Clausal Reasoning for Branching-Time Logics

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor of Philosophy
by

Lan Zhang

December 2010

Abstract

Computation Tree Logic (CTL) is a branching-time temporal logic whose underlying model of time is a choice of possibilities branching into the future. It has been used in a wide variety of areas in Computer Science and Artificial Intelligence, such as temporal databases, hardware verification, program reasoning, multi-agent systems, and concurrent and distributed systems.

In this thesis, firstly we present a refined clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL. The calculus requires a polynomial time computable transformation of an arbitrary CTL formula to an equi-satisfiable clausal normal form formulated in an extension of CTL with indexed existential path quantifiers. The calculus itself consists of eight step resolution rules, two eventuality resolution rules and two rewrite 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, establish that the clausal normal form transformation preserves satisfiability, provide proofs for the soundness and completeness of the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, and discuss the complexity of the decision procedure based on $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. As $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is based on the ideas underlying Bolotov's clausal resolution calculus for CTL, we provide a comparison between our calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ and Bolotov's calculus for CTL in order to show that $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ improves Bolotov's calculus in many areas. In particular, our calculus is designed to allow first-order resolution techniques to emulate resolution rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ so that $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ can be implemented by reusing any first-order resolution theorem prover.

Secondly, we introduce CTL-RP, our implementation of the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. CTL-RP is the first implemented resolution-based theorem prover for CTL. The prover takes an arbitrary CTL formula as input and transforms it into a set of CTL formulae in clausal normal form. Furthermore, in order to use first-order techniques, formulae in clausal normal form are transformed into first-order formulae, except for those formulae related to eventualities, i.e. formulae containing the eventuality operator $\Diamond$. To implement step resolution and rewrite rules of the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, we present an approach that uses first-order ordered resolution with selection to emulate the step resolution rules and related proofs. This approach enables us to make use of a first-order theorem prover, which implements the first-order ordered resolution with selection, in order to realise our calculus. Following this approach, CTL-RP utilises the first-order theorem prover SPASS to conduct resolution inferences for CTL and is implemented as a modification of SPASS. In particular, to implement the eventuality resolution rules, CTL-RP augments SPASS with an algorithm, called loop search algorithm for tackling eventualities in CTL. To study the performance of CTL-RP, we have compared CTL-RP with a tableau-based theorem prover for CTL. The experiments show good performance of CTL-RP.

Thirdly, we apply the approach we used to develop $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ to the development of a clausal resolution calculus for a fragment of Alternating-time Temporal Logic (ATL). ATL is a generalisation and extension of branching-time temporal logic, in which the temporal operators are parameterised by sets of agents. Informally speaking, CTL formulae can be treated as ATL formulae with a single agent. Selective quantification over paths enables ATL to explicitly express coalition abilities, which naturally makes ATL a formalism for specification and verification of open systems and game-like multi-agent systems. In this thesis, we focus on the Next-time fragment of ATL (XATL), which is closely related to Coalition Logic. The satisfiability problem of XATL has lower complexity than ATL but there are still many applications in various strategic games and multi-agent systems that can be represented in and reasoned about in XATL.

In this thesis, we present a resolution calculus $\mathsf{R}_{\mathrm{XATL}}$ for XATL to tackle its satisfiability problem. The calculus requires a polynomial time computable transformation of an arbitrary XATL formula to an equi-satisfiable clausal normal form. The calculus itself consists of a set of resolution rules and rewrite rules. We prove the soundness of the calculus and outline a completeness proof for the calculus $\mathsf{R}_{\mathrm{XATL}}$. Also, we intend to extend our calculus $\mathsf{R}_{\mathrm{XATL}}$ to full ATL in the future.

Acknowledgement

A huge thank to the best parents a child could hope for, Qing Jie Zhang and Wei Wei Zhang, for their unconditional love and support, and for always being there for me whenever I need them. There are no ways that I can thank them enough for how much both of them have helped with my life and career, and taught and given me all the things that have gotten me here. My parents have been my rock. I want to thank my incredible wife Sherly Novelia Nietiadi, for continuing understanding and great sacrifice she has made for these years in the support of my studying, for hard and tedious house work she has done so that I have more time to spend on my research, and for tremendous love she has poured into my heart to help me to recover quickly when I was very weak in the hospital.

I would like to thank my supervisors Dr. Ullrich Hustadt and Dr. Clare Dixon, for introducing me to the field of temporal logic and in particular, resolution-based approaches for automated theorem proving in temporal logic, for their expert advice and guidance that helps me find new directions when my research approaches a dead end and makes me realise what the important issues are when my thoughts are clouded by many trivial details, for their continued encouragement and inspiration, and for their great patience when testing the resolution-based theorem prover for Computation Tree Logic I have developed.

