcirc B_{5\langle ind\rangle}$, and $A_6 \Rightarrow \mathbf{E}\bigcirc B_{6\langle ind\rangle}$ are merged clauses, then so are $(A_1 \wedge A_2) \Rightarrow (B_1 \wedge B_2)$, $(A_1 \wedge A_4) \Rightarrow \mathbf{A}\bigcirc(B_1 \wedge B_4)$, $(A_1 \wedge A_6) \Rightarrow \mathbf{E}\bigcirc(B_1 \wedge B_6)_{\langle ind\rangle}$, $(A_3 \wedge A_4) \Rightarrow \mathbf{A}\bigcirc(B_3 \wedge B_4)$, $(A_3 \wedge A_6) \Rightarrow \mathbf{E}\bigcirc(B_3 \wedge B_6)_{\langle ind\rangle}$, and $(A_5 \wedge A_6) \Rightarrow \mathbf{E}\bigcirc(B_5 \wedge B_6)_{\langle ind\rangle}$.

E-*loop search algorithm*:
The algorithm takes as input a literal l, stemming either from an **A**-sometime clause $Q \Rightarrow \mathbf{A}\Diamond\neg l$ or from an **E**-sometime clause $Q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle}$, and a set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses among which we search for premises for the eventuality resolution rules. We assume the set T is saturated by step resolution, that is, rules SRES1 to SRES8.

The algorithm proceeds as follows:

1. Search in T for all the clauses of the form $X_j \Rightarrow l$, $X_j \Rightarrow \mathbf{A}\bigcirc l$, and $X_j \Rightarrow \mathbf{E}\bigcirc l_{\langle ind\rangle}$. Assuming there are m_0 such clauses, we build the first node as follows:

$$H_0 = \bigvee_{j=0}^{m_0} X_j$$

   Simplify H_0 using boolean simplification. If $H_0 \equiv \mathbf{true}$ a loop is found, we return **true** and the algorithm terminates. If $H_0 \equiv \mathbf{false}$ (which can only be the case if $m_0 = 0$), then no loop formula can be found and we return **false**.

2. Given a node H_i, where $i \geq 0$, build the next node H_{i+1} by looking in T for merged clauses of the form $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}\bigcirc(B_j \wedge l)_{\langle ind\rangle}$ such that $B_j \Rightarrow H_i$ is provable (in propositional

logic). Assuming there are m_{i+1} such merged clauses, we build the node H_{i+1} as follows:

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

Simplify H_{i+1} using boolean simplification.

3. Repeat the previous step until one of the conditions below is provable (in propositional logic).

   (a) $H_{i+1} \equiv$ **true**. A loop formula has been found. We return **true** and the algorithm terminates.

   (b) $H_{i+1} \equiv$ **false** (i.e., $m_{i+1} = 0$). No loop formula can be found. We return **false** and the algorithm terminates.

   (c) $H_i \equiv H_{i+1}$. A loop formula has been found. We return H_{i+1} and the algorithm terminates.

If we try to resolve an **E**-sometime clause $Q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle}$, then the input set T to the **E**-loop search algorithm consists of the set of all global and **A**-step clauses we currently have at our disposal plus all **E**-step clauses with index *ind*. If we try to resolve an **A**-sometime clause $Q \Rightarrow \mathbf{A}\Diamond\neg l$, then the input set T to the **E**-loop search algorithm consists of the set of all global, **A**-step clauses, and **E**-step clauses.

An important step in the algorithm is the task of "looking for merged clauses", which is again non-trivial. We will discuss this problem in more detail in Section 4.

A *derivation* from a set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is a sequence $T_0, T_1, T_2, \ldots$ of sets of clauses such that $T = T_0$ and $T_{i+1} = T_i \cup R_i$ where R_i is a set of clauses obtained as the conclusion of the application of a resolution rule to premises in T_i. A *refutation* of T (by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$) is a derivation from T such that for some $i \geq 0$, T_i contains a contraction. A derivation $T_0, \ldots, T_i, \ldots$ *terminates* iff either a contradiction is derived or there is no set R_i of clauses derivable as the conclusion of an application of a resolution rule to premises in T_i such that $R_i \not\subseteq T_i$.

3.4 Properties of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$

The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is sound and complete. We first show that $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is sound.

Theorem 4 (Soundness of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$) *Let T be a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. If there is a refutation of T by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, then T is unsatisfiable.*

Proof. The soundness of SRES1 to SRES4, ERES1 and ERES2 has been established in [5]. So we only need to prove the soundness of SRES5 to SRES8, RW1 and RW2.

Let $T_0, T_1, \ldots, T_n$ be a derivation from a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clause $T = T_0$ by the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. We will show by induction over the length of the derivation that if T_0 is satisfiable, then so is T_n.

For $T_0 = T$, the claim obviously holds. Now, consider the step of the derivation in which we derive T_{i+1} from T_i for some $i \geq 0$. Assume T_i is satisfiable and $M = \langle S, R, L, [\_], s_0\rangle$ is a model structure satisfying T_i.

Assume $\mathbf{A}\Box(\mathbf{true} \Rightarrow C \vee l)$ and $\mathbf{A}\Box(\mathbf{start} \Rightarrow D \vee \neg l)$ are in T_i. Let T_{i+1} be obtained by an application of SRES5 to $\mathbf{A}\Box(\mathbf{true} \Rightarrow C \vee l)$ and $\mathbf{A}\Box(\mathbf{start} \Rightarrow D \vee \neg l)$, i.e., $T_{i+1} = T_i \cup \{\mathbf{A}\Box(\mathbf{start} \Rightarrow C \vee D)\}$. We show that M also satisfies T_{i+1}. Consider an arbitrary state $s \in S$. If s is not s_0, then obviously $\langle M, s\rangle \models \mathbf{start} \Rightarrow C \vee D$ because $\mathbf{start}$ is false at the state s. Assume the state s is s_0. From $\langle M, s\rangle \models \mathbf{A}\Box(\mathbf{true} \Rightarrow C \vee l)$ and $\langle M, s\rangle \models \mathbf{A}\Box(\mathbf{start} \Rightarrow D \vee \neg l)$ and the semantics of $\mathbf{A}\Box$, we obtain $\langle M, s\rangle \models \mathbf{true} \Rightarrow C \vee l$ and $\langle M, s\rangle \models \mathbf{start} \Rightarrow D \vee \neg l$. From the semantics of $\mathbf{true}$, $\Rightarrow$ and $\mathbf{start}$, we obtain $\langle M, s\rangle \models C \vee l$ and $\langle M, s\rangle \models D \vee \neg l$. As l and $\neg l$ can not both be true at state s in M, we conclude $\langle M, s\rangle \models C \vee D$. As s is s_0, then from the semantics of $\mathbf{start}$ we have $\langle M, s\rangle \models \mathbf{start} \Rightarrow C \vee D$. Since $\mathbf{start} \Rightarrow C \vee D$ holds in s_0 and all other states, from the semantics of $\mathbf{A}\Box$ we conclude $\langle M, s\rangle \models \mathbf{A}\Box(\mathbf{start} \Rightarrow C \vee D)$. Thus the model structure M satisfies T_{i+1}, T_{i+1} is satisfiable and SRES5 is sound. For rules SRES6 to SRES8, the proofs are analogous to that for SRES5.

Regarding RW1, from the semantics of $\mathbf{A}\bigcirc$ and $\mathbf{false}$ we obtain that the formula $\mathbf{A}\Box(\wedge_{i=1}^{n} Q_i \Rightarrow \mathbf{A}\bigcirc\mathbf{false})$ is true iff $\mathbf{A}\Box(\wedge_{i=1}^{n} Q_i \Rightarrow \mathbf{false})$ is true. This formula is propositionally equivalent to $\mathbf{A}\Box(\vee_{i=1}^{n} \neg Q_i)$ which in turn, by the semantics of $\Rightarrow$ and $\mathbf{true}$, is equivalent to $\mathbf{A}\Box(\mathbf{true} \Rightarrow \vee_{i=1}^{n} \neg Q_i)$. The proof for RW2 is similar. $\Box$

Now we present the completeness proof for $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$. First, we give a brief discussion of how our proof proceeds. We introduce the idea of augmentation, which was originally developed for a resolution calculus for PLTL [16]. Next, we create a finite directed graph, called a labelled behaviour graph, for an augmented set of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses. To create a CTL model structure for a set T of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses from a labelled behaviour graph for T, some nodes and some subgraphs of a labelled behaviour graph graph for T cannot be involved. For instance, a node without any successor nodes in a labelled behaviour graph cannot be used to construct a CTL model structure, as all paths in a CTL model structure are infinite. To remove such nodes and subgraphs from a labelled behaviour graph, we define a set of deletion rules. We call a labelled behaviour graph H a reduced labelled behaviour graph if it is obtained by exhaustively applying deletion rules to H. We show that an augmented set T of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses is unsatisfiable iff its reduced labelled behaviour graph is empty. We also prove that each application of a deletion rule corresponds to a derivation from T by $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$. Therefore, if T is unsatisfiable, its reduced labelled behaviour graph H_{red} is empty and the sequence of application of the deletion rules, which reduce the labelled behaviour graph for T to an empty H_{red}, can be used to construct a refutation in $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$. In the following, we show the detailed completeness proof.

Let T be a set of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses obtained by applying the normal form transformation to a given CTL formula. Recall from Section 3.3 an application of ERES1 or ERES2 to the set T may introduce new propositions into T, but these propositions are uniquely associated with $\mathbf{A}\Diamond l$ and $\mathbf{E}\Diamond l_{\langle LC(ind)\rangle}$ formulae occurring in T. For our completeness proof it is convenient to assume that certain clauses associated with these new propositions are present in T right from the beginning, i.e. independent of applications of ERES1 and ERES2. We adapt the *augmentation* procedure used in [16] for PLTL to CTL to establish a relation between new atomic proposition introduced by an application of ERES1 or ERES2 and eventualities associated with them.

Definition 3 [*Augmentation*]

Given a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clause T, we construct an augmented set T_a as follows: the augmented set T_a is the smallest set containing T and satisfying the following conditions:

- For every **A**-sometime clause in T, $Q \Rightarrow \mathbf{A}\Diamond\neg l$, T_a contains the clauses

$$\begin{aligned} \mathbf{true} &\Rightarrow \neg Q \vee \neg l \vee w^{\mathbf{A}}_{\neg l} \\ w^{\mathbf{A}}_{\neg l} &\Rightarrow \mathbf{A}\bigcirc(\neg l \vee w^{\mathbf{A}}_{\neg l}) \end{aligned}$$

  where $w^{\mathbf{A}}_{\neg l}$ is a new proposition uniquely associated with $\mathbf{A}\Diamond\neg l$.

- For every **E**-sometime clause in T, $Q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle}$, T_a contains the clauses

$$\begin{aligned} \mathbf{true} &\Rightarrow \neg Q \vee \neg l \vee w^{ind}_{\neg l}, \\ w^{ind}_{\neg l} &\Rightarrow \mathbf{E}\bigcirc(\neg l \vee w^{ind}_{\neg l})_{\langle ind\rangle} \end{aligned}$$

  where $w^{ind}_{\neg l}$ is a new proposition uniquely associated with $\mathbf{E}\Diamond l_{\langle LC(ind)\rangle}$.

A proof that augmentation preserves satisfiability can be found in [5].

Given a set *Ind* of indices an *ind*-labelled graph H is an ordered pair $H = (N, E)$, where N is the set of nodes and E is the set of edges in H. An edge $(n, ind, n') \in E$ is a directed edge from a node $n \in N$ to a node $n' \in N$ labelled with a label $ind \in Ind$. (When the label on the edge is not important, we also use (n, n') to denote an edge, which means the label can be any index in *Ind*.)

Definition 4 [*ind-reachable node in a graph*]

Given a set *Ind* of indices, an *ind*-labelled graph (N, E), and a node $n \in N$, a node $n' \in N$ is *ind*-reachable from n iff there exists an edge $(n, ind, n') \in E$ or there exists an edge $(n'', ind, n') \in E$ and n'' is *ind*-reachable from n.

Definition 5 [*reachable node in a graph*]

Given a graph (N, E) and a node $n \in N$, a node $n' \in N$ is reachable from n iff there exists an edge $(n, n') \in E$ or there exists an edge $(n'', n') \in E$ and n'' is reachable from n.

Definition 6 [*labelled behaviour graph*]

Let T be a set of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses and $\text{Ind}(T)$ be the set of indices occurring in T. If $\text{Ind}(T)$ is empty, then let $\text{Ind}(T) = \{ind\}$, where ind is an arbitrary index in Ind. Given T and $\text{Ind}(T)$, we construct a finite directed graph $H = (N, E)$, called a *labelled behaviour graph* for T.

A node $n = (V, E_A, E_E)$ in H is a triple, where V, E_A, E_E are constructed as follows. Let V be a valuation of propositions occurring in T. Let E_A be a subset of $\{l \mid Q \Rightarrow \mathbf{A}\Diamond l \in T\}$ and E_E be a subset of $\{l_{\langle LC(ind)\rangle} \mid Q \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle} \in T\}$. Informally E_A and E_E contain eventualities that need to be satisfied either in the current node or some node reachable from the current node.

To define the set of edges E of H we use the following auxiliary definitions. Let $n = (V, E_A, E_E)$ be a node in N. Let $R_A(n, T) = \{D \mid Q \Rightarrow \mathbf{A}\bigcirc D \in T$, and $V \models Q\}$. Note if V does not satisfy the left-hand side of any **A**-step clause (i.e. $R_A(n, T) = \emptyset$), then there are no constraints from **A**-step clauses on the next node of the node n and any valuation satisfies $R_A(n, T)$. Let $R_{ind}(n, T) = \{D \mid Q \Rightarrow \mathbf{E}\bigcirc D_{\langle ind\rangle} \in T$ and $V \models Q\}$. Let $R_g(T) = \{D \mid \mathbf{true} \Rightarrow D \in T\}$.

Let functions $\mathsf{Ev}_{\mathbf{A}}(V, T)$ and $\mathsf{Ev}_{\mathbf{E}}(V, T)$ be defined as $\mathsf{Ev}_{\mathbf{A}}(V, T) = \{l \mid Q \Rightarrow \mathbf{A}\Diamond l \in T$ and $V \models Q\}$ and $\mathsf{Ev}_{\mathbf{E}}(V, T) = \{l_{\langle LC(ind)\rangle} \mid Q \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle} \in T$, and $V \models Q\}$, respectively.

Let functions $\mathsf{Unsat}_{\mathbf{A}}(E_A, V)$ and $\mathsf{Unsat}_{ind}(E_E, V)$ be defined as $\mathsf{Unsat}_{\mathbf{A}}(E_A, V) = \{l \mid l \in E_A$ and $V \not\models l\}$ and $\mathsf{Unsat}_{ind}(E_E, V) = \{l_{\langle LC(ind)\rangle} \mid l_{\langle LC(ind)\rangle} \in E_E$ and $V \not\models l\}$, respectively.

Then E contains an edge labelled by ind from a node (V, E_A, E_E) to a node (V', E'_A, E'_E) iff V' satisfies the set $R_A(n, T) \cup R_{ind}(n, T) \cup R_g(T)$, $E'_A = \mathsf{Unsat}_{\mathbf{A}}(E_A, V) \cup \mathsf{Ev}_{\mathbf{A}}(V', T)$ and $E'_E = \mathsf{Unsat}_{ind}(E_E, V) \cup \mathsf{Ev}_{\mathbf{E}}(V', T)$.

Let $R_0(T) = \{D \mid \mathbf{start} \Rightarrow D \in T\}$. Then the node (V, E_A, E_E), where V satisfies the set $R_0(T) \cup R_g(T)$, $E_A = \mathsf{Ev}_{\mathbf{A}}(V, T)$ and $E_E = \mathsf{Ev}_{\mathbf{E}}(V, T)$, is an initial node of H. The *labelled behaviour graph* for an augmented set of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses T is the set of nodes and edges reachable from the initial nodes.

Example 1 [*labelled behaviour graph*]

For the augmented set of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses below, the labelled behaviour graph is shown in Figure 4.

1. $\mathbf{start} \Rightarrow \neg q$
2. $\mathbf{start} \Rightarrow p$
3. $\mathbf{true} \Rightarrow p \lor q$
4. $p \Rightarrow \mathbf{A}\bigcirc p$
5. $q \Rightarrow \mathbf{E}\bigcirc \neg p_{\langle 1\rangle}$
6. $p \Rightarrow \mathbf{E}\Diamond q_{\langle LC(2)\rangle}$
7. $\mathbf{true} \Rightarrow \neg p \lor q \lor w^2_q$ (added by augmentation)
8. $w^2_q \Rightarrow \mathbf{E}\bigcirc(q \lor w^2_q)_{\langle 2\rangle}$ (added by augmentation)

Definition 7 [*Path from a node n to a node n' through a graph*]

A path from a node n_1 to a node n_n in a graph is a sequence of nodes $n_1, n_2, \ldots, n_n$ such that $(n_1, n_2), (n_2, n_3), \ldots, (n_{n-1}, n_n)$ are the edges of the graph.

Definition 8 [*Shortest path from a node n to a node n' through a graph*]

A shortest path from a node n to a node n' in a graph is a path from the node n to the node n' with the least number of edges amongst all the paths from the node n to the node n'.

Definition 9 [*Distance*]

Given a graph (N, E), if a node $n' \in N$ is reachable from another node $n \in N$, the distance from n to n' is the number of edges in a shortest path from n to n'.

Definition 10 [*ind-distance*]

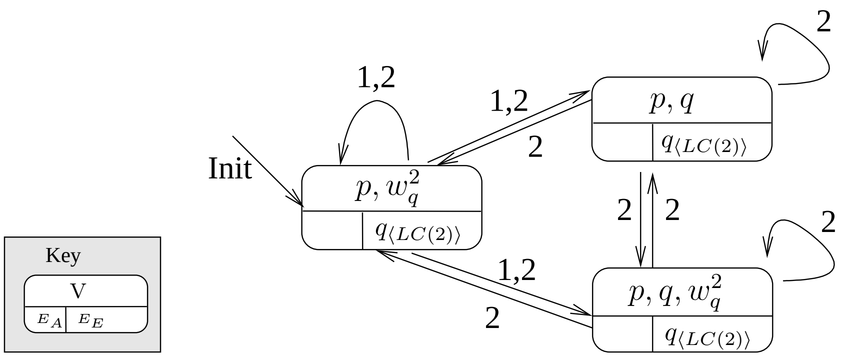
Given a graph (N, E), if a node $n' \in N$ is *ind*-reachable from a node $n \in N$, the *ind*-distance from n to n' is the number of edges in a shortest path such that every edge in it is labelled by *ind*.

Lemma 10 *Let T be an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and let $H = (N, E)$ be the labelled behaviour graph for T. If H contains an edge from a node $n = (V, E_A, E_E) \in N$ to a node $n' = (V', E'_A, E'_E) \in N$ such that $l \in E'_A$ then either there exists a clause $Q \Rightarrow \mathbf{A}\Diamond l \in T$ such that $V' \models Q$ or $l \in E_A$ and $V \not\models l$.*

Proof. From the construction of the labelled behaviour graph, we know $E'_A = \mathsf{Unsat}_{\mathbf{A}}(E_A, V) \cup \mathsf{Ev}_{\mathbf{A}}(V', T)$. Therefore, if $l \in E'_A$, then l is from either $\mathsf{Unsat}_{\mathbf{A}}(E_A, V)$ or $\mathsf{Ev}_{\mathbf{A}}(V', T)$. For the first case, l must be in E_A and $V \not\models l$. For the latter, there exists a clause $Q \Rightarrow \mathbf{A}\Diamond l \in T$ such that $V' \models Q$. □

Lemma 11 *Let T be an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and let $H = (N, E)$ be the labelled behaviour graph for T. For every node $n = (V, E_A, E_E)$ in H if $l \in E_A$ and $V \not\models l$ then $V \models w^{\mathbf{A}}_l$.*

Proof. From Lemma 10 if an edge from a node $n' = (V', E'_A, E'_E)$ to a node $n = (V, E_A, E_E)$ such that $l \in E_A$ exists in H then either there exists a clause $Q \Rightarrow \mathbf{A}\Diamond l \in T$ such that $V \models Q$ or $l \in E'_A$ and



Init
p, w^2_q
$q_{\langle LC(2)\rangle}$
p, q
$q_{\langle LC(2)\rangle}$
p, q, w^2_q
$q_{\langle LC(2)\rangle}$
1,2
1,2
2
2
2
2
2
1,2
2
Key
V
E_A
E_E

Figure 4: A labelled behaviour graph

$V' \not\models l$. Thus, by induction $l \in E_A$ is originally generated from a node where Q holds. The proof is by induction on the length of the shortest path from a node $n'' = (V'', E''_A, E''_E)$ such that there exists a clause $Q \Rightarrow \mathbf{A}\Diamond l \in T$ and $V'' \models Q$ to the node n.

In the base case the length is zero, i.e. $n = n''$ and therefore $V \models Q$ and by construction we obtain $l \in E_A$. By augmentation we know $\mathbf{true} \Rightarrow \neg Q \vee l \vee w_l^{\mathbf{A}} \in T$. By assumption $l \in E_A$ and $V \not\models l$, hence $V \models w_l^{\mathbf{A}}$.

Otherwise assume the lemma holds for nodes n' such that the distance from a node n'', where $V'' \models Q$, to the node n' is m and we prove it holds for those $n = (V, E_A, E_E)$ with $(n', ind, n) \in E, ind \in \mathrm{Ind}(T)$. By assumption $l \in E_A$ and $V \not\models l$. From the inductive hypothesis we have $V' \models w_l^{\mathbf{A}}$. By augmentation we have $w_l^{\mathbf{A}} \Rightarrow \mathbf{A}\bigcirc(w_l^{\mathbf{A}} \vee l) \in T$. Thus by construction we have $V \models w_l^{\mathbf{A}}$. □

Lemma 12 *Let T be an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and let $H = (N, E)$ be the labelled behaviour graph for T. If H contains an edge from a node $n = (V, E_A, E_E) \in N$ to a node $n' = (V', E'_A, E'_E) \in N$ such that $l_{\langle LC(ind)\rangle} \in E'_E$ then either there exists a clause $Q \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle} \in T$ such that $V' \models Q$ or $l_{\langle LC(ind)\rangle} \in E_E$ and $V \not\models l$.*

Proof. From the construction of the labelled behaviour graph, we know that $E'_E = \mathsf{Unsat}_{ind}(E_E, V) \cup \mathsf{Ev}_{\mathbf{E}}(V', T)$. Therefore, if $l_{\langle LC(ind)\rangle} \in E'_E$, then $l_{\langle LC(ind)\rangle}$ is from either $\mathsf{Unsat}_{ind}(E_E, V)$ or $\mathsf{Ev}_{\mathbf{E}}(V', T)$. For the first case, $l_{\langle LC(ind)\rangle}$ must be in E_E and $V \not\models l$. For the latter, there exists a clause $Q \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle} \in T$ and $V' \models Q$. □

Lemma 13 *Let T be an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and $H = (N, E)$ be the labelled behaviour graph for T. If for every node $n = (V, E_A, E_E)$ in H such that $l_{\langle LC(ind)\rangle} \in E_E$ and $V \not\models l$ then $V \models w_l^{ind}$.*

Proof. The proof proceeds in analogy of the proof of Lemma 11 and uses the fact that T contains the clause $w_l^{ind} \Rightarrow \mathbf{E}\bigcirc(w_l^{ind} \vee l)_{\langle ind\rangle}$. □

Lemma 14 *Let T be an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and let T' be a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained from T by adding any combination of initial, **A**-step, **E**-step or global clauses which only involve propositions and indices occurring in T. Then the labelled behaviour graph $H' = (N', E')$ of T' is a subgraph of the labelled behaviour graph $H = (N, E)$ of T.*

Proof. This is established by induction on the length of the shortest path from an initial node to a node in H'. For the base case, where the length of the path is zero, we show that any initial node in H' is an initial node in H.

As T' has been constructed by adding an combination of initial, **A**-step , **E**-step and global clauses to T, we have $R_0(T) \subseteq R_0(T')$ and $R_g(T) \subseteq R_g(T')$. Take any initial node $n_0 = (V_0, E_{A_0}, E_{E_0})$ in H'. By Definition 6, V_0 must satisfy $R_0(T') \cup R_g(T')$. As $R_0(T) \subseteq R_0(T')$ and $R_g(T) \subseteq R_g(T')$ then V_0

must also satisfy $R_0(T) \cup R_g(T)$. As the set of **A**- and **E**-sometime clauses in T and T' is the same. V_0 satisfies the left hand side of the same **A**- and **E**-sometime clauses in T and T' the sets E_{A_0} and E_{E_0} will be the same in both graphs. By Definition 6 n_0 is also an initial node in H.

Next we assume that every node $n_i = (V_i, E_{A_i}, E_{E_i})$, where the length of the shortest path in H' from an initial node to n_i is m, is in H. We show that every node $n_{i+1} = (V_{i+1}, E_{A_{i+1}}, E_{E_{i+1}})$ in H' with an incoming edge $(n_i, ind, n_{i+1}) \in E', ind \in \text{Ind}(T)$ is also in H.

V_{i+1} satisfies $R_A(n_i, T') \cup R_{ind}(n_i, T')$. Thus V_{i+1} also satisfies $R_A(n_i, T) \cup R_{ind}(n_i, T)$, as $R_A(n_i, T) \subseteq R_A(n_i, T')$ and $R_{ind}(n_i, T) \subseteq R_{ind}(n_i, T')$. As we have already proved that $R_g(T) \subseteq R_g(T')$, so V_{i+1} must also satisfy $R_g(T)$ as well. Furthermore as no change has been made to any **A**- or **E**-sometime clause in T, every eventuality outstanding from n_i or triggered by n_{i+1} will be the same in both graphs. Thus n_{i+1} is also present in H as is the edge (n_i, ind, n_{i+1}).

The proof that all the edges in H' are also in H is analogous to the proof above for nodes.

Therefore, $N' \subseteq N, E' \subseteq E$ and $H' \subseteq H$. □

Definition 11 [*Terminal node*]

A node n in a labelled behaviour graph for T is a terminal node iff there exists an index $ind \in \text{Ind}(T)$ such that no edges labelled with ind depart from n.

Note that in the labelled behaviour graph shown in Figure 4, the two nodes on the right-hand side of the graph are terminal nodes as they do not have any outgoing edges labelled with the index 1.

Definition 12 [*ind-labelled terminal subgraph for $l_{\langle LC(ind)\rangle}$*]

For a labelled behaviour graph (N, E) for T, a subgraph (N', E') is an *ind*-labelled terminal subgraph for $l_{\langle LC(ind)\rangle}$ of (N, E) iff

(ITS1) $N' \subseteq N$ and $E' \subseteq E$;

(ITS2) for all nodes n, $n' \in N$ and edges $(n, ind', n') \in E$, $n' \in N'$ and $(n, ind', n') \in E'$ iff $n \in N'$ and $ind = ind'$; and

(ITS3) for every node $n = (V, E_A, E_E) \in N'$, $l_{\langle LC(ind)\rangle} \in E_E$ and $V \models \neg l$.

Definition 13 [*Terminal subgraph for l*]

For a labelled behaviour graph (N, E) for T, a subgraph (N', E') is a terminal subgraph for l of (N, E) iff

(TS1) $N' \subseteq N$ and $E' \subseteq E$;

(TS2) for every node $n \in N'$ there exists some index $ind \in \text{Ind}(T)$ such that for all edges $(n, ind, n') \in E$, $n' \in N'$ and $(n, ind, n') \in E'$; and

(TS3) for every node $n = (V, E_A, E_E) \in N'$, $l \in E_A$ and $V \models \neg l$.

Figure 5 and Figure 6 show examples of an *ind*-labelled terminal subgraph for $q_{\langle LC(2)\rangle}$ and a terminal subgraph for q, respectively. (In both cases we assume the set of indices in the clause set for these labelled behaviour graphs is $\{1, 2\}$.)

Lemma 15 *Given a labelled behaviour graph $H = (N, E)$ and a node $n = (V, E_A, E_E) \in N$, if, for every eventuality $l_{\langle LC(ind)\rangle} \in E_E$, $l_{\langle LC(ind)\rangle}$ can be satisfied in n or in some node ind-reachable from n, then n is not in any ind-labelled terminal subgraph $H' = (N', E')$ for $l_{\langle LC(ind)\rangle}$ of H.*

Proof. Let $H' = (N', E')$ be an arbitrary *ind*-labelled terminal subgraph for some arbitrary eventuality $l_{\langle LC(ind)\rangle}$ of H. Proving this lemma is equivalent to proving that if $n \in N'$, then $l_{\langle LC(ind)\rangle}$ cannot be satisfied in n nor in any nodes *ind*-reachable from n in H. Assume that $n \in N'$. According to property (ITS2), all nodes which are *ind*-reachable from n are also in N'. By property (ITS3), for every node $n' = (V', E'_A, E'_E) \in N'$, $l_{\langle LC(ind)\rangle} \in E'_E$ and l is not satisfied in n'. Therefore, $l_{\langle LC(ind)\rangle}$ cannot be satisfied in n nor in any node *ind*-reachable from n in H. □

Definition 14 [*Reduced labelled behaviour graph*]

Given a labelled behaviour graph $H = (N, E)$ for an augmented set of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses T, the *reduced labelled behaviour graph* H_{red} for T is the result of exhaustively applying the following *deletion rules* to H.

1. If $n \in N$ is a terminal node with respect to an index in $\text{Ind}(T)$, then delete n and every edge into or out of n.

2. If there is an *ind*-labelled terminal graph (N', E') of H such that $ind \in \text{Ind}(T)$, then delete every node $n \in N'$ and every edge into or out of nodes in N'.

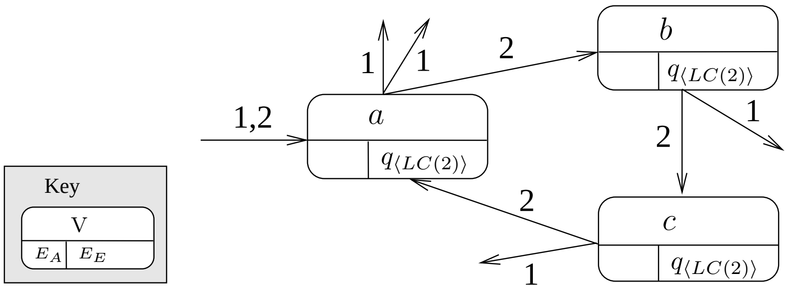

Key
V
E_A
E_E
1,2
1
1
2
a
$q_{\langle LC(2)\rangle}$
b
$q_{\langle LC(2)\rangle}$
1
2
c
$q_{\langle LC(2)\rangle}$
2
1

Figure 5: A 2-labelled terminal subgraph for $q_{\langle LC(2)\rangle}$

3. If there is a terminal graph (N', E') of H with respect to some indices in $\text{Ind}(T)$, then delete every node $n \in N'$ and every edge into or out of nodes in N'.

Lemma 16 *An augmented set of* $\text{SNF}^{\text{g}}_{\text{CTL}}$ *clauses* T *is unsatisfiable if and only if its reduced labelled behaviour graph* H *is empty.*

Proof. We start by showing the 'if' part. If T is satisfiable, then it has a CTL model structure $M = \langle S, R, L, [\_], s_0 \rangle$. We construct a labelled behaviour graph $H = (N, E)$ for T and inductively define a mapping h from M to H. Let P_T be the set of atomic propositions occurring in T. As the CTL model structure M satisfies the clause set T, $L(s_0)$ must satisfy $R_0(T) \cup R_g(T)$, which means there must be an initial node $n_0 = (V_0, E_{A_0}, E_{E_0})$ in H, where $V_0 = L(s_0) \cap P_T$, $E_{A_0} = \mathsf{Ev}_{\mathbf{A}}(V_0, T)$ and $E_{E_0} = \mathsf{Ev}_{\mathbf{E}}(V_0, T)$, and we define $h(s_0) = n_0$.

Next, we assume that $h(s_i) = n_i = (V_i, E_{A_i}, E_{E_i})$ is in H and $(s_i, s_{i+1}) \in [ind]$. As the CTL model structure M satisfies T, $L(s_{i+1})$ must satisfy $R_A(n_i, T) \cup R_{ind}(n_i, T) \cup R_g(T)$, which means there must be a node $n_{i+1} = (V_{i+1}, E_{A_{i+1}}, E_{E_{i+1}})$ in H, where $V_{i+1} = L(s_{i+1}) \cap P_T$, $E_{A_{i+1}} = \mathsf{Ev}_{\mathbf{A}}(V_{i+1}, T) \cup \mathsf{Unsat}_{\mathbf{A}}(E_{A_i}, V_i)$, $E_{E_{i+1}} = \mathsf{Ev}_{\mathbf{E}}(V_{i+1}, T) \cup \mathsf{Unsat}_{ind}(E_{E_i}, V_i)$, and we define $h(s_{i+1}) = n_{i+1}$. By the construction of the behaviour graph, the edge $(h(s_i), ind, h(s_{i+1}))$ is in H. Therefore, for every state $s \in S$, the node $h(s)$ is in H and for every pair $(s_i, s_{i+1}) \in R, i \geqslant 0$, the edge $(h(s_i), h(s_{i+1}))$ is in H.

We define N^M as the set $\{h(s) \mid s \in S\}$ and E^M as the set $\{(h(s_i), ind, h(s_{i+1})) \mid (s_i, s_{i+1}) \in [ind], ind \in \text{Ind}(T), s_i, s_{i+1} \in S\}$. It is obvious that $N^M \subseteq N$ and $E^M \subseteq E$. We define the subgraph H^M of H to be (N^M, E^M). Although a CTL model structure is infinite, H and H^M are finite graphs because the number of nodes in H and H^M is bounded by $2^{n^p} \times 2^{n^A} \times 2^{n^E}$, where n^p, n^A and n^E are the numbers of atomic propositions, $\mathbf{A}$-eventualities and $\mathbf{E}$-eventualities in T, respectively.

We will show that neither of the deletion rules given in Definition 14 is applicable to H^M. Since every state s in a CTL model structure has successors and in particular an *ind*-successor for each index $ind \in \text{Ind}(T)$, by the construction of H^M, deletion rule (1) is not applicable to H^M.

To prove that deletion rule (2) is not applicable to H^M, we need first to show that for every node $h(s) = n = (V, E_A, E_E)$ in H^M, if $l \in E_A$, then $M, s \models \mathbf{A}\Diamond l$. The proof can be done by induction on a

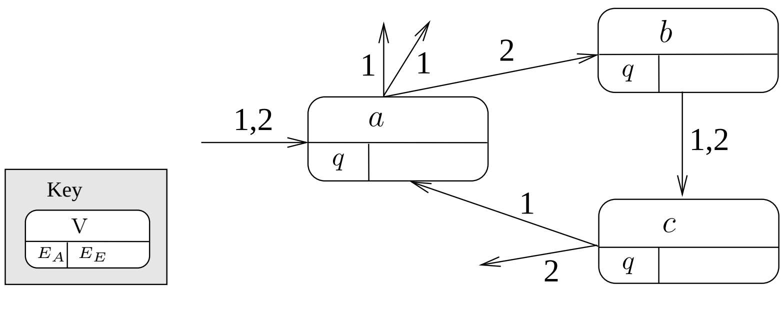


Figure 6: A terminal subgraph for q

path $\chi_{s_i} = s_i, s_{i+1}, s_{i+2}, \dots$ in M such that there exists a clause $\mathbf{A}\Box(Q \Rightarrow \mathbf{A}\Diamond l) \in T$ with $M, s_i \models Q$.

For the base case, we have $h(s_i) = n_i = (V_i, E_{A_i}, E_{E_i})$, $l \in E_{A_i}$, $\mathbf{A}\Box(Q \Rightarrow \mathbf{A}\Diamond l) \in T$, and $M, s_i \models Q$. By the semantics of $\Rightarrow$ and $\mathbf{A}\Box$, we obtain $M, s_i \models \mathbf{A}\Diamond l$.

For the induction step, we have the hypothesis that $h(s_{i+j}) = n_{i+j} = (V_{i+1}, E_{A_{i+j}}, E_{E_{i+j}})$, $l \in E_{A_{i+j}}$ and $M, s_{i+j} \models \mathbf{A}\Diamond l$. By the equivalence $\mathbf{A}\Diamond l \equiv l \vee \mathbf{A}\bigcirc\mathbf{A}\Diamond l$ shown in [13], we obtain $M, s_{i+j} \models l \vee \mathbf{A}\bigcirc\mathbf{A}\Diamond l$. For the node $h(s_{i+j+1}) = n_{i+j+1} = (V_{i+j+1}, E_{A_{i+j+1}}, E_{E_{i+j+1}})$, we know $l \in E_{A_{i+j+1}} = \mathsf{Ev}_{\mathbf{A}}(V_{i+j+1}, T) \cup \mathsf{Unsat}_{\mathbf{A}}(E_{A_{i+j}}, V_{i+j})$. If l is from $\mathsf{Ev}_{\mathbf{A}}(V_{i+j+1}, T)$, then there must exist a clause $\mathbf{A}\Box(Q' \Rightarrow \mathbf{A}\Diamond l) \in T$ and $V_{i+j+1} \models Q'$. By the definition of the mapping h, $M, s_{i+j+1} \models Q'$. Therefore, by the semantics of $\Rightarrow$ and $\mathbf{A}\Box$, $M, s_{i+j+1} \models \mathbf{A}\Diamond l$. Otherwise l must be from $\mathsf{Unsat}_{\mathbf{A}}(E_{A_{i+j}}, V_{i+j})$. By the definition of the function $\mathsf{Unsat}_{\mathbf{A}}$, $V_{i+j} \models \neg l$. By the mapping h, $M, s_{i+j} \models \neg l$. Therefore, by the semantics of $\vee, \mathbf{A}\bigcirc$ and $(s_{i+j}, s_{i+j+1}) \in R$, we obtain $M, s_{i+j+1} \models \mathbf{A}\Diamond l$.