I am so grateful for the effort Dave Shield made to configure and install all software I needed and solve all technical problems I encountered. I want to express my gratitude to my colleague Michel Ludwig for his detailed explanation of the internal structure of the first-order resolution theorem prover SPASS, which helped me a lot with the development of my own theorem prover CTL-RP.

Thank everyone in LoCo Group of Computer Science Department at the University of Liverpool for providing a friendly academic environment.

Last but not least, the financial support (EPSRC grant EP/D060451/1) by the Engineering and Physical Sciences Research Council is gratefully acknowledged.

Contents

Chapter 1

Introduction

In this thesis, we investigate two formal verification methods, which can be used for many various types of information processing systems to check whether an implementation satisfies a specification.

We first give an introduction of the background of formal methods. We briefly discuss where and why formal methods are desirable and present two essential components of formal methods, namely formal specification and formal verification. In addition, we also briefly introduce the validity and the satisfiability problem, which our methods aim to solve. Secondly, we present our main contributions in Section 1.2 and, lastly, we provide an overview of this thesis in Section 1.3.

1.1 Formal methods for system designs

The development of various computer-based systems, for example mobile phones, modern computer operating systems, air traffic control systems, flood control systems, and missile launcher systems, is a real challenge, especially if they are concurrent and distributed systems. Since the development of such systems is so complicated, they are prone to have errors and, consequently, the systems malfunction. In the following, we show several examples to illustrate some system failures and how extensive the damage, caused by those failures, can be.

1. In 1993, a floating point division bug was discovered in Intel's highly promoted Pentium chip. Due to this flaw, Intel lost 475 million dollars in total for replacing many sold chips. Moreover, this design flaw severely damaged Intel's reputation for manufacturing high quality CPUs.

2. In 1996, the Ariane 5, an unmanned rocket, exploded along with its cargo of four scientific research satellites, shortly after its launch. This failure was caused by an overflow error, which was generated by a conversion from a 64-bit floating-point number into a 16-bit integer. This loss cost nearly 500 million dollars.

3. Last but not least, in 1983, the Soviet early warning system falsely gave an alarm reporting that the United States had launched five missiles towards the Soviet Union due to a defect in its software system, which failed to filter out false missile detection generated by sunlight reflections. Fortunately, the Soviet duty officer had reported it as a false alarm instead of

following the protocol to immediately respond to the attack with their own nuclear missiles. This event could have caused the loss of millions of human lives or even worse than that, i.e. the whole planet could have been consumed by nuclear war.

Therefore, it is commonly agreed in the Information Technology industry that ensuring the correctness of hardware/software system designs is important. For example, in many software development methodologies such as *Test-driven Development* and *Extreme Programming*, it is an important principle that the code for testing is written first before the code of the actual software is written. However, in reality correctness is still very hard to achieve, although people pay extensive attention to it.

As people rely on computers to do more and more tasks, the complexity of many computer-based systems is increasing inevitably quickly, however the measures to guarantee the correctness of the designs for those systems can not keep up with the speed with which their complexity grows. The current most common verification methods used in the Information Technology industry are *simulation* and *testing*. These usually only test some of the possible behaviours of the systems and are obviously not sufficient to be the only methods for system verification. Thus, the demands for new verification methods, which

1. can explore all possible behaviours,
2. are able to automatically and efficiently find subtle system design flaws in rather complex systems,
3. improve the confidence in the correctness of the design, and
4. notably reduce the effort expended on testing,

is growing fast. Formal methods have been developed for these purposes.

Formal methods have attracted more and more attention not only from academic researchers but also from engineers and designers who work in the Information Technology industry. As a result, many formal method tools [21, 46, 48, 14, 2, 8] have been developed and applied to industrial design processes. We give a number of examples to demonstrate the usefulness of formal methods.

- NASA used the formal verification tool SPIN [46] to verify some crucial algorithms developed for the NASA Cassini spacecraft.
- IBM has developed a hardware formal verification tool, called IBM RuleBase [14]. With the help of this tool, IBM design engineers have detected 65 bugs in AS/400 processors.
- Intel has developed a new formal specification language, ForSpec Temporal Logic [9], and utilised this language to formalise the design of some of their products.

So far, we have discussed the necessity and usefulness of formal methods in guaranteeing the correctness of system designs in general. Next, we discuss two important aspects of formal methods, namely *Formal Specification* and *Formal Verification*.