The proof that for every node $h(s) = n = (V, E_A, E_E)$ in H^M, if $l_{\langle LC(ind)\rangle} \in E_E$, then $M, s \models \mathbf{E}\Diamond l_{\langle LC(ind)\rangle}$ can be done in analogy.

Assume node $n = (V, E_A, E_E)$ is in an *ind*-labelled terminal subgraph for $l_{\langle LC(ind)\rangle}$ of H and $h(s) = n$. By Definition 12, we know $l_{\langle LC(ind)\rangle} \in E_E$. So $M, s \models \mathbf{E}\Diamond l_{\langle LC(ind)\rangle}$. By Lemma 15, l does not hold on any the possible paths labelled with *ind* departing from n. By our mapping h, l does not hold on the actual path M used. Therefore, we have $M, s \not\models \mathbf{E}\Diamond l_{\langle LC(ind)\rangle}$. That is a contradiction implying that there are no nodes in an *ind*-labelled terminal subgraph for $l_{\langle LC(ind)\rangle}$ of H and deletion rule (2) is not applicable to H^M.

The proof for deletion rule (3) is analogous to the proof for deletion rule (2) and deletion rule (3) is also not applicable to H^M. As H^M is a subgraph of H and no deletion rules are applicable to H^M, the labelled behaviour graph H for T cannot be reduced to an empty graph.

We now show the 'only if' part. Assume that the reduced labelled behaviour graph $H = (N, E)$ of T is non-empty, then we show how to construct a CTL model structure $M = \langle S, R, L, [\_], s_0\rangle$ from H satisfying T.

First we define some additional notation that will be used later.

Let $M = \langle S, R, L, [\_], s_0\rangle$ be a CTL model structure.

- By $P(s_0, ind)$ we denote a path in M consisting of an infinite sequence $s_0, s_1, s_2, \dots$ of states such that $s_0, s_1, s_2, \dots \in S$ and for every $i \geq 0, (s_i, s_{i+1}) \in [ind]$, and $ind \in \text{Ind}(T)$. Alternatively, we view $P(s_0, ind)$ as an infinite sequence $(s_0, s_1), (s_1, s_2), \dots$ of pairs of states.

- By $P(s_0, *)$ we denote a path in M consisting of an infinite sequence $s_0, s_1, s_2, \dots$ of states such that $s_0, s_1, s_2, \dots \in S$, and for every $i \geq 0, (s_i, s_{i+1}) \in R$. Alternatively, we view $P(s_0, *)$ as an infinite sequence $(s_0, s_1), (s_1, s_2), \dots$ of pairs of states.

- By $RP(s_n)$ we denote a reverse path consisting of a finite sequence $s_n, s_{n-1}, \dots, s_0$ of states such that $s_n, s_{n-1}, \ldots, s_0 \in S$, s_0 is the root of M, and for every i, $0 \leq i \leq n-1, (s_i, s_{i+1}) \in R$.

To construct a CTL model structure $M = \langle S, R, L, [\_], s_0 \rangle$ from a reduced labelled behaviour graph $H = (N, E)$, if $n = (V, E_A, E_E) \in N$, let the function $cs(n)$ return a state s such that $L(s) = V$.

In addition, we introduce several properties of M which are necessary and sufficient for M to satisfy T.

(P1) $L(s_0)$ must satisfy $R_0(T) \cup R_g(T)$.

(P2) Every pair $(s_i, s_{i+1}) \in R$ must satisfy the set of **A**-step, **E**-step and global clauses in T, that is

- $L(s_i)$ and $L(s_{i+1})$ satisfy $R_g(T)$;
- for every **A**-step clause $P \Rightarrow \mathbf{A}\bigcirc Q \in T$ if $L(s_i)$ satisfies P, then $L(s_{i+1})$ must satisfy Q;
- for every **E**-step clause $P \Rightarrow \mathbf{E}\bigcirc Q_{\langle ind \rangle} \in T$ if $L(s_i)$ satisfies P and $(s_i, s_{i+1}) \in [ind]$, then $L(s_{i+1})$ must satisfy Q.

(P3) For every **E**-sometime clause $P \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind) \rangle} \in T$ and every state $s \in S$, if $M, s \models P$, then the path $P(s, ind)$ must contain a state $s' \in S$ such that $l \in L(s')$.

(P4) For every **A**-sometime clause $P \Rightarrow \mathbf{A}\Diamond l \in T$ and every state $s \in S$, if $M, s \models P$, then every path $P(s, *)$ must contain a state $s' \in S$ such that $l \in L(s')$.

Now we inductively define the construction of a CTL model structure from a reduced labelled behaviour graph H and a mapping h from M to H.

The state s_0 of M is given by $s_0 = cs(n_0)$, where n_0 is an arbitrary initial node in H, and we define $h(s_0) = n_0$. By the construction of H, property (P1) holds for s_0.

Suppose we have constructed the state s_i for M and $RP(s_i) = s_i, s_{i-1}, \ldots, s_0$. Then our task is to choose for each index $ind \in \mathrm{Ind}(T)$ a pair $(s_i, s_{i+1}) \in [ind]$ for M. Assume $h(s_i) = n$ and n has k ind-successors $\{n_1, n_2, \ldots, n_k\}$ ($k > 0$ as otherwise n would be a terminal node in H). Let S^{RP} be a set $\{s_j \mid s_{j-1}, s_j \in RP(s_i), h(s_{j-1}) = n, h(s_j) \in \{n_1, n_2, \ldots, n_k\}$ and $(s_{j-1}, s_j) \in [ind]\}$.

- if the set S^{RP} is empty, then $s_{i+1} = cs(n_1)$;
- else, let $s \in S^{SP}$ be the state such that the distance between s_i and s is the shortest among all the distances between s_i and a state in S^{RP} and $h(s) = n_m \in \{n_1, n_2, \ldots, n_k\}, 1 \leqslant m \leqslant k$, then
  - $s_{i+1} = cs(n_{m+1})$, if $m \neq k$;
  - $s_{i+1} = cs(n_1)$, if $m = k$.

By this algorithm, for an arbitrary path χ_{s_0}, if a node n is infinitely often used to construct states $s \in \chi_{s_0}$ and the index ind is infinitely often used to construct the next states of s on χ_{s_0}, then ind-successors of the node n are fairly chosen. This construction ensures that all eventualities will be satisfied in M.

Following the instructions we provided and using a breadth-first order for the construction, from the state s_0, a CTL model structure M is constructed from H. By the construction of M and H, property (P2) holds for M.

Now we prove the model structure M we constructed satisfies property (P3).

Assume the clause $P \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle} \in T$ and let s be an arbitrary state in S such that $M, s \models P$. We need to show that the path $P(s, ind)$ contains a state s' such that $l \in L(s')$. Assume l does not hold on $P(s, ind)$, i.e. $l \notin L(s')$ for any state $s' \in P(s, ind)$. We know the path $P(s, ind)$ is an infinite sequence, whereas the set of nodes in H is finite, which implies that there are nodes $\{n_1^t, n_2^t, \ldots, n_k^t\} \in H, k \geqslant 1$ are used infinitely often to construct the path $P(s, ind)$. Then for $1 \leqslant i \leqslant k$, $n_i^t = (V_i^t, {E_A}_i^t, {E_E}_i^t)$ and by our assumption, $V_i^t \models \neg l$ and $l_{\langle LC(ind)\rangle} \in {E_E}_i^t$. By the way we construct M, we use all the ind-successors of each node in $\{n_1^t, n_2^t, \ldots, n_k^t\}$. Thus, the set of nodes $\{n_1^t, n_2^t, \ldots, n_k^t\}$ in H and all the ind-labelled edges departing from those nodes form an ind-labelled terminal subgraph for $l_{\langle LC(ind)\rangle}$ of H. However, H is a reduced labelled behaviour graph, so no ind-labelled terminal subgraph exists in H. We obtain a contradiction. Therefore, l must hold on some state of the path $P(s, ind)$ and property (P3) holds for M.

The proof that property (P4) holds for M is analogous to the proof that property (P3) holds for M. □

Lemma 17 *If a set of initial and global clauses is unsatisfiable then there is a refutation using only step resolution rules.*

Proof. If a set T of initial and global clauses is unsatisfiable, then the set $T' = \{D \mid \mathbf{true} \Rightarrow D \in T$ or $\mathbf{start} \Rightarrow D \in T\}$ is unsatisfiable by the semantics of $\mathbf{A}\Box$ and **start**.

The set T' only consists of propositional clauses. Therefore, it has a refutation by propositional ordered resolution with selection using the same ordering and selection function as for $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. Then, we can use step resolution rules SRES4, SRES5, and SRES8 on this set T to derive an empty clause, namely either $\mathbf{start} \Rightarrow \mathbf{false}$ or $\mathbf{true} \Rightarrow \mathbf{false}$. □

Lemma 18 *If the unreduced labelled behaviour graph for an augmented set of* $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ *clauses T is empty then a contradiction can be obtained by applying step resolution rules to clauses in or derived from T.*

Proof. If the unreduced labelled behaviour graph is empty then from the definition of labelled behaviour graph, there are no initial nodes, which means there does not exist a valuation V such that the right-hand sides of all initial and global clauses of T are true under V. Thus, the subset of T containing all initial and global clauses in T is unsatisfiable and by Lemma 17 there exists a refutation of T using step resolution rules SRES4, SRES5, and SRES8. $\square$

Theorem 5 (Completeness of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$) *If a set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is unsatisfiable, then T has a refutation using the resolution rules SRES1 to SRES8, ERES1 and ERES2 and the rewrite rules RW1 and RW2.*

Proof. Let T be an unsatisfiable set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. The proof proceeds by induction on the sequence of applications of the deletion rules to the labelled behaviour graph of T. If the unreduced labelled behaviour graph is empty then by Lemma 18 we can obtain a refutation by applying step resolution rules SRES4, SRES5 and SRES8.

Now suppose the labelled behaviour graph H is non-empty. The reduced labelled behaviour graph must be empty by Lemma 16, so there must be a node that can be deleted from H.

Suppose there is a node n which would be subject to the first deletion rule in Definition 14. Then n is a terminal node $n = (V, E_A, E_E)$. Consider $W = \{D \mid P \Rightarrow \mathbf{A}\bigcirc D \in T \text{ and } V \models P\} \cup \{D' \mid P' \Rightarrow \mathbf{E}\bigcirc D'_{\langle ind \rangle} \in T, ind \in \mathrm{Ind}(T), \text{ and } V \models P'\} \cup \{D'' \mid \mathbf{true} \Rightarrow D'' \in T\}$, where P, P'' are conjunctions of literals whereas D, D', D'' are disjunctions of literals. By Definition 6, W must be unsatisfiable.

Given that W is a set of propositional clauses, it has a refutation by propositional ordered resolution with selection using the same ordering and selection function as for $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. We prove that a clause $\mathbf{true} \Rightarrow \mathbf{false}$, $Q \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$ or $Q \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind \rangle}, ind \in \mathrm{Ind}(T)$, where Q is a conjunction of literals and satisfied by V, can be derived from T by SRES1 to SRES3 and SRES6 to SRES8. The proof proceeds by induction over the length of the propositional refutation of W. In particular, given a refutation $N_0, N_1, \ldots, N_n$ of W such that $N_0 = W$ and for every $i, 1 \leqslant i \leqslant n, N_i = N_{i-1} \cup \{C_i\}$, where C_i is a propositional clause derived from N_{i-1} and $C_n = \mathbf{false}$. We show that there exists a derivation $N'_0, N'_1, \ldots, N'_n$ such that $N'_0 = \{P \Rightarrow \mathbf{A}\bigcirc D \in T \text{ and } V \models P\} \cup \{P' \Rightarrow \mathbf{E}\bigcirc D'_{\langle ind \rangle} \in T, ind \in \mathrm{Ind}(T), \text{ and } V \models P'\} \cup \{\mathbf{true} \Rightarrow D'' \in T\}$ and for every $i, 1 \leqslant i \leqslant n, N'_i = N'_{i-1} \cup \{C'_i\}$, where C'_i is either $\mathbf{true} \Rightarrow C_i$, $P_i \Rightarrow \mathbf{A}\bigcirc C_i$ with $V \models P_i$ or $P_i \Rightarrow \mathbf{E}\bigcirc C_{i\langle ind \rangle}, ind \in \mathrm{Ind}(T)$ with $V \models P_i$. C'_n is either $\mathbf{true} \Rightarrow \mathbf{false}$, $P_n \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$ or $P_n \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind \rangle}, ind \in \mathrm{Ind}(T)$ with $V \models P_n$.

We prove the base case first. For the refutation of W, if $N_1 = W \cup \{C_1\}$, then we show $N'_1 = N'_0 \cup \{C'_1\}$, where C'_1 is the form of $\mathbf{true} \Rightarrow C_1$, $P_1 \Rightarrow \mathbf{A}\bigcirc C_1$ with $V \models P_1$ or $P_1 \Rightarrow \mathbf{E}\bigcirc C_{1\langle ind \rangle}, ind \in \mathrm{Ind}(T)$ with $V \models P_1$ and derived by an application of one of the resolution rules SRES1 to SRES3 and SRES6 to SRES8) from N'_0. Suppose $C_1 = C_2 \vee C_3$ is derived from two clauses $C_2 \vee l$ and $C_3 \vee \neg l$, then by the construction of W we are able to find a clause $G = P_0 \Rightarrow \mathbf{A}\bigcirc(C_2 \vee l)$, $P_0 \Rightarrow \mathbf{E}\bigcirc(C_2 \vee l)_{\langle ind \rangle}$ or $\mathbf{true} \Rightarrow C_2 \vee l$ and $G' = P_0 \Rightarrow \mathbf{A}\bigcirc(C_3 \vee \neg l)$, $P_0 \Rightarrow \mathbf{E}\bigcirc(C_3 \vee \neg l)_{\langle ind \rangle}$ or $\mathbf{true} \Rightarrow C_3 \vee \neg l$ in N'_0. Note that if G and G' are both $\mathbf{E}$-step clauses, then the indices ind in them are identical. Depending on the form of G and G' we can distinguish the following cases.

$$\begin{array}{rl} \text{From } G = & P_0 \Rightarrow \mathbf{A}\bigcirc C_2 \vee l \\ \text{and } G' = & P'_0 \Rightarrow \mathbf{A}\bigcirc C_3 \vee \neg l \\ \hline \text{we can derive } C'_1 = & P_0 \wedge P'_0 \Rightarrow \mathbf{A}\bigcirc C_2 \vee C_3 \text{ by SRES1} \end{array}$$

$$\begin{array}{rl} \text{From } G = & P_0 \Rightarrow \mathbf{E}\bigcirc C_2 \vee l_{\langle ind\rangle} \\ \text{and } G' = & P'_0 \Rightarrow \mathbf{A}\bigcirc C_3 \vee \neg l \\ \hline \text{we can derive } C'_1 = & P_0 \wedge P'_0 \Rightarrow \mathbf{E}\bigcirc C_2 \vee C_{3\langle ind\rangle} \text{ by SRES2} \end{array}$$

$$\begin{array}{rl} \text{From } G = & P_0 \Rightarrow \mathbf{A}\bigcirc C_2 \vee l \\ \text{and } G' = & P'_0 \Rightarrow \mathbf{E}\bigcirc C_3 \vee \neg l_{\langle ind\rangle} \\ \hline \text{we can derive } C'_1 = & P_0 \wedge P'_0 \Rightarrow \mathbf{E}\bigcirc C_2 \vee C_{3\langle ind\rangle} \text{ by SRES2} \end{array}$$

$$\begin{array}{rl} \text{From } G = & P_0 \Rightarrow \mathbf{E}\bigcirc C_2 \vee l_{\langle ind\rangle} \\ \text{and } G' = & P'_0 \Rightarrow \mathbf{E}\bigcirc C_3 \vee \neg l_{\langle ind\rangle} \\ \hline \text{we can derive } C'_1 = & P_0 \wedge P'_0 \Rightarrow \mathbf{E}\bigcirc C_2 \vee C_{3\langle ind\rangle} \text{ by SRES3} \end{array}$$

$$\begin{array}{rl} \text{From } G = & \mathbf{true} \Rightarrow C_2 \vee l \\ \text{and } G' = & P'_0 \Rightarrow \mathbf{A}\bigcirc C_3 \vee \neg l \\ \hline \text{we can derive } C'_1 = & P'_0 \Rightarrow \mathbf{A}\bigcirc C_2 \vee C_3 \text{ by SRES6} \end{array}$$

$$\begin{array}{rl} \text{From } G = & P_0 \Rightarrow \mathbf{A}\bigcirc C_2 \vee l \\ \text{and } G' = & \mathbf{true} \Rightarrow C_3 \vee \neg l \\ \hline \text{we can derive } C'_1 = & P_0 \Rightarrow \mathbf{A}\bigcirc C_2 \vee C_3 \text{ by SRES6} \end{array}$$

$$\begin{array}{rl} \text{From } G = & \mathbf{true} \Rightarrow C_2 \vee l \\ \text{and } G' = & P'_0 \Rightarrow \mathbf{E}\bigcirc C_3 \vee \neg l_{\langle ind\rangle} \\ \hline \text{we can derive } C'_1 = & P'_0 \Rightarrow \mathbf{E}\bigcirc C_2 \vee C_{3\langle ind\rangle} \text{ by SRES7} \end{array}$$

$$\begin{array}{rl} \text{From } G = & P_0 \Rightarrow \mathbf{E}\bigcirc C_2 \vee l_{\langle ind\rangle} \\ \text{and } G' = & \mathbf{true} \Rightarrow C_3 \vee \neg l \\ \hline \text{we can derive } C'_1 = & P_0 \Rightarrow \mathbf{E}\bigcirc C_2 \vee C_{3\langle ind\rangle} \text{ by SRES7} \end{array}$$

$$\begin{array}{rl} \text{From } G = & \mathbf{true} \Rightarrow C_2 \vee l \\ \text{and } G' = & \mathbf{true} \Rightarrow C_3 \vee \neg l \\ \hline \text{we can derive } C'_1 = & \mathbf{true} \Rightarrow C_2 \vee C_3 \text{ by SRES8} \end{array}$$

where l is eligible in $C_2 \vee l$ and $\neg l$ is eligible in $C_3 \vee \neg l$ for a given atom ordering and a given selection function S of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ as otherwise we would not have been able to derive $C_1 = C_2 \vee C_3$ on the propositional level using ordered resolution with selection given the ordering $\succ$ and the selection function S.

Because $C_1 = C_2 \vee C_3$, C_1' is one of the clauses $P_0 \wedge P_0' \Rightarrow \mathbf{A}\bigcirc C_1$, $P_0 \wedge P_0' \Rightarrow \mathbf{E}\bigcirc C_{1\langle ind\rangle}$, $P_0 \Rightarrow \mathbf{A}\bigcirc C_1$, $P_0' \Rightarrow \mathbf{A}\bigcirc C_1$, $P_0 \Rightarrow \mathbf{E}\bigcirc C_{1\langle ind\rangle}$, $P_0' \Rightarrow \mathbf{E}\bigcirc C_{1\langle ind\rangle}$, or $\mathbf{true} \Rightarrow C_2 \vee C_3$. It is easy to see that since $V \models P_0$ and $V \models P_0'$, we have $V \models P_1$. Further, in our cases above (SRES1 to SRES3 and SRES6 to SRES8) cover all the possibilities to derive C_1'. Thus, if there exists a derived clause C_1, then C_1' can derived by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$.

Next we prove the induction step. For the refutation $N_0, N_1, \ldots, N_i, N_{i+1}, \ldots, N_n$ of W, if $N_{i+1} = N_i \cup C_i$, then we show that $N_{i+1}' = N_i' \cup \{C_i'\}$, where C_i' is the form of $\mathbf{true} \Rightarrow C_i$, $P_i \Rightarrow \mathbf{A}\bigcirc C_i$ with $V \models P_i$ or $P_i \Rightarrow \mathbf{E}\bigcirc C_{i\langle ind\rangle}, ind \in \mathrm{Ind}(T)$ with $V \models P_i$ and derived by an application of one of the resolution rules SRES1 to SRES3 and SRES6 to SRES8 from N_i'. The proof proceeds in analogy to the base case.

Thus, we have shown that we can derive a clause $C_n' = P_n \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$, $P_n \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind\rangle}$ or $\mathbf{true} \Rightarrow \mathbf{false}$ from T. From $P_n \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$ or $P_n \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind\rangle}$ we can obtain the clause $\mathbf{true} \Rightarrow \neg P_n$ in normal form using RW1 or RW2.

By Lemma 14, the labelled behaviour graph H' for N_n' is a subgraph of H. In particular, every node in H' has to satisfy $\neg P_n$ or $\mathbf{false}$. Obviously, the node $n \in N$ does not satisfy this global clause and is thus not a node in H'.

Suppose the second (or third) deletion rule in Definition 14 is applicable to H. Then there must exist an eventuality $l_{\langle LC(ind)\rangle}$ (or l), where $l_{\langle LC(ind)\rangle}$ (or l) is not satisfied in an ind-labelled terminal subgraph for $l_{\langle LC(ind)\rangle}$ (or a terminal subgraph for l) of nodes ind-reachable (or reachable). We have two cases depending on the type of terminal subgraphs:

- ***ind*-labelled terminal subgraph for $l_{\langle LC(ind)\rangle}$.** Let $Q \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle}$ be a clause in T and $H' = (N', E')$ be an ind-labelled terminal subgraph for $l_{\langle LC(ind)\rangle}$ of the behaviour graph H. For each $n = (V, E_A, E_E) \in N'$, let $F \Rightarrow \mathbf{E}\bigcirc G_{\langle ind\rangle}$ be the conjunction of all global, **A**-step, **E**-step clauses labelled with ind in T whose left-hand sides are satisfied by V. To show this is a loop in $\neg l$, we must check the following two conditions.
  - For each $n \in N'$, we must have $\models G \Rightarrow \neg l$. By Definition 6 every valuation V' satisfying G must be a valuation of an ind-successor of the node n. By property (ITS2), all ind-successors are in H'. By property (ITS3), for every ind-successor $n' = (V', E_A', E_E')$ of the node n, $V' \models \neg l$, which implies $\models G \Rightarrow \neg l$. As n is arbitrary, for all $n \in N'$, we have $\models G \Rightarrow \neg l$.
  - For each $n \in N'$ we must have $\models G \Rightarrow \bigvee_{n \in N'} F$. Let $\{n_1, n_2, \ldots, n_k\}, k \geqslant 1$ be the set of ind-successors of the node n. Let $F_i \Rightarrow \mathbf{E}\bigcirc G_{i\langle ind\rangle}$ be the conjunction of all global, **A**-step and **E**-step clauses labelled with ind in T whose left-hand sides are satisfied by V_i, where V_i

is the valuation of the ind-successor $n_i = (V_i, E_{A_i}, E_{E_i})$ of n. By Definition 6 every valuation V' satisfying G must be a valuation of an ind-successor of the node n. Thus, for every $i, 1 \leqslant i \leqslant k, V_i \models G$. Since $V_i \models F_i$, we have $V_i \models G \Rightarrow F_i$. For all the valuation V_i, $\models G \Rightarrow F_1 \vee F_2 \vee \ldots \vee F_k$. By property (ITS2), all ind-successors of n are in H', we have $\{n_1, n_2, \ldots, n_k\} \subseteq N'$ and $\models G \Rightarrow \bigvee_{n \in N'} F$. As n is arbitrary, for all $n \in N'$, we have $\models G \Rightarrow \bigvee_{n \in N'} F$.

By the correctness of the loop search algorithm for CTL [6], we are able to use the set of conjunctions of $F \Rightarrow \mathbf{E}\bigcirc G_{\langle ind \rangle}$ in an application of ERES2 with the eventuality $l_{\langle LC(ind) \rangle}$ occurring in $Q \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind) \rangle} \in T$. Let L be defined as

$$L = \bigvee_{n \in N'} F$$

Then $T' = T \cup \{w_l^{ind} \Rightarrow \mathbf{E}\bigcirc(\neg L \vee l)_{\langle ind \rangle}, \mathbf{true} \Rightarrow \neg Q \vee \neg L \vee l\}$ is the result of adding the resolvents derived by ERES2 to T. Note that for every node $n = (V, E_A, E_E)$ in H', $V \models L, V \models \neg l$, and by Lemma 13, $V \models w_l^{ind}$. Recall that through augmentation the set T contains clauses

$$\begin{array}{rcl} w_l^{ind} & \Rightarrow & \mathbf{E}\bigcirc(l \vee w_l^{ind})_{\langle ind \rangle} \\ \mathbf{true} & \Rightarrow & (\neg Q \vee l \vee w_l^{ind}) \end{array}$$

By Lemma 12 either there is an edge $(n', ind, n) \in E$, where $n' = (V', E'_A, E'_E)$ such that $l_{\langle LC(ind) \rangle} \in E'_E, V' \models \neg l$ or $V \models Q, V \models \neg l$. Therefore we must have $V' \models w_l^{ind}$ by Lemma 13. So, for the aforementioned resolvent $w_l^{ind} \Rightarrow \mathbf{E}\bigcirc(\neg L \vee l)_{\langle ind \rangle}$, V' satisfies w_l^{ind} but V does not satisfy $(\neg L \vee l)$. Thus, the labelled behaviour graph for T' does not contain the edge (n', ind, n). Moreover, by Definition 6, the valuation of each predecessor of n must satisfy w_l^{ind}. Thus, all edges into n are not in the labelled behaviour graph for T'. Otherwise, for the latter, we have $V \models Q, V \models L, V \models \neg l$. Now, n does not satisfy the aforementioned resolvent $\mathbf{true} \Rightarrow \neg Q \vee \neg L \vee l$ in T' and so n is not a node in the labelled behaviour graph for T'. Therefore, the labelled behaviour graph for T' is a strict subgraph of that for T and by induction we assume that as T' has a refutation so must T.

- **Terminal subgraph for l.** The proof is analogous to the proof for ind-labelled terminal subgraphs for $l_{\langle LC(ind) \rangle}$.

□

Theorem 6 *Any derivation from a set T of SNF$^{\mathrm{g}}_{\mathrm{CTL}}$ clauses by the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ terminates.*

Proof. Let T be constructed from a set P of n atomic propositions and a set Ind of m indices. Then the number of SNF$^{\mathrm{g}}_{\mathrm{CTL}}$ clauses that can be constructed from P and Ind is finite. We can have at most 2^{2n} initial clauses, 2^{2n} global clauses, 2^{4n} **A**-step clauses, $m \cdot 2^{4n}$ **E**-step clauses, $n \cdot 2^{2n+1}$ **A**-sometime clauses, and $m \cdot n \cdot 2^{2n+1}$ **E**-sometime clauses. In total, there could be at most $(m+1)2^{4n} + (m \cdot n + n + 1)2^{2n+1}$

different $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses. Any derivation from a set of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses by the calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ will terminate when either no more new clauses can be derived or a contradiction is obtained. Since there is only a bounded number of different $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses, one of these two conditions will eventually be true. $\square$

Next we consider the complexity of $\mathsf{R}^{\succ,S}_{\text{CTL}}$.

Theorem 7 *The complexity of a $\mathsf{R}^{\succ,S}_{\text{CTL}}$ based decision procedure is in EXPTIME.*

Proof. Assume a set of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses is constructed from a set P of n propositions and a set Ind of m indices. The cost of deciding whether a step resolution rule can be applied to two determinate clauses is $\mathsf{A} = 4n + 1$ in the worst case, provided we can compute $S(C)$ in linear time and compute literal and indices, if they both have one, in constant time. From the proof of Theorem 6, we know the number of determinate clauses is at most $\mathsf{B} = 2^{2n} + 2^{2n} + 2^{4n} + m \cdot 2^{4n}$. Therefore, to naively compute a new clause from an application of some step resolution rule, we might need to look at $\mathsf{C} = \frac{\mathsf{B}(\mathsf{B}-1)}{2}$ pairs of two clauses and the associated cost is $(\mathsf{C} \cdot \mathsf{A})$. Moreover, to decide whether the resolvent is a new clause or not, we need to compare the resolvent with at most B clauses and the cost is $\mathsf{D} = \mathsf{B} \cdot (\mathsf{A} + 4n^2)$. In the worst case where each pair of clauses generates a resolvent but the resolvent already exists and only the last pair of clauses gives a new clause, to gain a new clause from an application of some step resolution rule, the complexity is of the order $(\mathsf{C} \cdot \mathsf{A} \cdot \mathsf{D})$, that is, EXPTIME.

To compute a new clause from an application of some eventuality resolution rule, the complexity depends on the complexity of the so-called CTL loop search algorithm which computes premises for the eventuality resolution rules [6]. The CTL loop search algorithm is a variation of the PLTL loop search algorithm [11] which has been shown to be in EXPTIME and we can show that the complexity of the CTL loop search algorithm from [6] is also in EXPTIME. Since a new clause is produced by an application of either step resolution or eventuality resolution, the complexity of generating a new clause is of the order EXPTIME. According to the proof of Theorem 6, there can be at most $(m+1)2^{4n} + (m \cdot n + n + 1)2^{2n+1}$ different $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses. Therefore, the complexity of saturating a set of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses and thereby deciding its satisfiability is in EXPTIME. $\square$

4 Implementation

To implement an efficient theorem prover for the calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ is a non-trivial job. Besides the effort to implement all the resolution rules, the effort to implement many techniques to prune search space for $\mathsf{R}^{\succ,S}_{\text{CTL}}$, for example reduction, subsumption and so on, are also required. Currently there are many state of the art first-order theorem provers, which have already implemented aforementioned techniques in many ways. Moreover, in [20] authors presented a method to bridge PLTL to first-order logic and used an existing first-order theorem prover to construct a prover for PLTL. The two facts above inspired us to adopt an approach to implement the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ and reuse those highly refined first-order prover to build our own prover for CTL.

First, we transform all $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses except **A**-sometime clauses and **E**-sometime clauses into first-order clauses. Then we are able to use first-order ordered resolution with selection to emulate step resolution. **A** and **E**-sometime clauses cannot be translated to first-order logic. Therefore, we continue to use the rules ERES1 and ERES2 for inferences with **A**- and **E**-sometime clauses, respectively, and use the loop search algorithm presented in Section 3.3 to find suitable premises for these rules. We utilise first-order ordered resolution with selection to perform the task of "looking for merged clauses" in the loop search algorithm and we compute the results of applications of the eventuality resolution rules in the form of first-order clauses.

4.1 Preliminaries of first-order ordered resolution with selection

We introduce several necessary notions before we start, following their definition in [4].

Let F, P and V be three pairwise disjoint (countable) sets. The elements of F are called *function symbols*, the elements of P *predicate symbols*, and the elements of V *variables.* Each element of F and P is associated with an *arity* $n \in \mathbb{N}_0$. The pair (F, P) is a *signature*.

A *term* is either a variable or an expression $f(t_1, \dots, t_n)$ where f is a function symbol of arity n and $t_1, \dots, t_n$ are terms. For a term $t = f(t_1, \dots, t_n)$ the terms $t_1, \dots, t_n$ are called the *arguments* of t. Let $\mathsf{T}(\mathsf{F}, \mathsf{V})$ denote the set of all terms built from function symbols in F and variables in V. A term is *ground* if it does not contain variables, i.e., it is an element of $\mathsf{T}(\mathsf{F}, \emptyset)$.

The *depth* $dp(t)$ of a term is inductively defined as (i) if t is a variable or a constant then $dp(t) = 1$, and (ii) if $t = f(t_1, \dots, t_n)$, then $dp(t) = 1 + \max(\{dp(t_i) \mid 1 \leq i \leq n\})$.

An *atom* is an expression $p(t_1, \dots, t_n)$ where $t_1, \dots, t_n$ are terms in $\mathsf{T}(\mathsf{F}, \mathsf{V})$ and p is a predicate symbol of arity n in P. A *literal* is an expression A (a positive literal) or $\neg A$ (a negative literal) where A is an atom. For a literal $L = (\neg)p(t_1, \dots, t_n)$ the terms $t_1, \dots, t_n$ are the *arguments* of L. A literal is *ground* if all its arguments are ground. The *depth* $dp(L)$ of a literal $L = (\neg)p(t_1, \dots, t_n)$ is given by $\max(\{dp(t_i) \mid 1 \leq i \leq n\})$ if the arity of p is greater than zero, and $dp(L) = 0$ otherwise.

A first-order *clause* $C = L_1 \vee \ldots \vee L_n$ is a multiset of literals with variables implicitly assumed to be universally quantified. A *subclause* D of a clause C is a sub-multiset D of C. A *strict subclause* D of a clause C is a subclause of C not identical to C. The *depth* $dp(C)$ of a clause C is given by $\max(\{dp(L) \mid L \in C\})$ if C is non-empty, and $dp(C) = 0$ otherwise.

We assume that the notions of a *substitution*, a *most general unifier*, an *instance*, etc, are defined in the usual way.

A *condensation* $\mathrm{Cond}(C)$ of a clause C is a minimal subclause of C which is also an instance of it. A clause C is *condensed* if there exists no condensation of C which is a strict subclause of C. Note that the condensation Cond(C) of a clause C is either identical to C or is a strict subclause of C which is a factor of C and which makes C redundant according to the definition of redundancy we give below.

An *atom ordering* $\succ_{FOL}$ is a well-founded, total ordering on ground atoms. For two non-ground atoms A and B we define $A \succ_{FOL} B$ if $A\sigma \succ_{FOL} B\sigma$ for all ground instances $A\sigma$ and $B\sigma$. As for the propositional case, the ordering $\succ_{FOL}$ is extended to literals by identifying each positive literal p with the singleton multiset $\{p\}$ and each negative literal $\neg p$ with the multiset $\{p, p\}$ and comparing such multisets of first-order atoms by using the multiset extension of $\succ_{FOL}$. Also, the notion of a (*strictly*) *maximal* literal with respect to a clause C is again defined as in the propositional case. Finally, the multiset extension of the literal ordering $\succ_{FOL}$ induces an ordering $\succ_{FOL}$ on ground clauses.

A *selection function* S_{FOL} assigns to each clause C a possibly empty set of occurrences of negative literals in C. If C is a clause, then the literals in $S_{FOL}(C)$ are called *selected*.

The resolution calculus $\mathsf{R}_{\text{FOL}}^{\succ_{FOL},S_{FOL}}$ is parameterised by an atom ordering $\succ_{FOL}$ and a selection function S_{FOL}, and consists of the following two inference rules:

- Ordered resolution with selection

$$\frac{C \vee A \qquad \neg B \vee D}{(C \vee D)\sigma}$$

  where

  1. σ is the most general unifier of A and B.
  2. No literal is selected in C and $A\sigma$ is strictly $\succ_{FOL}$-maximal with respect to $C\sigma$. (We say that A is *eligible* in $C \vee A$ for the substitution σ.)
  3. $\neg B$ is selected in $\neg B \vee D$ or no literal is selected in D and $\neg B\sigma$ is $\succ_{FOL}$-maximal with respect to $D\sigma$. (We say that $\neg B$ is *eligible* in $\neg B \vee D$ for the substitution σ.)

- Ordered positive factoring with selection

$$\frac{C \vee A \vee B}{(C \vee A)\sigma}$$

  where

  1. σ is the most general unifier of A and B.
  2. No literal is selected in C and $A\sigma$ is $\succ_{FOL}$-maximal with respect to $C\sigma$.

Given a set N of clauses and a clause C, then C is *redundant in* N if for all ground substitutions σ there exist clauses $C_1, \ldots, C_n$, $n \geq 0$, in N and ground substitutions σ_j such that $C_1\sigma_1, \ldots, C_n\sigma_n \models C\sigma$ and $C\sigma \succ_{FOL} C_i\sigma_i$ for every i, $1 \leq i \leq n$.

A set N is *saturated (up to redundancy)* with respect to $\mathsf{R}_{\text{FOL}}^{\succ_{FOL},S_{FOL}}$ if all clauses that can be derived by an application of the rules of $\mathsf{R}_{\text{FOL}}^{\succ_{FOL},S_{FOL}}$ to non-redundant premises in N are either contained in N

or else are redundant in N.