1.1.1 Formal specification

A specification of a system is a description, which explains what the system should do but not how functions of the system are achieved. Ensuring that a specification is correct is a prerequisite for obtaining a correct implementation. Commonly, a specification is described in a natural language. However, natural language is ambiguous and inaccurate compared to mathematical languages. Therefore, using natural language for specification often causes errors or results in an inconsistent specification, i.e. the requirements in the specification can not be satisfied together. In order to avoid the drawbacks of natural language in specifications, some mathematical language needs to be developed for modelling the system and describing the desired properties of the system. Usually such a language is a *formal language (logic)*[1] and a specification of a system obtained by using a formal language is a *formal specification*.

Formal specification also provides other benefits. Firstly, since formal specifications are precise and accurate, some formal methods can be applied to them to detect errors in the specifications. Secondly, there are some techniques [64], which can be used to derive an implementation from a formal specification. Lastly, using formal specification, it becomes possible to apply formal verification (discussed in the next section) in the system development process.

1.1.2 Formal verification

By formal verification we understand the process of checking whether an implementation of a system S satisfies a specification Φ_{spec}. Currently, many formal verification methods have been developed and they usually can be applied in two different ways.

Model checking

In model checking, one describes the specification of the system and the model of the system using formal specification methods. However, the specification and the model are specified by different methods, i.e. the specification Φ_{spec} is described in a formal language (logic) whereas the model is given as an interpretation M_{imp} of this logic. *Model checking* is the process of evaluating the specification Φ_{spec} in the interpretation M_{imp}. Therefore, model checking can be used to identify the flaws in system design which are not consistent with the specification.

Theorem proving

Alternatively, one may specify the model Φ_{imp} of the system and the specification Φ_{spec} of the system in the same formal language (logic) and investigate Φ_{spec} and Φ_{imp} together with a suitable calculus for such a logic. In order to guarantee that the specification holds for its implementation, one usually wants to confirm that Φ_{imp} implies Φ_{spec}, i.e. formally prove that $\Phi_{imp} \rightarrow \Phi_{spec}$ is valid. This *validity problem* can be solved using *automated theorem proving* techniques. The problem closely related to the validity problem is the *satisfiability problem*, i.e. the task of determining whether a formula Φ is satisfiable. Typically, the two problems are interreducible. A formula Φ is

[1]Some researchers also regard programming languages, hardware description languages and other mathematical languages as formal languages. In this thesis, unless indicated otherwise, formal languages are formal logics.

valid iff the formula $\neg\Phi$ is unsatisfiable. In the particular case of formal verification, $\Phi_{imp} \rightarrow \Phi_{spec}$ is valid iff $\Phi_{imp} \wedge \neg\Phi_{spec}$ is unsatisfiable.

In this thesis, we present the following two theorem proving methods for two different formal languages:

1. A clausal resolution calculus for solving the satisfiability problem of Computation Tree Logic (CTL).
   CTL was proposed in 1982 by Emerson and Clarke in [32]. It is a propositional branching-time temporal logic whose underlying model of time is a choice of possibilities branching into the future and can be used to represent and verify many systems, such as real time and concurrent systems [53].

2. A clausal resolution calculus for solving the satisfiability problem of the Next-time fragment of Alternating-time Temporal Logic (XATL).
   Alternating-time Temporal Logic (ATL) was proposed in 1997 by Alur, Henzinger and Kupferman in [5]. It is a propositional branching-time temporal logic containing temporal operators parameterised by a set of agents to express coalition abilities of agents. XATL is a non-trivial fragment of ATL that can be used to represent and verify many systems, in particular, multi-agent systems [57, 76].

1.2 Novel contributions

The novel contributions that can be found in this thesis are the following.

1. We provide a new clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL. The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is based on a previously defined calculus for CTL by Bolotov [15] but many aspects have been improved. The calculus requires a transformation of a CTL formula to a clausal normal form, called *Separated Normal Form with Global Clauses* (denoted by $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$). $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ is an extension of CTL with indexed temporal operators. We provide a formal semantics for $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ and craft a new set of transformation rules, which is shown to be more efficient than the set of transformation rules in [15] in terms of the number of extra propositions required and the number of clauses in the normal form obtained by the transformation. The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ itself consists of so-called step resolution rules and eventuality resolution rules. We show that some eventuality resolution rules in [15] are redundant. Since eventuality resolution rules are the most costly rules in resolution calculi, we gain significant efficiency. Also, our calculus makes use of an ordering $\succ$ and a selection function S to reduce applicability of the resolution rules. Finally, a new completeness proof and a complexity and termination analysis are given. The main content of this contribution is published in [78, 79, 77].