For a set N_0 of clauses, a *derivation* from N_0 is a sequence of clause sets $N_0, N_1, \ldots$, where for every i, $i \geq 0$, $N_{i+1} = N_i \cup \{C\}$ and C is derived by applying a $\mathsf{R}_{\mathrm{FOL}}^{\succ_{FOL}, S_{FOL}}$ inference rule to premises in N_i, or $N_{i+1} = N_i \setminus \{C\}$ and C is redundant in N_i. A derivation $N_0, N_1, \ldots$ is a *refutation (of N_0)* if for some i, $0 \leq i$, N_i contains the empty clause. We say a derivation $N_0, N_1, \ldots$ from N_0 *terminates* if for some i, $0 \leq i$, N_i is saturated up to redundancy. A derivation $N_0, N_1, \ldots$ is *fair* if every clause C that can be deduced from non-redundant premises in the *limit* $N_\infty = \bigcup_j \bigcap_{k \leq j} N_k$ is contained in some set N_j.

$\mathsf{R}_{\mathrm{FOL}}^{\succ_{FOL}, S_{FOL}}$ is a sound and complete refutation calculus [4]: for a set of clauses N_0 and fair derivation $N_0, N_1, \ldots$ from N_0, N_0 is unsatisfiable iff the clause set $\bigcup_j N_j$ contains the empty clause. Furthermore, if for some $i \geq 0$, N_i is saturated up to redundancy, then N_0 is unsatisfiable iff N_i contains the empty clause.

4.2 Representing determinate $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses as first-order clauses

In order to represent every determinate $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clause by a first-order clause we uniquely associate every propositional variable p with a unary predicate symbol Q_p. Besides these predicate symbols we assume that our first-order vocabulary includes a countably infinite set of variables x, y, $\ldots$, a constant $\mathsf{0}$, a binary function symbol app, and for every $ind \in \mathsf{Ind}$ a constant s_{ind}.

The constant 0 represents the state $s_0 \in S$ in an extended model structure $M = \langle S, R, L, [\_], s_0 \rangle$, while a constant s_{ind} represents $[ind]$. The variables x and y quantify over states in M, and the variable s quantifies over the successor functions $[ind]$ with $ind \in \mathsf{Ind}$.

The first-order atom $Q_p(x)$, represents that p holds at a state x, while $Q_p(0)$ represents that p holds at the state s_0.

The first-order term $\mathsf{app}(s, x)$ represents the state resulting from the application of the successor function s to the state x, while $\mathsf{app}(s_{ind}, x)$ represents the state resulting from the application of the successor function $[ind]$ to the state x. Then the first-order atoms $Q_p(\mathsf{app}(s, x))$ and $Q_p(\mathsf{app}(s_{ind}, x))$ represent that p holds at states $\mathsf{app}(s, x)$ and $\mathsf{app}(s_{ind}, x)$, respectively.

Finally, for a disjunction of propositional literals $C = (\neg)p_0 \vee \ldots \vee (\neg)p_n$, let $\lceil C \rceil(t)$, where t is term, denote the first-order clause $(\neg)Q_{p_0}(t) \vee \ldots \vee (\neg)Q_{p_n}(t)$.

We are then able to represent every initial, global, **A**-step and **E**- step clause Γ by a first-order clause $\lceil \Gamma \rceil$ as follows:

1. An initial clause **start** $\Rightarrow C$ is represented by $\lceil C \rceil(\mathsf{0})$

2. A global clause **true** $\Rightarrow C$ is represented by $\lceil C \rceil(x)$

3. An **A**-step clause $P \Rightarrow \mathbf{A}\bigcirc C$ is represented by $\lceil \neg P \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s, x))$

4. An **E**-step clause $P \Rightarrow \mathbf{E}\bigcirc C_{\langle ind \rangle}$ is represented by $\lceil \neg P \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s_{ind}, x))$

Note that it is possible for C to be empty in an **A**-step clause $P \Rightarrow \mathbf{A}\bigcirc C$ or an **E**-step clause $P \Rightarrow \mathbf{E}\bigcirc C_{\langle ind\rangle}$. In the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, such clauses are subject to the rewrite rules RW1 and RW2, and would both be replaced by the global clause **true** $\Rightarrow \neg P$. We see that in our first-order representation of determinate $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses, for an empty clause C, the $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses $P \Rightarrow \mathbf{A}\bigcirc C$, $P \Rightarrow \mathbf{E}\bigcirc C_{\langle ind\rangle}$, and **true** $\Rightarrow \neg P$ all have the same representation, namely, $\lceil \neg P\rceil(x)$. Thus, on the first-order level, the rewrite rules RW1 and RW2 are superfluous.

4.3 Implementing step resolution

The representation of determinate $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$[4] clauses by first-order clauses allows for all our step resolution rules to be implemented using the ordered first-order resolution with selection calculus presented in Section 4.1.

To this end, we have to define the atom ordering and selection function on the first-order level in such a way that they mirror their definition on the propositional level.

Regarding the first-order atom ordering, note that our signature only contains unary predicate symbols. Let $\succ$ be a propositional atom ordering. Then we allow $\succ_{FOL}$ to be any ground first-order atom ordering such that $Q_q(s) \succ_{FOL} Q_p(t)$ if

(i) $dp(s) > dp(t)$; or

(ii) $dp(s) = dp(t)$ and $q \succ p$.

According to this definition, lifted to the non-ground level, we have

$$Q_w(\mathsf{app}(s_{ind'}\mathsf{app}(s_{ind}, x),)) \succ_{FOL} Q_p(\mathsf{app}(s_{ind}, x)) \succ_{FOL} Q_q(x),$$

for any predicate symbols Q_p, Q_q, Q_w, and constants $s_{ind'}$ and s_{ind}, and

$$Q_q(\mathsf{app}(s_{ind}, x)) \succ_{FOL} Q_p(\mathsf{app}(s_{ind}, x))Q_q(x) \succ_{FOL} Q_p(x)$$

provided $q \succ p$.

Regarding the first-order selection function S_{FOL}, we use the close correspondence between determine $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and their first-order representation to define S_{FOL} as follows:

1. A literal$\lceil \neg l\rceil(0)$ is selected in $\lceil C\rceil(0)$ by S_{FOL} iff $\neg l$ is selected by S in C;
2. A literal $\lceil \neg l\rceil(x)$ is selected in $\lceil C\rceil(x)$ by S_{FOL} iff $\neg l$ is selected in C by S;
3. A literal $\lceil \neg l\rceil(\mathsf{app}(s, x))$ is selected in[4] $\lceil \neg P\rceil(x) \vee \lceil C\rceil(\mathsf{app}(s, x))$ by S_{FOL}, with C being non-empty, iff $\neg l$ is selected in C by S;

An application of SRES2

$$\frac{\begin{array}{c} P \Rightarrow \mathbf{E}\bigcirc(C \vee l)_{\langle ind \rangle} \\ Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l) \end{array}}{P \wedge Q \Rightarrow \mathbf{E}\bigcirc(C \vee D)_{\langle ind \rangle}} \qquad \begin{array}{l} \text{where } l \text{ is eligible in } C \vee l \\ \text{and } \neg l \text{ is eligible in } D \vee \neg l \end{array}$$

can be emulated by the following inference using ordered resolution with selection

$$\frac{\begin{array}{c} \lceil \neg P \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s_{ind}, x)) \vee Q_l(\mathsf{app}(s_{ind}, x)) \\ \lceil \neg Q \rceil(y) \vee \lceil D \rceil(\mathsf{app}(z, y)) \vee \neg Q_l(\mathsf{app}(z, y)) \end{array}}{\mathrm{Cond}(\lceil \neg P \rceil(x) \vee \lceil \neg Q \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s_{ind}, x)) \vee \lceil D \rceil(\mathsf{app}(s_{ind}, x)))}$$

where the definition of $\succ_{FOL}$ and S_{FOL} ensures that $Q_l(\mathsf{app}(s_{ind}, x))$ and $\neg Q_l(\mathsf{app}(z, y))$ are both eligible in their respective clauses for the substitution σ.

Figure 7: Emulating SRES2 inferences in first-order logic

An application of SRES3

$$\frac{\begin{array}{c} P \Rightarrow \mathbf{E}\bigcirc(C \vee l)_{\langle ind \rangle} \\ Q \Rightarrow \mathbf{E}\bigcirc(D \vee \neg l)_{\langle ind \rangle} \end{array}}{P \wedge Q \Rightarrow \mathbf{E}\bigcirc(C \vee D)_{\langle ind \rangle}} \qquad \begin{array}{l} \text{where } l \text{ is eligible in } C \vee l \\ \text{and } \neg l \text{ is eligible in } D \vee \neg l \end{array}$$

can be emulated by the following inference using ordered resolution with selection

$$\frac{\begin{array}{c} \lceil \neg P \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s_{ind}, x)) \vee Q_l(\mathsf{app}(s_{ind}, x)) \\ \lceil \neg Q \rceil(y) \vee \lceil D \rceil(\mathsf{app}(s_{ind}, y)) \vee \neg Q_l(\mathsf{app}(s_{ind}, y)) \end{array}}{\mathrm{Cond}(\lceil \neg P \rceil(x) \vee \lceil \neg Q \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s_{ind}, x)) \vee \lceil D \rceil(\mathsf{app}(s_{ind}, x)))}$$

where the definition of $\succ_{FOL}$ and S_{FOL} ensures that $Q_l(\mathsf{app}(s_{ind}, x))$ and $\neg Q_l(\mathsf{app}(s_{ind}, y))$ are both eligible in their respective clauses for the substitution σ. Note that $Q_l(\mathsf{app}(s_{ind}, x))$ and $Q_l(\mathsf{app}(s_{ind'}, y))$ are only unifiable if $ind = ind'$.

Figure 8: Emulating SRES3 in first-order logic

4. A literal $\lceil \neg l \rceil(\mathsf{app}(s_{ind}, x))$ is selected in $\lceil \neg P \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s_{ind}, x))$ by S_{FOL}, with C being non-empty, iff $\neg l$ is selected in C by S.

There is one more complication that we need to overcome. In the case of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses, we have made the simplifying assumption that the conjunctions and disjunctions of propositional literals occurring in these clauses do not contain duplicate occurrences of the same literals, thus avoiding the need for factoring inference rules in our calculus. For example, resolving two global clauses $\Gamma_1 = \mathbf{true} \Rightarrow \neg p \vee q$ and $\Gamma_2 = \mathbf{true} \Rightarrow p \vee q$ simply results in $\Gamma_3 = \mathbf{true} \Rightarrow q$. However, for first-order clauses we have followed the definition of clauses as multisets of first-order literals. Thus, resolving $\lceil \Gamma_1 \rceil = \neg Q_p(x) \vee Q_q(x)$ with $\lceil \Gamma_2 \rceil = Q_p(x) \vee Q_q(x)$ results in $Q_q(x) \vee Q_q(x)$, which is not identical to $\lceil \Gamma_3 \rceil = Q_q(x)$.

To eliminate this mismatch, we require that on the first-order level all clauses are kept in condensed form, that is, every first-order clauses C is replaced by its condensation $\mathrm{Cond}(C)$. In our example, we have $\mathrm{Cond}(Q_q(x) \vee Q_q(x)) = Q_q(x) = \lceil \Gamma_3 \rceil$.

We are then able to establish the following correspondence between $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ inferences on determinate $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ inferences on their first-order representation.

An application of SRES7

$$\begin{array}{c} \mathbf{true} \Rightarrow C \vee l \\ Q \Rightarrow \mathbf{E}\bigcirc(D \vee \neg l)_{\langle ind \rangle} \\ \hline Q \Rightarrow \mathbf{E}\bigcirc(C \vee D)_{\langle ind \rangle} \end{array} \qquad \begin{array}{l} \text{where } l \text{ is eligible in } C \vee l \\ \text{and } \neg l \text{ is eligible in } D \vee \neg l \end{array}$$

can be emulated by the following inference using ordered resolution with selection

$$\begin{array}{l} \lceil C \rceil(x) \vee Q_l(x) \\ \lceil \neg Q \rceil(y) \vee \lceil D \rceil(\mathsf{app}(s_{ind}, y)) \vee \neg Q_l(\mathsf{app}(s_{ind}, y)) \\ \hline \text{Cond}(\lceil \neg Q \rceil(y) \vee \lceil C \rceil(\mathsf{app}(s_{ind}, y)) \vee \lceil D \rceil(\mathsf{app}(s_{ind}, y))) \end{array}$$

where the definition of $\succ_{FOL}$ and S_{FOL} ensures that $Q_l(x)$ and $\neg Q_l(\mathsf{app}(s_{ind}, y))$ are both eligible in their respective clauses for the substitution σ.

Figure 9: Emulating SRES7 in first-order logic

Theorem 8 *Let $\succ$ and S be an atom ordering and selection function, respectively, for $\mathsf{R}^{\succ,S}_{\text{CTL}}$ and let $\succ_{FOL}$ and S_{FOL} be a corresponding atom ordering and a corresponding selection function, respectively, for $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\text{FOL}}$. Let Γ_1 and Γ_2 be two determinate $\text{SNF}^{\text{g}}_{\text{CTL}}$. Then a determinate clause Γ_3 is derivable from Γ_1 and Γ_2 by SRES1 to SRES8 in $\mathsf{R}^{\succ,S}_{\text{CTL}}$ iff there exists a clause C derivable from $\lceil \Gamma_1 \rceil$ and $\lceil \Gamma_2 \rceil$ by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\text{FOL}}$ such that $\lceil \Gamma_3 \rceil$ is a condensation of C.*

Proof. Our definition of S_{FOL} and $\succ_{FOL}$ ensures that there is a one-to-one correspondence between eligible literals in determinate $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses and eligible literals in the first-order representation of these clauses.

Therefore, we can show that for any inference by SRES1 to SRES8 there is a corresponding inference by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\text{FOL}}$. Figures 7, 8 and 9 show this relationship for inferences by SRES2, SRES3 and SRES7. The relationship for the inferences by the remainder of step resolution rules is analogous.

Now, we show that, provided we keep first-order clauses in condensed form, $\mathsf{R}^{\succ,S}_{\text{CTL}}$ does not allow additional inferences which do not have a correspondence by SRES1 to SRES8.

As we know a determinate clause is either an initial C^i, a global C^g, an **A**-step C^A or an **E**-step clause C^E, we can distinguish ten different types of inference, $C^iCg, C^iC^A, C^iC^E, C^gC^A, C^gC^E, C^AC^E$, C^iC^i, C^gC^g, C^AC^A and C^EC^E. SRES1 to SRES8 maps themselves to eight types except C^iC^A and C^iC^E, which implies that it is not allow to have inferences between an initial clause and an **A**-step clause or an **E**-step clause.

For the type of inference of C^iC^A, consider the four cases:

Case 1

$$\begin{array}{rlll} & (Q_q(0) & \vee \ldots) & \text{is transformed from an initial clause} \\ (\ldots \vee & \neg Q_q(\mathsf{app}(s, x)) & \vee \ldots) & \text{is transformed from an } \mathbf{A}\text{-step clause} \end{array}$$

$Q_q(0)$ and $\neg Q_q(\mathsf{app}(s, x))$ do not unify;

Case 2

$$\begin{array}{rlr}(Q_q(0) & & \vee \ldots) \quad \text{is transformed from an initial clause}\\ (\ldots \vee & \neg Q_q(x) \vee (\neg)Q_p(\mathsf{app}(s,x)) & \vee \ldots) \quad \text{is transformed from an } \mathbf{A}\text{-step clause}\end{array}$$

$\neg Q_q(x)$ is not eligible in this clause: Condition (i) in the definition of $\succ_{FOL}$ ensures that $(\neg)Q_p(\mathsf{app}(s,x))\sigma \succ_{FOL} Q_q(x)\sigma$ for any substitution σ and S_{FOL} is defined in such a way that $\neg Q_q(x)$ is not selected.

Case 3

$$\begin{array}{rlr}(& \neg Q_q(0) & \vee \ldots) \quad \text{is transformed from an initial clause}\\ (\ldots \vee & Q_q(\mathsf{app}(s,x)) & \vee \ldots) \quad \text{is transformed from an } \mathbf{A}\text{-step clause}\end{array}$$

$\neg Q_q(0)$ and $Q_q(\mathsf{app}(s,x))$ do not unify;

Case 4

$$\begin{array}{rlr}(& \neg Q_q(0) & \vee \ldots) \quad \text{is transformed from an initial clause}\\ (\ldots \vee & Q_q(x) \vee (\neg)Q_p(\mathsf{app}(s,x)) & \vee \ldots) \quad \text{is transformed from an } \mathbf{A}\text{-step clause}\end{array}$$

$Q_q(x)$ is not eligible in this clause: Condition (i) in the definition of $\succ_{FOL}$ ensures that $(\neg)Q_p(\mathsf{app}(s,x))\sigma \succ_{FOL} Q_q(x)\sigma$ for any substitution σ and S_{FOL} only select negative literals.

For inference of C^iC^E, the proof is analogous. Therefore, $\mathsf{R}_{\text{FOL}}^{\succ_{FOL},S_{FOL}}$ disallows types of inference of C^iC^A and C^iC^E.

Further, all inferences by SRES1 to SRES8 involve only literals on the right-hand sides of determinate clauses. The we prove inferences of $\mathsf{R}_{\text{FOL}}^{\succ_{FOL},S_{FOL}}$ do so as well. It is again the atom ordering $\succ_{FOL}$ and the selection function S_{FOL} which guarantee that requirement.

For example,

Case 5

$$\begin{array}{rlr}(& \neg Q_q(x) \vee (\neg)Q_{q_1}(\mathsf{app}(s,x)) & \vee \ldots) \quad \text{is transformed from an } \mathbf{A}\text{-step clause}\\ (\ldots \vee & Q_q(x) \vee (\neg)Q_{q_2}(\mathsf{app}(s,x)) & \vee \ldots) \quad \text{is transformed from an } \mathbf{A}\text{-step clause}\end{array}$$

$Q_q(x)$ is not eligible in this clause: Condition (i) in the definition of $\succ_{FOL}$ ensures that $(\neg)Q_{q_2}(\mathsf{app}(s,x))\sigma \succ_{FOL} Q_q(x)\sigma$ for any substitution σ and S_{FOL} only select negative literals.

Case 6

$$\begin{array}{rlr}(& \neg Q_q(x) \vee (\neg)Q_{q_1}(\mathsf{app}(s_{ind},x)) & \vee \ldots) \quad \text{is transformed from an } \mathbf{E}\text{-step clause}\\ (\ldots \vee & Q_q(x) \vee (\neg)Q_{q_2}(\mathsf{app}(s_{ind},x)) & \vee \ldots) \quad \text{is transformed from an } \mathbf{E}\text{-step clause}\end{array}$$

$Q_q(x)$ is not eligible in this clause: Condition (i) in the definition of $\succ_{FOL}$ ensures that $(\neg)Q_{q_2}(\mathsf{app}(s,x))\sigma \succ_{FOL} Q_q(x)\sigma$ for any substitution σ and S_{FOL} only select negative literals.

```
 1 procedure $eres$(T, C)
 2 // T is a saturated set of determinate clauses
 3 // C is a sometime clause $Q \Rightarrow \mathbf{A}\Diamond\neg l$ or $Q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle}$
 4 begin
 5     if C is an $\mathbf{A}$-sometime clause then
 6         SOS := $\{$D $|$ D is a global or step clause in T$\}$;
 7     else if C is an $\mathbf{E}$-sometime clause then
 8         SOS := $\{$D $|$ D is a global, $\mathbf{A}$-step, or $\mathbf{E}$-step clause with the index $ind$ in T$\}$;
 9     end if
10     i := 0;
11     $\mathtt{H}_{-1}$(x) := $\mathbf{true}$;
12     do
13         Goals := $\{ls(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee \neg \mathtt{H}_{\mathtt{i}-1}(x)\sigma\}$, where $\sigma = \{x \leftarrow \mathsf{app}(s,x)\}$;
14         $\mathtt{T}_1$ := $resolution\_sos$(SOS, Goals);
15         $\mathtt{T}_2$ := $\{G(x) \mid G(x) \vee ls(x) \in \mathtt{T}_1$ and $depth(G(x)) \leq 1\}$;
16         $\mathtt{H}_\mathtt{i}$(x) := $\neg(\bigwedge \mathtt{T}_2)$;
17         if $\mathtt{H}_\mathtt{i}(x)$ is equivalent to true then
18             return $eresolvent$(C, $\mathbf{true}$);
19         else if $\mathtt{H}_\mathtt{i}(x)$ is equivalent to false then
20             return $\emptyset$
21         else if $\mathtt{H}_\mathtt{i}$(x) is equivalent to $\mathtt{H}_{\mathtt{i}-1}$(x) then
22             return $eresolvent$(C, $\mathtt{H}_\mathtt{i}$(x));
23         end if
24         i := i+1;
25     while ($\mathtt{T}_2 \neq \emptyset$)
26 end
```

Figure 10: *eres*: A loop search implementation using first-order resolution

Proofs for other situations are analogous. Therefore, $\mathsf{R}^{\succ_{FOL}, S_{FOL}}_{\mathrm{FOL}}$ does not allow an inference resolving literals from left-hand sides of determinate clauses. Finally, condensation ensures that the ordered factoring with selection rule of $\mathsf{R}^{\succ_{FOL}, S_{FOL}}_{\mathrm{FOL}}$ is not applicable.

This establishes the desired one-to-one correspondence between inferences by SRES1 to SRES8 and inferences by $\mathsf{R}^{\succ_{FOL}, S_{FOL}}_{\mathrm{FOL}}$. $\square$

4.4 Implementing eventuality resolution

To implement the eventuality resolution rules ERES1 and ERES2, we will need to augment a first-order theorem prover with an implementation of the **E**-loop search algorithm defined in Section 3.3. Figure 10 shows the pseudocode for the implementation of this algorithm in our prover CTL-RP.

The procedure *eres* takes as input a set `T` of determinate clauses, which we assume to be saturated under the step resolution rules SRES1 to SRES8 and the rewrite rules RW1 and RW2, and a **A**-sometime clause or **E**-sometime clause `C`. As stated in Section 3.3, if C is an **A**-sometime clause $Q \Rightarrow \mathbf{A}\Diamond\neg l$, then the loop search algorithm considers all global, **A**-step clauses, and **E**-step clauses in T, while if C is an **E**-sometime clause $Q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle}$, then the loop search we try to resolve an **E**-sometime clause algorithm considers all global, **A**-step clauses, all **E**-step clauses with index *ind* in T. Lines 5 to 9 of our algorithm implement this case distinction and store the set of clauses that needs to be considered

```
procedure $eresolvent$(C, H$_i$(x))
// C is a sometime clause $Q \Rightarrow \mathbf{A}\Diamond\neg l$ or $Q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle}$
// H$_i(x)$ = $\neg \bigwedge_{i=1}^{n} G_i(x)$ is a loop formula
begin
    if H$_i$(x) = $\mathbf{true}$ then
       $G_i$(x) := $\mathbf{false}$;
    end if
    if C is an $\mathbf{A}$-sometime clause then
      resolvents := $\{\lceil\neg Q\rceil(x) \vee \neg Q_l(x) \vee G_i(x) \mid 1 \leq i \leq n\} \cup$
                    $\{\neg Q_{w^{\mathtt{A}}_{\neg l}}(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee G_i(x)\sigma \mid 1 \leq i \leq n,\ \sigma = \{x \leftarrow \mathsf{app}(s,x)\}\} \cup$
                    $\{\lceil\neg Q\rceil(x) \vee \neg Q_l(x) \vee Q_{w^{\mathtt{A}}_{\neg l}}(x),$
                     $\neg Q_{w^{\mathtt{A}}_{\neg l}}(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee Q_{w^{\mathtt{A}}_{\neg l}}(\mathsf{app}(s,x))\}$;
    else if C is an $\mathbf{E}$-sometime clause then
      resolvents := $\{\lceil\neg Q\rceil(x) \vee \neg Q_l(x) \vee G_i(x) \mid 1 \leq i \leq n\} \cup$
                    $\{\neg Q_{w^{ind}_{\neg l}}(x) \vee \neg Q_l(\mathsf{app}(s_{ind},x)) \vee G_i(x)\sigma \mid 1 \leq i \leq n,$
                                                   $\sigma = \{x \leftarrow \mathsf{app}(s_{ind},x)\}\} \cup$
                    $\{\lceil\neg Q\rceil(x) \vee \neg Q_l(x) \vee Q_{w^{ind}_{\neg l}}(x),$
                     $\neg Q_{w^{ind}_{\neg l}}(x) \vee \neg Q_l(\mathsf{app}(s_{ind},x)) \vee Q_{w^{ind}_{\neg l}}(\mathsf{app}(s_{ind},x))\}$;
    end if
    return resolvents;
end
```

Figure 11: The *eresolvent* procedure

in a set SOS. The main part of the algorithm, lines 12 to 25, consists of a loop in which we construct a sequence of formulae $\mathtt{H_{-1}(x)}$, $\mathtt{H_0(x)}$, $\mathtt{H_1(x)}$, ... until one of the three termination conditions is satisfied: (a) if $\mathtt{H_i(x)}$ is equivalent to true, then we use the procedure *eresolvent* to return the resolvents for C and the loop true (lines 17 and 18); (b) if $\mathtt{H_i(x)}$ is equivalent to false, then no loop can be found and we return the empty set of resolvents (lines 19 and 20); (c) if $\mathtt{H_i(x)}$ is equivalent to $\mathtt{H_{i-1}(x)}$, then we again use the procedure *eresolvent* to return the resolvents for C and the loop $\mathtt{H_i(x)}$ (line 21 and 22). Line 13 to 16 deal with the construction of the formula $\mathtt{H_i(x)}$ for the current index i. Recall from Section 3.3 that to construct H_i, we need to look for merged clauses $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}\bigcirc(B_j \wedge l)_{\langle ind\rangle}$ such that $B_j \Rightarrow H_{i-1}$ (or, equivalently, $\mathbf{A}\bigcirc B_j \Rightarrow \mathbf{A}\bigcirc H_{i-1}$). To do so, we construct a set of goal clauses Goals with each clause containing the literal $\neg\mathtt{Q_l}(\mathsf{app}(\mathtt{s},\mathtt{x}))$, the first-order representation of $\mathbf{A}\bigcirc\neg l$ and a disjunct from $\neg\mathtt{H_{i-1}}(\mathsf{app}(s,x))$, the first-order representation of $\mathbf{A}\bigcirc H_{i-1}$. When trying to prove these goal clauses using the clauses in SOS, all newly derived clauses of depth one or less would be the first-order representations of the A_j's that we look for. To make it easier to identify newly derived clauses, we add a literal $ls(x)$, where ls is a new unary predicate symbol, to each of the goal clause. As there are no negative occurrences of $ls(x)$ in SOS, $ls(x)$ occurs in all clauses derived from our goal clauses. In Figure 10, line 13 constructs the goal clauses, line 14 calls the *resolution_sos* procedure to saturate SOS $\cup$ Goals using a set of support strategy, line 15 collects all newly derived clauses of depth one or less from the saturated set using the literal $ls(x)$ to identify newly derived clauses, and, finally, line 16 computes $\mathtt{H_i(x)}$.

The following example illustrates how our implementation of the loop search algorithm works. The set T consists of the three $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses $a \Rightarrow \mathbf{A}\bigcirc l$, $b \Rightarrow \mathbf{A}\bigcirc l$, and $a \Rightarrow \bigcirc a_{\langle ind\rangle}$ and we are looking for a loop in $\neg l$. The first-order representation of these clauses is given by

(1) $\neg Q_a(x) \vee Q_l(\mathsf{app}(s,x))$

(2) $\neg Q_b(x) \vee Q_l(\mathsf{app}(s,x)))$

(3) $\neg Q_a(x) \vee Q_a(\mathsf{app}(s_{ind},x)))$

For our atom ordering we use a lexicographic path ordering based on the precedence $app > Q_l > Q_a > Q_b > s_{ind} > ls$ and a selection function which returns the empty set for every clause, i.e. no literals are selected in any clause.

In the following description of resolution derivations, $[G]$ indicates a goal clause that has been added to T, $[n, R, m]$ indicates a resolvent of the clauses labelled (n) and (m), and $[n, C]$ indicates the condensation of the clause labelled (n).

During the first iteration of the main loop of *eres*, the set of goal clauses consists of the single clause $ls(x) \vee \neg Q_l(\mathsf{app}(s,x))$ and *resolution_sos* conducts the following inferences:

[G] (4) $ls(x) \vee \neg Q_l(\mathsf{app}(s,x))$

[1, R, 4] (5) $ls(x) \vee \neg Q_a(x)$

[2, R, 4] (6) $ls(x) \vee \neg Q_b(x)$

[3, R, 5] (7) $ls(\mathsf{app}(s_{ind},x)) \vee \neg Q_a(x)$

Of these clauses, only clauses (5) and (6) contribute to the construction of $H_0(x)$ (see lines 15 and 16 of the *eres*) and we obtain $H_0(x) = Q_a(x) \vee Q_b(x)$. As $H_0(x)$ does not satisfy any of the three termination conditions, the main loop of *eres* will be executed a second time. This time, we have two goal clauses, clauses (8) and (9) below:

[G] (8) $ls(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee \neg Q_a(\mathsf{app}(s,x))$

[G] (9) $ls(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee \neg Q_b(\mathsf{app}(s,x))$

[1, R, 8] (10) $ls(x) \vee \neg Q_a(x) \vee \neg Q_a(\mathsf{app}(s,x))$

[1, R, 9] (11) $ls(x) \vee \neg Q_a(x) \vee \neg Q_b(\mathsf{app}(s,x))$

[2, R, 8] (12) $ls(x) \vee \neg Q_b(x) \vee \neg Q_a(\mathsf{app}(s,x))$

[2, R, 9] (13) $ls(x) \vee \neg Q_b(x) \vee \neg Q_b(\mathsf{app}(s,x))$

[3, R, 10] (14) $ls(x) \vee \neg Q_a(x) \vee \neg Q_a(x)$

[14, C] (15) $ls(x) \vee \neg Q_a(x)$

[3, R, 12] (16) $ls(x) \vee \neg Q_b(x) \vee \neg Q_a(x)$

[3, R, 15] (17) $ls(\mathsf{app}(s_{ind},x)) \vee \neg Q_a(x)$

As the condensed clause (15) makes clause (14) redundant and clause (15) also subsumes clause (16), of all the clauses in the saturated set, only clause (15) contributes to the construction of $H_1(x)$ and we

```
procedure resolution_prover(N)
begin
    Wo := $\emptyset$; Us := taut(sub(N));
    while (Us $\neq \emptyset$ and $\perp \notin$ Us)
        (Given,Us) := choose(Us);
        Wo  := Wo $\cup$ {Given};
        New := res(Given,Wo) $\cup$ fac(Given);
        New := taut(sub(New));
        New := sub(sub(New,Wo),Us);
        Wo  := sub(Wo,New);
        Us  := sub(Us,New) $\cup$ New;
    end
    output();
end
```

Figure 12: A simple resolution prover [26]

obtain $H_1(x) = Q_a(x)$. Again, $H_1(x)$ does not satisfy any of the three termination conditions, and a third iteration of the main loop is *eres* is required. There is only one goal clause, clause (18).

[G] (18) $ls(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee \neg Q_a(\mathsf{app}(s,x))$

[1, R, 18] (19) $ls(x) \vee \neg Q_a(x) \vee \neg Q_a(\mathsf{app}(s,x))$

[2, R, 18] (20) $ls(x) \vee \neg Q_b(x) \vee \neg Q_a(\mathsf{app}(s,x))$

[3, R, 19] (21) $ls(x) \vee \neg Q_a(x) \vee \neg Q_a(x)$

[21, C] (22) $ls(x) \vee \neg Q_a(x)$

[3, R, 22] (23) $ls(\mathsf{app}(s_{ind},x)) \vee \neg Q_a(x)$

Again, the condensed clause (22) makes clause (21) redundant and only clause (22) remains to contribute to the construction of $H_2(x)$. We obtain $H_2(x) = Q_a(x)$ which is equivalent to $H_1(x)$. Thus, the third termination condition of *eres* is satisfied (line 21) and the *eresolvent* procedure, shown in Figure 11, will return the appropriate resolvents.

We are now in the position to formulate the correspondence between derivations by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ and derivations by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ supplemented by the *eresolvent* procedure and to state the correctness of this approach to implementing $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$.

Let T be a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses such that T^{det} is the set of all determinate clauses in T and T^{ev} is the set of all eventuality clauses in T. Let $\lceil T^{det} \rceil$ denote the set $\{\lceil \Gamma \rceil \mid \Gamma \in T^{det}\}$ of first-order clauses representing the determinate clauses in T^{det}.

Then a $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$-*emulating derivation* from T by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ is a sequence $N_0, N_1, N_2, \ldots$ of sets of first order clauses such that $N_0 = \lceil T^{det} \rceil$ and for every i, $i \geq 0$, $N_{i+1} = N_1 \cup \{C\}$ where C is the condensation of a clause derived by applying the ordered resolution with selection rule of $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ with an atom ordering $\succ_{FOL}$ and selection function S_{FOL} corresponding to $\succ$ and S, respectively, to premises in N_i, or $N_{i+1} = N_1 \cup R$ where R is $eres(N_i, \Gamma)$ for some eventuality clauses Γ in T^{ev}. A $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$-*emulating refutation* of T by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ is a $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$-emulating derivation $N_0, N_1, \ldots$ from T by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ such that for some $i \geq 0$, N_i contains the empty clause.

Theorem 9 *Let T be a set of SNF$^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. Then T has a refutation by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ iff there is a $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$-emulating refutation of T by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$.*

Proof. Let $T_0, T_1, \ldots$ be a refutation of $T = T_0$ by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ where we restrict applications of ERES1 and ERES2 to loop formulae that can be found by a CTL equivalent of our loop search algorithm.

First, we establish that this restriction is still complete. Basically our loop search algorithm in Figure 10 is almost same as the original loop search algorithm in Section 3.3. The only difference is that we provide implementations for the following two tasks in the original loop search algorithm.

(i) "Search in T_i for all the clauses of the form $X_j \Rightarrow l$, $X_j \Rightarrow \mathbf{A}\bigcirc l$, and $X_j \Rightarrow \mathbf{E}\bigcirc l_{\langle ind \rangle}$."

(ii) "looking in T_i for merged clauses of the form $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}\bigcirc(B_j \wedge l)_{\langle ind \rangle}$ such that $B_j \Rightarrow H_i$ is provable (in propositional logic)."

Thus, we only need to prove the correctness of our implementation for those two tasks.

For task (i), we simply insert a new clause $ls(x) \vee \neg Q_l(\mathsf{app}(s, x))$, which is equivalent to an $\mathbf{A}$-step clause $\neg ls \Rightarrow \mathbf{A}\bigcirc\neg l$. By Theorem 8, we just need to prove that there exist clauses of the form $X_j \Rightarrow l$, $X_j \Rightarrow \mathbf{A}\bigcirc l$, or $X_j \Rightarrow \mathbf{E}\bigcirc l_{\langle ind \rangle}$ in T_i iff by adding $\neg ls \Rightarrow \mathbf{A}\bigcirc\neg l$ into T_i and applying step resolution of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for a certain atom ordering $\succ$ and a certain selection function S to T_i, we can find clauses $\neg ls \wedge X_j \Rightarrow \mathbf{false}$, $\neg ls \wedge X_j \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$, or $\neg ls \wedge X_j \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind \rangle}$ in $T_i \cup R$, where R is the set of resolvents.

If $\Gamma = X_j \Rightarrow \mathbf{A}\bigcirc l \in T_i$, then $\neg ls \Rightarrow \mathbf{A}\bigcirc\neg l$ is added into T_i. By SRES1, for an arbitrary atom ordering $\succ$ and arbitrary selection function S we can derive $\neg ls \wedge X_j \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$. This result obviously holds when $\Gamma = X_j \Rightarrow l$ and $\Gamma = X_j \Rightarrow \mathbf{E}\bigcirc l_{\langle ind \rangle}$.

For the other direction, if a clause $\Gamma = \neg ls \wedge X_j \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$ is found in $T_i \cup R$, then Γ is not from T_i because the atom ls does not occur in T_i before $\neg ls \Rightarrow \mathbf{A}\bigcirc\neg l$ is added. Thus Γ must be derived from an application of some step resolution rule of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for certain $\succ$ and S and one of the premise of this application must be $\neg ls \Rightarrow \mathbf{A}\bigcirc\neg l$. Therefore, the other premise in T_i must be $X_j \Rightarrow \mathbf{A}\bigcirc l$ or $X_j \Rightarrow l$. The result obviously holds when $\Gamma = \neg ls \wedge X_j \Rightarrow \mathbf{false}$ and $\Gamma = \neg ls \wedge X_j \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind \rangle}$.

For task (ii), the proof is analogous. By the completeness of loop search algorithm for CTL [5], our version of the loop search algorithm is complete as well.