2. We have implemented our resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL in the theorem prover *CTL-RP* (short for Computation Tree Logic Resolution Prover). To our knowledge, CTL-RP is the first resolution-based theorem prover for CTL. Moreover, we provide a new technique for implementing the resolution rules in our calculus through first-order resolution. Thus, the implementation of CTL-RP is able to reuse the first-order resolution-based theorem prover

SPASS [54] to realise the calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$. Moreover, due to the reuse of SPASS, the effort to produce a theorem prover for CTL is reduced. We have conducted a number of experiments with CTL-RP which show that it is able to efficiently verify a variety of satisfiability problems encoded in CTL. It also outperforms the only other CTL theorem prover currently available, namely the CTL module of the Tableau Workbench on these problems. This contribution has been published in [79].

3. Lastly but not least, we provide a clausal resolution calculus R_{XATL} for XATL. Again, the calculus R_{XATL} requires a transformation of an XATL formula to a clausal normal form, SNF_{XATL} (short for *Separated Normal Form for XATL*). To this end, we introduce a new set of transformation rules. We prove that our transformation rules for XATL preserve satisfiability and the transformation procedure is terminating and, furthermore, can be computed in polynomial time in the size of the input formula. The calculus consists of several *step resolution* and *rewrite* rules, which are used to deal with constraints on the next moment and formula rewriting, respectively. We prove that our resolution calculus R_{XATL} is sound and outline a completeness proof.

1.3 Overview of this thesis

The main content of this thesis is as follows.

- In Chapter 2, we provide the preliminaries to our research. We present the syntax and semantics, some important properties, and resolution calculi for *Propositional Logic (PL)* and *Propositional Linear-time Temporal Logic (PLTL)*, which are the foundation of our resolution calculi in this thesis.

- Chapter 3 presents a refined clausal resolution calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ for CTL. Firstly, a clausal normal form for CTL, $\text{SNF}^{\text{g}}_{\text{CTL}}$, and its semantics are introduced. The normal form $\text{SNF}^{\text{g}}_{\text{CTL}}$ contains six different types of clauses: *initial, global,* **A***-step,* **E***-step,* **A***-sometime* and **E***-sometime clauses*. A procedure for transformation of an arbitrary CTL formula into its equi-satisfiable set of formula in $\text{SNF}^{\text{g}}_{\text{CTL}}$ is shown and proven to be correct. Then the calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ consisting of step resolution, eventuality resolution and rewrite rules is defined. We show soundness and completeness of the calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$. Furthermore, we show that given a set T of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses, the calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ will only derive a finitely bounded number of additional clauses from T. Thus, any derivation from T by $\mathsf{R}^{\succ,S}_{\text{CTL}}$ terminates. Finally, related work is discussed and conclusions are drawn.

- In Chapter 4, we discuss CTL-RP, our resolution theorem prover for CTL. We first present how we transform initial, global, **A**-step and **E**-step clauses into first-order formulae and how first-order resolution techniques are used to implement the step resolution rules for CTL. Then we present the algorithms of implementing the eventuality resolution rules for CTL and give a small example to demonstrate how the algorithms work. In addition, we establish a relationship between first-order resolution inferences and resolution inferences for CTL, which shows that our implementation is correct. Finally, we present our empirical study of CTL-RP and conclusions for this chapter.

- In Chapter 5, we provide the first clausal resolution calculus for XATL. Firstly, we give syntax and semantics of XATL and define a clausal normal form for XATL, $\mathrm{SNF}_{\mathrm{XATL}}$. We then define a transformation procedure that transforms an arbitrary XATL formulae into a set of XATL formulae in $\mathrm{SNF}_{\mathrm{XATL}}$, and prove that the transformation procedure preserves satisfiability and is terminating. Finally, we show details of our calculus, its soundness proof and the outline of its completeness proof.

- Finally, in Chapter 6, we draw the conclusions of our work and discuss potential future research.

Chapter 2

Preliminaries to resolution for branching-time temporal logic

In this chapter, we consider two resolution calculi R_{PL} and R_{PLTL} for Propositional Logic (PL) [20] and Propositional Linear-time Temporal Logic (PLTL) [59], respectively. The calculus R_{PL} [20] consists of a single inference rule that can be used to prove whether a formula of PL is satisfiable. R_{PL} is often regarded as the cornerstone of all other resolution-based proof methods for classical and non-classical logics. Based on resolution techniques, in 1991, Fisher et al. developed a clausal resolution method R_{PLTL} [37] for temporal logic. Since then, R_{PLTL} has become the foundation for many resolution methods for temporal logics and their variants including our resolution calculi for CTL and XATL.