We show by induction over the $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ refutation that we can construct a $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$-emulating derivation $N_0, N_1, \ldots$ from T such that for every i, $i \geq 0$, $N_i = \lceil T_i^{det} \rceil$. The base case, where we consider $T = T_0$ is trivial, as by definition $N_0 = \lceil T_0^{det} \rceil$. For the induction step, we have to consider whether T_{i+1} is derived from T_i by adding the resolvent of a step resolution inference or the results of an application of an eventuality resolution rule. In the first case, Theorem 8 to establish the required correspondence. In

```
procedure main($\varphi$)
// $\varphi$ is a CTL formula
begin
    N := transform_to_fol(transform_to_snf(simp(nnf($\varphi$))));
    New := $\{$C $|$ C is a determinate clause in N$\}$;
    ST  := $\{$C $|$ C is a sometime clause in N$\}$;
    SOS := $\emptyset$;
    do
        New := reduction_mrr(New);
        SOS := resolution_sos(SOS, New);
        New := $\emptyset$;
        if ($\bot \notin$ SOS) then
           foreach A-sometime clause and E-sometime clause C in ST
               G := eres(SOS, C);
               if (G $\neq$ $\emptyset$) then
                  New := New $\cup$ G;
               end if
           end for
           New := sub(New, SOS);
        end if
    while ($\bot \notin$ SOS and New $\neq \emptyset$)
    output();
end
```

Figure 13: The main procedure of CTL-RP

the second case, since we use essentially the same loop search algorithms, thus the *eresolvent* procedure in Figure 11 will find a first-order representation of the same loop formula and return the first-order representation of the same result.

Therefore, if T_i contains a contradiction for some $i \geq 0$, then N_i contains the empty clause as $N_i = \lceil T_i^{det} \rceil$. The $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$-emulating derivation $N_0, N_1, \ldots$ from T that we have just constructed is a refutation.

The proof for the reverse direction of the theorem is analogous. $\square$

4.5 The main procedure of our implementation

The architecture of our resolution theorem prover for CTL is dictated by the differentiation that we have to make between sometime clauses, which are subject to the eventuality resolution rules ERES1 and ERES1, implemented by the procedure *eres*, and determinate clauses, which are subject to the step resolution rules, implemented by ordered resolution with selection. There are two possibilities how these two can be integrated.

The first possibility is to treat *eres* as just another inference rule besides the resolution (and factoring) rule of first-order resolution. To illustrate this approach, consider the main procedure of a simple first-order resolution provers [26] as shown in Figure 12. In this procedure, *choose*(N) selects and removes a clause from a clause set N, *fac*(C) is the set of all factors derivable from a clause C, *res*(C, N) is the set of all resolvents derivable between a clause C and a set of clauses N, $taut(N)$ is the result of exhaustive tautology elimination to N, $sub(N)$ returns the set N after exhaustive application of subsumption deletion, and $sub(N, M)$ returns all clauses in N not subsumed by clauses in M. To integrate *eres* we could simply replace line 7 with a case distinction: if `Given` is the first-order representation of a determinate clause, then let `New` be the set of all condensed resolvents between `Given` and `Wo` under ordered resolution with selection; if `Given` is a sometime clause, then let `New` be the result of applying *eres* to the set of first-order representations of determinate clauses in $\mathtt{Us} \cup \mathtt{Wo}$, i.e. all currently available determinate clauses, and `Given`. However, *eres* assumes that the set of clauses it is given is already saturated and that only inferences between this set and the goal clauses constructed in *eres* are required, otherwise not all loop formulae might be found. But $\mathtt{Us}\cup\mathtt{Wo}$ is not saturated as inferences between clauses in `Us` have not been computed yet. So, we would need to saturate $\mathtt{Us}\cup\mathtt{Wo}$ within *eres* itself, which obviously leads to repeated inferences as *resolution_prover* will continue to saturate $\mathtt{Us} \cup \mathtt{Wo}$ independently of *eres*. Thus, this would not be an efficient approach.

The second possibility is to perform the saturation of determinate clauses first before we try to apply *eres*. This obviously ensures that *eres* receives a saturated set of determinate clauses as input. But since each application of *eres* to a sometime clause may derive new determinate clauses, we will have to reiterate the overall saturation process with these new clauses. This gives rise to the algorithm for the main procedure of CTL-RP shown in Figure 13. The procedure takes a CTL formula φ as input and transforms φ into a set `N` of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses in first-order representation by computing the negation normal form of φ using *nnf* and performing boolean and CTL simplifications, including tautology removal, using *simp*, then transforming the resulting CTL formula into an equi-satisfiable set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses using *transform_to_snf*, and finally giving these clauses a first-order representation using *transform_to_fol* (line 4). We split `N` into the set `New` of first-order representations of determinate clauses and the set `ST` of sometime clauses (lines 5 and 6). As we will repeatedly saturate a set of clauses, a set of support strategy is used, with the initial set of support `SOS` being empty (line 7).

We then enter the main loop of the procedure which will be repeated until either the empty clause has been derived or we cannot derive any new clauses. Within the main loop we first simplify `New` using matching replacement resolution [22] (line 9) which we found to be an effective reduction in early experiments with CTL-RP. We then saturate the set `New` with respect to the current set of support `SOS` using the procedure *resolution_sos* and the resulting set of clauses becomes the new set of support (line 10). If we have not derived the empty clause yet, then we try to apply *eres* to each of the sometime clauses (lines 13 to 18). The union of all the resolvents generated by applications of *eres* becomes the set of new clauses `New`. Some of these resolvents may be redundant, in particular, if applications of *eres* in a previous iteration of the loop have already been successful, i.e. have produced a non-empty set of resolvents. Therefore, we eliminate clauses from `New` which are subsumed by clauses in `SOS` (line 19).

```
procedure resolution_sos(SOS, N)
// SOS is a saturated set of first-order clauses
// N   is a non-saturated set of first-order clauses
begin
    while (N $\neq \emptyset$ and $\perp \notin$ N)
        (Given,N) := choose(N);
        SOS := SOS $\cup$ {Given};
        New := cond(ores(Given, SOS));
        New := sub(sub(New, SOS), N);
        SOS := sub(SOS, New);
        N := sub(N, New) $\cup$ New;
    end
    if $\perp$ $\in$ N then
        SOS := SOS $\cup$ {$\perp$};
    end if
    return SOS;
end
```

Figure 14: The *resolution_sos* procedure

The procedure *resolution_sos* is shown in Figure 14. The procedure takes as input a set of clauses SOS which is assumed to be saturated and not to contain a contradiction, and a set of clauses N. It returns the saturation of SOS $\cup$ N. The procedure is a minor variation of the simple resolution prover *resolution_prover* in Figure 12, with the set SOS taking the place of the set Wo of worked-off clauses. Thus, while *resolution_prover* starts with an empty set of worked-off clauses to which we add clauses chosen from N, and from derived clauses, one by one, here we start with the potentially non-empty set SOS to which we add clauses chosen from N, and from derived clauses. In addition, we require the use of ordered resolution with selection: $ores(C, N)$ is the set of all resolvents derivable between a clause C and a set of clauses N by the ordered resolution with selection rule, $cond(N)$ is the set of clauses $\{\text{Cond}(C) \mid C \in N\}$, where N is a set of determinate clauses.

4.6 CTL-RP

Our resolution theorem prover for CTL, CTL-RP, is based on the first-order resolution prover SPASS 3.0 [24, 27]. The main procedure of SPASS provides the implementation of *resolution_sos* and all the inference and redundancy elimination rules for first-order ordered resolution with selection. To this we have added our own implementations of the procedures *nnf*, *simp*, *transform_to_snf*, and *transform_to_fol* that are required to transform a given CTL formula into a set of first-order representations of determinate clauses and a set of sometime clauses, and we have added implementations of the procedures *main*, *eres* and *eresolvent*.

TRES1
$$\frac{\begin{array}{c} P^\dagger \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l \\ q \Rightarrow \mathbf{A}\Diamond\neg l \end{array}}{q \Rightarrow \mathbf{A}(\neg P^\dagger \mathcal{W} \neg l)}$$

TRES2
$$\frac{\begin{array}{c} P^\dagger \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l \\ q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(ind)\rangle} \end{array}}{q \Rightarrow \mathbf{E}(\neg P^\dagger \mathcal{W} \neg l)_{\langle LC(ind)\rangle}}$$

where $P^\dagger$ is a disjunction of conjunctions of literals and l and q are literals.

Figure 15: Redundant eventuality resolution rules

5 Related work

$\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is based on Bolotov's resolution calculus for CTL [5]. For instance, the use of indices to translate into the normal form was introduced in [5]. However, no formal interpretation was given for indices and no formal semantics stated for $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$. In this paper, we provide a formal semantics for $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$.

Compared to the definition of $\mathrm{SNF}_{\mathrm{CTL}}$ in [5], we use an additional type of clauses, namely *global clauses*. Our definition of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ provides several advantages over [5]. Firstly, global clauses inevitably occur as a result of inferences by step resolution rules. For example, from $m_1 \Rightarrow \mathbf{A}\bigcirc l$ and $m_2 \Rightarrow \mathbf{A}\bigcirc\neg l$ we can derive $m_1 \wedge m_2 \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$, while from $m_1 \Rightarrow \mathbf{E}\bigcirc l_{\langle ind\rangle}$ and $m_2 \Rightarrow \mathbf{E}\bigcirc\neg l_{\langle ind\rangle}$ we can derive $m_1 \wedge m_2 \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind\rangle}$. Both $m_1 \wedge m_2 \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$ and $m_1 \wedge m_2 \Rightarrow \mathbf{E}\bigcirc\mathbf{false}_{\langle ind\rangle}$ are transformed into a global clause $\mathbf{true} \Rightarrow \neg m_1 \vee \neg m_2$ by RW1 and RW2, respectively.

As the normal form in [5] does not allow for such clauses, in the approach taken in [5] such global clauses must further be rewritten into equivalent pairs of an initial clause and an **A**-step clause as follows:

$$\mathbf{true} \Rightarrow \bigvee_{j=1}^{k} m_j \;\longrightarrow\; \begin{cases} \mathbf{start} \Rightarrow \bigvee_{j=1}^{k} m_j \\ \mathbf{true} \Rightarrow \mathbf{A}\bigcirc \bigvee_{j=1}^{k} m_j \end{cases}$$

where each m_j, $1 \leq j \leq k$, is a literal. For the same reason, in [5] the rewrite rules RW1 and RW2 will each produce two clauses, whereas in $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ the analogous rewrite rules produce only one. Thus, one obvious advantage of allowing global clauses is that compared to [5] we will have fewer clauses transformed from the original CTL formula and generated by resolution.

Secondly, global clauses also inevitably occur as a result of inferences in an implementation of step resolution via first-order resolution as described in the previous section. Thus, removing global clauses via rewriting would require additional implementation effort in this approach. Furthermore, the main disadvantage of the introduction of global clauses, namely the need for the additional step resolution rules SRES5 to SRES8 that allow to resolve on global clauses, disappears. All step resolution rules map onto the rule for ordered first-order resolution.

Another difference to [5] is the approach taken in our completeness proof. The proof in [5] relates the application of deletion rules on a CTL tableau to a sequence of resolution steps. Then, completeness of the resolution calculus follows from the completeness of the tableau construction and deletion process. To show completeness of our calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ we construct a graph known as a *labelled behaviour graph.*

This is an extension of the concept of a behaviour graph used in [16] for proving completeness of a clausal resolution for PLTL and related to the concept of a labelled behaviour graph used [12]. However, our labelled behaviour graph has differences in construction to capture the semantics of indices in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$. We believe our completeness proof through a behaviour graph demonstrates a closer relationship between the application of resolution rules and deletions in the labelled behaviour graph. Moreover, it is relatively easy to generate a CTL model structure from a non-empty reduced labelled behaviour graph. Hence, we could potentially use the labelled behaviour graph construction to generate counter models given failed proofs. Our labelled behaviour graph can be easily extended to be used for a completeness proof of resolution calculi for the combination of CTL and other logics, for example, the combination of CTL and modal logic KD45 [12].

Furthermore, in the resolution calculus for CTL presented in [5, 12] step resolution is not constrained by an ordering and a selection function. Therefore, the step resolution rules in [5, 12] allow for considerably more, and superfluous, inferences. In addition, this earlier resolution calculus contains four eventuality resolution rules, TRES1 to TRES4, where ERES1 and ERES2 correspond to TRES3 and TRES4. The other two eventuality resolution rules are given below in Figure 15. Using our completeness proof we can prove that the two eventuality resolution rules TRES1 and TRES2 in [5, 12] are redundant.

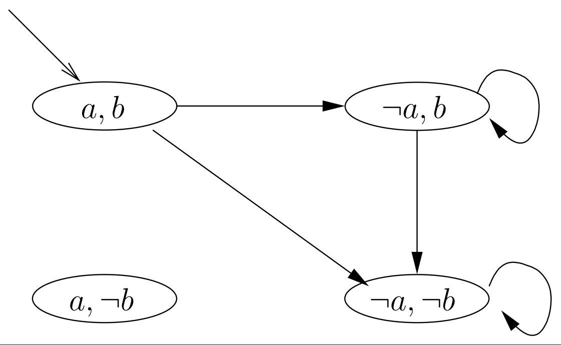
We give a brief explanation why this is the case. Informally note that the only difference between TRES1 and ERES1 is their first premise. For TRES1, it is $P^{\dagger} \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l$ and for ERES1, it is $P^{\dagger} \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$. In [5, 12], $\mathbf{A}\bigcirc\mathbf{A}\Box l$ is called an **A**-loop and $\mathbf{E}\bigcirc\mathbf{E}\Box l$ is called an **E**-loop. According to the semantics of CTL, $\mathbf{A}\bigcirc\mathbf{A}\Box l \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$, meaning if there exists an **A**-loop, there must be an **E**-loop as well. So, whenever we can apply TRES1 (TRES2), ERES1 (ERES2) is applicable as well. More formally, in our completeness proof we only identify two types of subgraphs where some eventuality can not be fulfilled, namely, *ind*-labelled terminal subgraphs and terminal subgraphs. Both are **E**-loops according to the definition in [5] and the deletion of both types of subgraphs correlates to applications of ERES1 or ERES2. Thus, no further inference rules are required showing that TRES1 and TRES2 are redundant. Considering that the eventuality resolution rules are computationally very expensive, we gain a significant improvement here.

Finally, complexity of the method is not discussed in [5]. In this paper, we prove that a decision procedure based on $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is of the order EXPTIME.

6 Performance of CTL-RP

Besides CTL-RP, there only seems to be one other CTL theorem prover, namely a CTL module for the Tableau Workbench (TWB) [1].

The Tableau Workbench is a generic framework for building automated theorem provers for arbitrary



Representation of this state transition system in CTL

$$\begin{array}{rl}
(a \wedge b) & \\
\mathbf{A}\Box(a \wedge b & \Rightarrow \mathbf{A}\bigcirc((\neg a \wedge b) \vee (\neg a \wedge \neg b))) \\
\mathbf{A}\Box(a \wedge b & \Rightarrow \mathbf{E}\bigcirc(_{\langle()\rangle}\neg a \wedge b))) \\
\mathbf{A}\Box(a \wedge b & \Rightarrow \mathbf{E}\bigcirc(_{\langle()\rangle}\neg a \wedge \neg b))) \\
\mathbf{A}\Box(\neg a \wedge b & \Rightarrow \mathbf{A}\bigcirc((\neg a \wedge b) \vee (\neg a \wedge \neg b))) \\
\mathbf{A}\Box(\neg a \wedge b & \Rightarrow \mathbf{E}\bigcirc(_{\langle()\rangle}\neg a \wedge b))) \\
\mathbf{A}\Box(\neg a \wedge b & \Rightarrow \mathbf{E}\bigcirc(_{\langle()\rangle}\neg a \wedge \neg b))) \\
\mathbf{A}\Box(\neg a \wedge \neg b & \Rightarrow \mathbf{A}\bigcirc(\neg a \wedge b))
\end{array}$$

Figure 16: A state transition system

	Property	Status	CTL-RP	TWB
1.	$\mathbf{A}\bigcirc(\neg\mathbf{E}\Diamond(_{\langle LC(a)\rangle} \wedge b))$	Valid	0.06s	23.79s
2.	$\mathbf{A}\Box(\mathbf{A}(a\,\mathcal{U}\,\neg a))$	Valid	0.02s	25.84s
3.	$\mathbf{A}\Box(a \vee b)$	Not Valid	0.01s	0.85s
4.	$\mathbf{A}\Box(\mathbf{E}\bigcirc\neg_{\langle b\rangle})$	Valid	0.03s	46.95s
5.	$\mathbf{E}(b\,\mathcal{U}\,\neg b)$	Valid	0.01s	2.95s

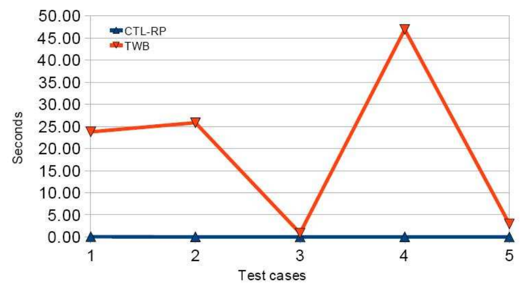


Figure 17: Performance on finite state transition system properties

propositional logics which provides a general architecture and a high-level language which allows users to specify tableau rules and provers based on these rules. It provides a number of pre-defined provers for a wide range of logics, for example, propositional logic, linear-time temporal logic and CTL. Regarding CTL, it implements a so-called one-pass tableau calculus for this logic which results in double-EXPTIME decision procedure [2]. Therefore the complexity this CTL decision procedure is higher than the complexity of CTL-RP, which is EXPTIME. It should note that the prime aim of TWB is not efficiency.

There is no established way to evaluate the performance of CTL decision procedures nor is there a repository or random generator of CTL formulae that one might use for such an evaluation. We have therefore created four sets of benchmark formulae ourselves that we have used to compare CTL-RP version 00.09 with TWB version 3.4. The comparison was performed on a Linux PC with an Intel Core 2 CPU@2.13 GHz and 3G main memory, using the Fedora 9 operating system.

The first set of benchmark formulae, CTL-BF1, consist of eight well-known equivalences between temporal formulae taken from [13].