2.1 Propositional logic

2.1.1 Syntax and semantics of propositional logic

Propositional Logic [20] is a classical logic and a foundation of many other complex logics such as PLTL, CTL and ATL. Axiom systems and proof methods related to PL have been studied extensively since the seminal work of Boole (1815-1864) [19]. In this section, we give the syntax and semantics of PL and state some properties of PL.

The fundamental building blocks of the language of PL are *propositions*, which are declarative statements, for example,

- today is Tuesday;
- it rains today;
- the product of multiplying 2 by 2 is 3; and
- the sun is bigger than the Earth.

In other words, propositions are statements that are either *true* or *false*. In PL, each proposition is represented by an individual *atomic proposition* such as p, q and r.

The language of PL is then based on

- a set of atomic propositions P_{PL};
- two *propositional constants*, **true** and **false**, also called *truth values*.
- several logical connectives, also called *boolean operators*, $\neg$ (negation), $\wedge$ (and), $\vee$ (or) and $\Rightarrow$ (implication).

The set of *well-formed* formulae of PL is inductively defined as follows:

1. **true** and **false** are PL formulae;
2. all atomic propositions in P_{PL} are PL formulae; and
3. if φ and ψ are PL formulae, then so are $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, and $(\varphi \Rightarrow \psi)$.

It should be noted that in the remainder of this thesis, $\Leftrightarrow$ (double-implication) is an abbreviation, i.e. if φ and ψ are PL formulae, then the expression $(\varphi \Leftrightarrow \psi)$ is a shorthand for the formula $((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi))$. The abbreviation $\Leftrightarrow$ can also be applied to formulae of PLTL, CTL and XATL.

Formulae of PL over P_{PL} are interpreted in *interpretations*. Each interpretation L is a function $L : \mathsf{P}_{\mathsf{PL}} \to \{\mathbf{true}, \mathbf{false}\}$ mapping each proposition in P_{PL} to either **true** or **false**.

The satisfaction relation $\models$ between an interpretation L and a formula of PL is inductively defined as follows:

$$\begin{array}{ll}
L \models \mathbf{true} & \\
L \not\models \mathbf{false} & \\
L \models p & \text{iff } L(p) = \mathbf{true} \text{ for an atomic proposition } p \in \mathsf{P}_{\mathsf{PL}} \\
L \models \neg\varphi & \text{iff } L \not\models \varphi \\
L \models (\varphi \wedge \psi) & \text{iff } L \models \varphi \text{ and } L \models \psi \\
L \models (\varphi \vee \psi) & \text{iff } L \models \varphi \text{ or } L \models \psi \\
L \models (\varphi \Rightarrow \psi) & \text{iff } L \models \neg\varphi \text{ or } L \models \psi
\end{array}$$

A formula φ of PL is *satisfiable* under an interpretation L iff $L \models \varphi$, in which case L is called a model of φ. A formula φ is called *satisfiable* iff there exists at least one interpretation L such that φ is satisfied under L. A formula φ of PL is *valid* iff for every interpretation L, $L \models \varphi$, in which case we also write $\models \varphi$.

Definition 2.1 *Equivalence*
Two formulae φ and ψ are *equivalent*, denoted by $\varphi \equiv \psi$, iff, for every model of φ is also a model of ψ and vice versa, i.e. $\models \varphi \Leftrightarrow \psi$ holds.

In the following, we introduce some very useful equivalences between PL formulae, which will be used in many proofs in the remainder of this thesis. Let φ and ψ be arbitrary PL formulae. Then the equivalences below hold.

$$\begin{array}{rclcrcl}
\neg(\varphi \wedge \psi) & \equiv & \neg\varphi \vee \neg\psi & \qquad & \neg(\varphi \Rightarrow \psi) & \equiv & \varphi \wedge \neg\psi \\
\neg(\varphi \vee \psi) & \equiv & \neg\varphi \wedge \neg\psi & & \varphi \Rightarrow \psi & \equiv & \neg\psi \Rightarrow \neg\varphi \\
\varphi \Rightarrow \psi & \equiv & \neg\varphi \vee \psi & & \neg\neg\varphi & \equiv & \varphi
\end{array}$$

The proofs for these equivalences are straightforward.

2.1.2 Resolution for propositional logic

There are many proof methods for PL, such as truth tables, tableaux, natural deduction which have been developed over many years. One of the most successful approaches is resolution [20]. The proof procedure of resolution was first proposed in [63] for first-order logic, which is more complex than PL. Here we focus on PL and present a propositional resolution calculus for it. To simplify matters, we will not consider redundancy elimination in this context. We introduce some auxiliary definitions first.

Definition 2.2 *Literal*
A *literal* is either an atomic proposition or its negation. Two literals l_1 and l_2 are complementary iff $l_1 \equiv \neg l_2$.

Definition 2.3 *Disjunction of literals*
We inductively define a *disjunction of literals* in the following:

- a literal l is a disjunction of literals; and
- if φ and ψ are disjunctions of literals, then $\varphi \vee \psi$ and $(\varphi \vee \psi)$ are also disjunctions of literals.

Definition 2.4 *PL Clause*
A *PL clause* is a set of literals $\{l_1, l_2, \ldots, l_n\}$, written as a disjunction of literals

$$l_1 \vee l_2 \vee \ldots \vee l_n$$

where for $i, 1 \leq i \leq n$, l_i is literal. The *empty clause*, denoted by $\perp$, is the empty set of literals.

Definition 2.5 *Conjunctive normal form (CNF)*
A formula φ of PL is in *conjunctive normal form* iff it is a conjunction of clauses of PL, that is, φ is of the form

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where for $i, 1 \leq i \leq n, C_i$ is a PL clause.

Theorem 2.1 *[20] For every formula φ of PL, there exists a formula ψ in CNF with $\varphi \equiv \psi$.*

The *resolution inference rule for PL* below derives a new clause, also called a *resolvent*, from two other clauses, also called *premises*, containing complementary literals:

PRES

$$\frac{C \vee l \qquad D \vee \neg l}{C \vee D}$$

where $C \vee l$ and $D \vee \neg l$ are two clauses of PL; C and D are disjunctions of literals; l and $\neg l$ is a pair of complementary literals. It should be noted that PL clauses are sets, which means that a literal l can occur at most once in a PL clause. Consequently, the resolvent of $p \vee q \vee r$ and $\neg r \vee p \vee q$ is the clause $p \vee q$ instead of $p \vee q \vee p \vee q$. The resolution calculus R_{PL} for PL consists of the single inference rule PRES.

The *resolution procedure for the validity problem of PL* is a *proof by contradiction* method. Given a formula φ of PL, it proceeds as follows.

1. Negate the formula φ;
2. Transform $\neg\varphi$ into a set T of PL clauses;
3. Construct a sequence $T_0, T_1, T_2, \ldots$ of sets of PL clauses such that (i) $T_0 = T$, (ii) $T_{i+1} = T_i \cup \{\Gamma\}$, where Γ is the resolvent of two PL clauses Γ_1 and Γ_2 in T_i, and (iii) $\Gamma \notin T_i$.
4. The construction in step 3 terminates if

   ex1 there exists an index $i, i \geq 0$ such that T_i contains the empty clause, or

   ex2 there exists an index $i, i \geq 0$ such that every resolvent of clauses in T_i is already an element of T_i.

If the algorithm above terminates by the exit condition **ex1**, then $\neg\varphi$ is unsatisfiable and, therefore, φ is valid. If, on the other hand, the algorithm terminates by the exit condition **ex2** but for all $i \geq 0, T_i$ does not contain the empty clause, then $\neg\varphi$ is satisfiable and φ is not valid. As we have mentioned earlier, resolution can also be used for satisfiability checking. In this case, the resolution procedure skips the first step, i.e. negate the formula φ. If the algorithm terminates by the exit condition **ex1**, then φ is unsatisfiable. If the algorithm terminates by the exit condition **ex2** but for all $i \geq 0, T_i$ does not contain the empty clause, then φ is satisfiable.

The resolution procedure above motivates the following definitions.

Definition 2.6 *Derivation of* R_{PL}
A *derivation* from a set T of PL clauses by R_{PL} is a sequence $T_0, T_1, T_2, \ldots$ of sets of clauses such that $T_0 = T$ and $T_{t+1} = T_t \cup \{R_t\}$ where R_t is a resolvent obtained as the conclusion of an application of the resolution rule PRES to premises in T_t.

Definition 2.7 *Refutation of* R_{PL}
A *refutation* of a set T of PL clauses by R_{PL} is a derivation from T such that, for some $i \geq 0$, T_i contains a *contradiction*, where a contradiction is an empty clause.