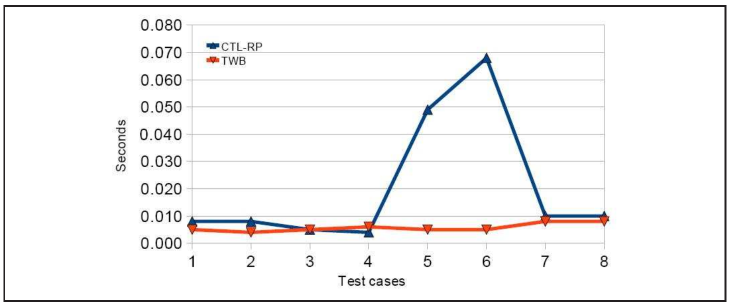


Figure 18: Performance on eight 'textbook' CTL formulae

	CTL equivalence	CTL-RP	TWB
1.	$\mathbf{A}\Box p \equiv \neg\mathbf{E}\Diamond\neg p$	0.008s	0.005s
2.	$\mathbf{E}\Box p \equiv \neg\mathbf{A}\Diamond\neg p$	0.008s	0.004s
3.	$\mathbf{E}\bigcirc(p \vee q) \equiv \mathbf{E}\bigcirc p \vee \mathbf{E}\bigcirc q$	0.005s	0.005s
4.	$\mathbf{A}\bigcirc p \equiv \neg\mathbf{E}\bigcirc\neg p$	0.004s	0.006s
5.	$\mathbf{E}(p\,\mathcal{U}\,q) \equiv q \vee (p \wedge \mathbf{E}\bigcirc\mathbf{E}(p\,\mathcal{U}\,q))$	0.049s	0.005s
6.	$\mathbf{A}(p\,\mathcal{U}\,q) \equiv q \vee (p \wedge \mathbf{A}\bigcirc\mathbf{A}(p\,\mathcal{U}\,q))$	0.068s	0.005s
7.	$\mathbf{E}\Diamond p \equiv \mathbf{E}(\mathbf{true}\ \mathcal{U}\,p)$	0.010s	0.008s
8.	$\mathbf{A}\Diamond p \equiv \mathbf{A}(\mathbf{true}\ \mathcal{U}\,p)$	0.010s	0.008s

The CTL equivalences themselves and the CPU time required by TWB and CTL-RP to prove each of them is shown in Figure 18. Both system easily prove each of the equivalence in less then 0.1 seconds, however, with TWB being significantly faster on two of the formulae.

For the second set of benchmark formulae, CTL-BF2, we have created a small finite state transition system and formalised it in CTL as shown in Figure 16. We have then defined five properties, each given by a CTL formula, that one might try to establish for this state transition system, and each benchmark formula in the second set is an implication stating that the CTL specification of the finite state system implies one of these properties. Figure 17 shows the five properties, their validity status with respect to the finite state transition system, and the CPU time in second required by TWB and CTL-RP to establish that status. CTL-RP outperforms TWB by a factor of about 1000 on two of the benchmark formulae and by a factor of 100 for the remaining three benchmark formulae in CTL-BF2.

The third set of benchmarks, CTL-BF3, generalises the idea underlying CTL-BF2. Instead of using a specification of a finite state system and properties that we have 'crafted' ourselves, we use randomly generates ones. In particular, let a *state specification* be a conjunction of literals l_i, $1 \leq i \leq 4$, with each l_i being an element of $\{a_i, \neg a_i\}$. Let a *transition specification* be a CTL formula in the form $\mathbf{A}\Box(s \Rightarrow \mathbf{A}\bigcirc(\bigvee_{i=1}^{n} s_i))$ or $\mathbf{A}\Box(s \Rightarrow \mathbf{E}\bigcirc(\bigvee_{i=1}^{n} s_i))$, where n is a randomly generated number between 1 and 3, and s and each s_i, $1 \leq i \leq n$ is a randomly generated state specification. Furthermore, let a

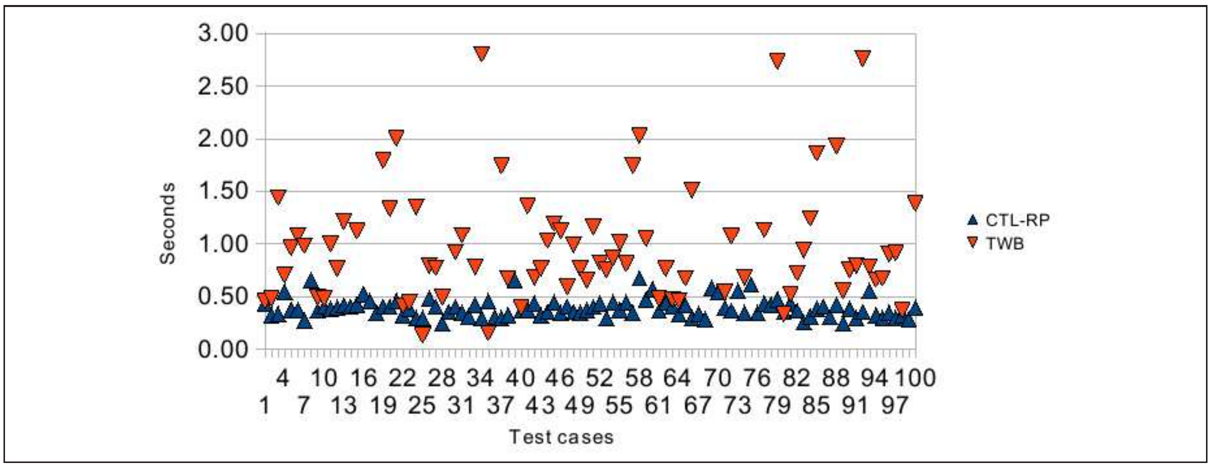


Figure 19: Performance on the third set of benchmark formulae CTL-BF3

property specification be a CTL formula of the form $*(\bigvee_{i=1}^{n} s_i)$, where $*$ is a randomly chosen element of $\{\mathbf{A}\bigcirc, \mathbf{E}\bigcirc, \mathbf{A}\Box, \mathbf{E}\Box, \mathbf{A}\Diamond, \mathbf{E}\Diamond, \mathbf{A}\mathcal{U}, \mathbf{E}\mathcal{U}\}$, n is a randomly generated number between 1 and 2, and each s_i, $1 \leq i \leq n$, is a randomly generated state specification. CTL-BF3 consists of one hundred formulae with each formula being a conjunction (set) of 30 transition specifications and 5 property specifications. Figure 19 shows a graph indicating the CPU in seconds required by TWB and CTL-RP to establish the satisfiability or unsatisfiability of each benchmark formula in CTL-BF3. For CTL-RP, each of the 100 benchmark formulae was solved in less than one CPU second. TWB, on the other hand, required more time for most of benchmark formulae and was not able to solve 21 of the benchmark formulae in less than 200 CPU seconds each, which was the time limit we have given to both provers. The results on the CTL-BF3 show that CTL-RP can provide a proof for each benchmark formula in a reasonable time with the vast majority of formulae being solved in less than 0.50 seconds. In contrast, the performance of TWB is much more variable, with a high percentage of formulae not being solved. The last set of benchmarks CTL-BF4 is based on a real world problem. We have specified a network protocol, the Alternating Bit Protocol (ABP) [21] in CTL and specified and verified three of its properties by CTL-RP and TWB. The Alternating Bit Protocol involves two participants, namely a *Transmitter* and a *Receiver*. The Transmitter wants to send messages in a reliable way to the Receiver through an unreliable communication channel. To this end, the Transmitter appends to each message a control bit. We assume that for the first message the Transmitter sends, it will use the control bit 0. The Transmitter will repeatedly send the message including the control bit until it receives an acknowledgement from the Receiver with the same control bit. The Transmitter will then complement the control bit and start transmitting the next message including the new control bit.

This behaviour of Transmitter and Receiver can be described by finite state transitions systems as the ones shown in Figure 20. If the Transmitter is in its initial state $s0$, then it attaches the control bit 0 to the current message and sends it to the Receiver. It will stay in state $s0$, i.e. follow the transition labelled

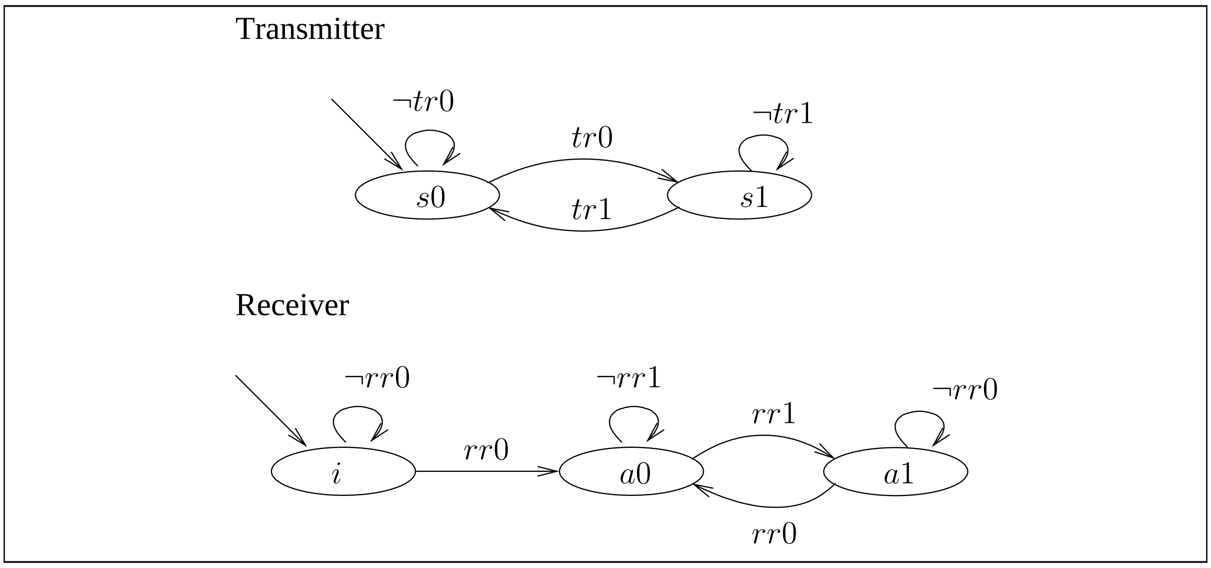


Figure 20: A model for ABP

$\neg tr0$, until it receives an acknowledgement with control bit 0, in which case it follows the transition labelled *tr0* to state $s1$. The behaviour of the Transmitter in state $s1$ is identical to its behaviour in state $s0$, but with the control bit 1 taking the place of control bit 0.

If the Receiver is in its initial state i, then it will stay in state i, i.e. follow the transition labelled $\neg rr0$, until it receives a message with control bit 0. It will then follow the transition labelled *rr0* to state $a0$. In state $a0$, the Receiver will send an acknowledgement with control bit 0 to the Transmitter. It will then stay in state $a0$, i.e. follow the transition labelled $\neg rr1$ until it receives a message with control bit 1. It then follows the transition labelled *rr1* to state $a1$. The behaviour of the Receiver in state $a1$ is identical to its behaviour in state $a0$, but with the control bit 1 taking the place of control bit 0.

To represent the behaviour of Transmitter and Receiver in CTL, we associate a propositional variable with every state and every positive transition label in the two finite state transition systems. Then the initial condition of the Transmitter can be described by the CTL formula

$$s0 \wedge \neg tr0 \wedge \neg tr1$$

and the transitions of the Transmitter are represented by the following CTL formulae.

$$\begin{aligned}
\mathbf{A}\Box(s0 \wedge \neg tr0 \Rightarrow \mathbf{A}\bigcirc s0)\\
\mathbf{A}\Box(s0 \wedge tr0 \Rightarrow \mathbf{A}\bigcirc s1)\\
\mathbf{A}\Box(s1 \wedge \neg tr1 \Rightarrow \mathbf{A}\bigcirc s1)\\
\mathbf{A}\Box(s1 \wedge tr1 \Rightarrow \mathbf{A}\bigcirc s0)
\end{aligned}$$

Moreover, the following formulae ensure that at any moment, the Transmitter can only be at one state.

$$\mathbf{A}\Box(s0 \vee s1)$$

$$\mathbf{A}\Box(s0 \Rightarrow \neg s1)$$

$$\mathbf{A}\Box(s1 \Rightarrow \neg s0)$$

In analogy, the initial condition of the Receiver is described by

$$i \wedge \neg rr0 \wedge \neg rr1$$

and the transitions of the Receiver are represented by the following CTL formulae.

$$\mathbf{A}\Box(i \wedge \neg rr0 \Rightarrow \mathbf{A}\bigcirc i)$$

$$\mathbf{A}\Box(i \wedge rr0 \Rightarrow \mathbf{A}\bigcirc a0)$$

$$\mathbf{A}\Box(a0 \wedge \neg rr1 \Rightarrow \mathbf{A}\bigcirc a0)$$

$$\mathbf{A}\Box(a0 \wedge rr1 \Rightarrow \mathbf{A}\bigcirc a1)$$

$$\mathbf{A}\Box(a1 \wedge \neg rr0 \Rightarrow \mathbf{A}\bigcirc a1)$$

$$\mathbf{A}\Box(a1 \wedge rr0 \Rightarrow \mathbf{A}\bigcirc a0)$$

Again, we impose additional constrains to ensure that at any moment, the Receiver can only be in one state.

$$\mathbf{A}\Box(i \vee a0 \vee a1)$$

$$\mathbf{A}\Box(i \Rightarrow \neg a0 \wedge \neg a1)$$

$$\mathbf{A}\Box(a0 \Rightarrow \neg i \wedge \neg a1)$$

$$\mathbf{A}\Box(a1 \Rightarrow \neg i \wedge \neg a0)$$

In addition, we specify that the Transmitter and the Receiver will always eventually be successful in transmitting their messages.

$$\mathbf{A}\Box(s0 \Rightarrow \mathbf{A}\Diamond rr0)$$

$$\mathbf{A}\Box(s1 \Rightarrow \mathbf{A}\Diamond rr1)$$

$$\mathbf{A}\Box(a0 \Rightarrow \mathbf{A}\Diamond tr0)$$

$$\mathbf{A}\Box(a1 \Rightarrow \mathbf{A}\Diamond tr1)$$

Finally, we have to specify the properties that we want to establish. We want to prove that the Receiver is initially in state i and remains in that state until it transits to state $a0$. Once in state $a0$ it will remain there until it transits to state $a1$. In analogy, once in state $a1$ the receiver remains in that state until it transits to state $a0$. These three properties are given by the following CTL formulae:

1. $\mathbf{A}(i\,\mathcal{U}\,a0)$
2. $\mathbf{A}\Box(a0 \Rightarrow \mathbf{A}(a0\,\mathcal{U}\,a1))$
3. $\mathbf{A}\Box(a1 \Rightarrow \mathbf{A}(a1\,\mathcal{U}\,a0))$

The set CTL-BF4 consists of three formulae with each formula being an implication stating that the conjunction (set) of CTL formulae specifying Transmitter and Receiver implies one of the three properties above.

While CTL-RP was able to establish the validity of each of the three benchmark formulae as indicated in the table below, TWB did not terminate within 20 hours of CPU time.

Property	CTL-RP	TWB
1	1.39s	-
2	192.86s	-
3	326.02s	-

7 Conclusion

Currently, there are many non-classical logics for which sound and complete calculi are known, however, implementations of these calculi are lacking. This applies even to such a well-known and well-established logic as Computational Tree Logic. One explanation is the considerable effort that is required to implement a reasonably efficient theorem prover for these logics. The approach we took is that we first develop a resolution calculus for the non-classical logic we are interested in and then build a bridge to first-order resolution which allows us to re-use existing first-order theorem provers.

Previously, this approach has been successfully applied to linear-time temporal logic [19]. In this paper, we construct a bridge from CTL to first-order logic.

We define a new normal form $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ for CTL and give a formal semantics for it. We provide an improved clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL and show the EXPTIME complexity of a CTL decision procedure based on $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. Furthermore, we present a new completeness proof with a different approach from [5]. The proof also shows that some eventuality resolution rules in [5], which are the most costly rules of the calculus, are redundant. We provide a new technique to implement step resolution rules via ordered first-order ordered resolution with selection and describe an algorithm for the eventuality resolution rules of our calculus. This makes our calculus useful from both a theoretical and a practical perspective. Using the methods we propose in this paper, we utilise an existing, efficient automated resolution theorem prover for first-order logic, SPASS, to implement our CTL theorem prover CTL-RP.

In future, we intend to extend our approach to logics closely related to CTL including, for example, alternating-time temporal logic [3].

References

[1] P. Abate and R. Goré. The Tableaux Workbench. In *Proc. TABLEAUX'03*, pages 230–236. LNCS 2796. Springer, 2003.

[2] P. Abate, R. Goré, and F. Widmann. One-Pass Tableaux for Computation Tree Logic. In *Proc. LPAR'07*, pages 32–46. LNCS 4790. Springer, 2007.

[3] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[4] L. Bachmair and H. Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–99. Elsevier, 2001.

[5] A. Bolotov. *Clausal Resolution for Branching-Time Temporal Logic.* PhD thesis, Manchester Metropolitan University, 2000.

[6] A. Bolotov and C. Dixon. Resolution for Branching Time Temporal Logics: Applying the Temporal Resolution Rule. In *Proc. TIME'00*, pages 163–172. IEEE, 2000.

[7] A. Bolotov and M. Fisher. A Clausal Resolution Method for CTL Branching-Time Temporal Logic. *JETAI*, 11(1):77–93, 1999.

[8] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. CAV'02*, pages 359–364. Springer, 2002.

[9] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71. LNCS 131. Springer, 1982.

[10] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 2000.

[11] C. Dixon. Temporal Resolution Using a Breadth-First Search Algorithm. *Annals of Mathematics and Artificial Intelligence*, 22(1-2):87–115, 1998.

[12] C. Dixon, M. Fisher, and A. Bolotov. Clausal Resolution in a Logic of Rational Agency. *Artifical Intelligence*, 139(1):47–89, 2002.

[13] E. A. Emerson. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, chapter 16, pages 996–1072. Elsevier, 1990.

[14] E. A. Emerson and J. Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.

[15] E. A. Emerson, T. Sadler, and J. Srinivasan. Efficient Temporal Satisfiability. *J. Log. Comput.*, 2(2):173–210, 1992.

[16] M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, 2001.

[17] B. T. Hailpern. *Verifying Concurrent Processes Using Temporal Logic.* Springer, 1982.

[18] K. Heljanko. Model Checking the Branching Time Temporal Logic CTL. Research Report A45, Helsinki University of Technology, 1997.

[19] U. Hustadt and B. Konev. TRP++ 2.0: A temporal resolution prover. In *Proc. CADE-19*, volume 2741 of *LNAI*, pages 274–278. Springer, 2003.

[20] U. Hustadt and B. Konev. TRP++: A Temporal Resolution Prover. In *Collegium Logicum*, pages 65–79. Kurt Gödel Society, 2004.

[21] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, 2004.

[22] T. Lev-Ami, C. Weidenbach, T. W. Reps, and M. Sagiv. Labelled clauses. In *Proc. CADE-21*, volume 4603 of *LNCS*, pages 311–327, 2007.

[23] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer, 1992.

[24] Max-Planck-Institut Informatik. Automation of logic: Spass. `http://www.spass-prover.org/download/`.

[25] M. Reynolds. Towards a CTL* Tableau. In *Proc. FSTTCS'05*, volume 3821 of *Lecture Notes in Computer Science*, pages 384–395. Springer, 2005.

[26] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 27, pages 1967–2015. Elsevier, 2001.

[27] C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: Spass version 3.0. In *Proc. CADE-21*, pages 514–520. LNCS 4603. Springer, 2007.