Definition 2.8 *Termination of a derivation*
A derivation *terminates* iff either a contradiction is derived or no new clauses can be derived by any further application of the rule PRES.

Definition 2.9 *Saturation with respect to* R_{PL}
A set T of PL clauses is *saturated with respect to* R_{PL} iff all resolvents that can be derived by an application of the rule PRES to premises in T are already contained in T.

With the definitions above, we are able to state the following two properties of R_{PL}, namely *soundness* and *completeness*.

Theorem 2.2 (Soundness and completeness of R_{PL} [63]) *Let T be a finite set of PL clauses and let $T_0, T_1, \ldots, T_n$ be a derivation from T such that $T_0 = T$ and T_n is saturated with respect to R_{PL}. Then T is unsatisfiable iff T_n contains a contradiction.*

We give a small example to show how to derive an empty clause ($\perp$) from a set of PL clauses which is unsatisfiable.

It is often convenient to use a simpler notion of a derivation. Let T be a set of PL clauses. A derivation of a clause Γ from T is a sequence $\Gamma_0, \Gamma_1, \ldots, \Gamma_n$ of PL clauses of n such that (i) $\Gamma_n = \Gamma$ and (ii) every clause $\Gamma_i, 0 \leq i \leq n$, is either an element of T or there are two indices $j, k, 0 \leq j, k \leq i$, such that Γ_i is the resolvent of Γ_j and Γ_k. The length of a derivation $\Gamma_0, \ldots, \Gamma_n$ is $n+1$. Therefore, a refutation of T can be defined as a derivation of the empty clause from T.

Theorem 2.3 *Let $\Gamma_0, \ldots, \Gamma_n$ be a derivation from a set T of PL clauses. Then a derivation in the form of Definition 2.6 can be constructed from $\Gamma_0, \ldots, \Gamma_n$.*

Proof. Given a derivation $\Gamma_0, \ldots, \Gamma_n$ from T, it is straightforward to construct a corresponding derivation $T_0, \ldots, T_m$ such that $T_0 = T$ and $\{\Gamma_0, \ldots, \Gamma_n\} \subseteq T_m$ as follows.

The construction proceeds by induction over the length $n+1$ of the derivation. For $n = 0$, the derivation consists of a single clause Γ_0 which is an element of T. The corresponding derivation consists of a single set of PL clauses $T_0 = T$. Now we assume that for a derivation $\Gamma_0, \ldots, \Gamma_n$ we have constructed a corresponding derivation $T_0, \ldots, T_m$ such that $\{\Gamma_0, \ldots, \Gamma_n\} \subseteq T_m$. Consider a derivation $\Gamma_0, \ldots, \Gamma_{n+1}$.

1. If $\Gamma_{n+1} \in T$, then $T_0, \ldots, T_m$ is also the corresponding derivation for $\Gamma_0, \ldots, \Gamma_{n+1}$. Since $T = T_0 \subseteq T_1 \subseteq \ldots \subseteq T_m$ and by induction hypothesis $\{\Gamma_0, \ldots, \Gamma_n\} \subseteq T_m$, we have $\Gamma_0, \ldots, \Gamma_n, \Gamma_{n+1} \subseteq T_m$.

2. If Γ_{n+1} has been derived from Γ_i and Γ_j with $i, j \leq n$, then by induction hypothesis, $\Gamma_i, \Gamma_j \in T_m$. We can thus derive Γ_{n+1} from T_m and define $T_{m+1} = T_m \cup \{\Gamma_{n+1}\}$. Obviously, $\{\Gamma_0, \ldots, \Gamma_n, \Gamma_{n+1}\} \subseteq T_{m+1}$.

□

Example 2.1
Let T be a set of PL clauses $\{p \vee q, \neg q \vee r, \neg r, \neg p\}$. Then the following is a derivation by R_{PL} from

T.

1. $p \vee q$	
2. $\neg q \vee r$	
3. $\neg r$	
4. $\neg p$	
5. $p \vee r$	$[1, 2, \mathrm{PRES}]$
6. p	$[3, 5, \mathrm{PRES}]$
7. $\perp$	$[4, 6, \mathrm{PRES}]$

where $[c_1, c_2, \mathrm{PRES}]$ indicates the application of the rule PRES to clauses c_1 and c_2.

2.2 Propositional linear-time temporal logic (PLTL)

Propositional Linear-time Temporal Logic [59] is a modal logic intended to represent and reason about the changing truth values of assertions over time. The flow of time underlying the semantics of PLTL is discrete and linear, i.e. every moment of time has exactly one successive future moment. Applications of PLTL include reasoning about program [59], temporal databases [68], hardware verification [53, 49]. Whether these applications can be successful often depends on how efficient the satisfiability problem for PLTL can be addressed. Therefore, to tackle this problem, many approaches have been developed, for example tableau-based [50, 2], automata-based [66] and non-clausal resolution-based methods [1] for PLTL. In this section we discuss one of the resolution-based methods, namely a clausal resolution calculus $\mathsf{R}_{\mathrm{PLTL}}$, introduced by Fisher et al. [38] in 1991. This resolution calculus is the first clausal resolution calculus for temporal logic. It forms the foundation for many other resolution calculi developed later for more complex logics including two calculi we will introduce in this thesis. Many important terminologies used by our calculi are originated from $\mathsf{R}_{\mathrm{PLTL}}$, for example, the special propositional constant **start**, Separated Normal Form (SNF), step resolution, eventuality resolution, loops, augmentation, behaviour graphs and so on. Therefore, we give a brief introduction of $\mathsf{R}_{\mathrm{PLTL}}$ as the preliminary to our CTL resolution calculus.

Next, we show the syntax and semantics of PLTL and then present the calculus $\mathsf{R}_{\mathrm{PLTL}}$. Again, to simplify matters, we will not consider redundancy elimination in this context.

2.2.1 Syntax and semantics of PLTL

Our presentation of the syntax and semantics of PLTL follows Fisher et al. [38], which is different, but equivalent to the one [31], which is commonly used.

The language of PLTL is based on

- a set of atomic propositions P_{PL};
- two propositional constants, **true** and **false**;
- boolean operators, $\neg$ (negation), $\wedge$ (and), $\vee$ (or) and $\Rightarrow$ (implication); and
- temporal operators, $\bigcirc$ (at the next moment), $\Diamond$ (eventually in the future), $\Box$ (always in the future), $\mathcal{U}$ (until) and $\mathcal{W}$ (unless).

The set of *well-formed* formulae of PLTL is inductively defined as follows:

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

Formulae of PLTL over P_{PL} are interpreted over model structures, $\sigma = \langle S, x, L\rangle$, where

1. S is a set of states;
2. $x : \mathbb{N}_0 \rightarrow S$ is an infinite sequence of states $s_0, s_1, s_2, \ldots$; and
3. $L : S \rightarrow 2^{\mathsf{P}_{\mathsf{PL}}}$ is an interpretation function mapping each state $s \in S$ to the set of atomic propositions true at state s.

The satisfaction relation $\models$ between a pair $\langle\sigma, i\rangle$, consisting of a model structure σ and a natural number i, and a PLTL formula is inductively defined as follows:

$$
\begin{array}{ll}
\langle\sigma, i\rangle \models \textbf{true} & \\
\langle\sigma, i\rangle \not\models \textbf{false} & \\
\langle\sigma, i\rangle \models p & \text{iff } p \in \mathsf{P}_{\mathsf{PL}} \text{ and } p \in L(x(i)) \\
\langle\sigma, i\rangle \models \neg\varphi & \text{iff } \langle\sigma, i\rangle \not\models \varphi \\
\langle\sigma, i\rangle \models (\varphi \wedge \psi) & \text{iff } \langle\sigma, i\rangle \models \varphi \text{ and } \langle\sigma, i\rangle \models \psi \\
\langle\sigma, i\rangle \models (\varphi \vee \psi) & \text{iff } \langle\sigma, i\rangle \models \varphi \text{ or } \langle\sigma, i\rangle \models \psi \\
\langle\sigma, i\rangle \models (\varphi \Rightarrow \psi) & \text{iff } \langle\sigma, i\rangle \models \neg\varphi \text{ or } \langle\sigma, i\rangle \models \psi \\
\langle\sigma, i\rangle \models \bigcirc\varphi & \text{iff } \langle\sigma, i+1\rangle \models \varphi \\
\langle\sigma, i\rangle \models \Diamond\varphi & \text{iff there exists a } k \in \mathbb{N} \text{ such that } k \geq i \text{ and } \langle\sigma, k\rangle \models \varphi \\
\langle\sigma, i\rangle \models \Box\varphi & \text{iff for all } j, j \geq i,\ \langle\sigma, j\rangle \models \varphi \\
\langle\sigma, i\rangle \models (\varphi\,\mathcal{U}\,\psi) & \text{iff there exists a } k \in \mathbb{N} \text{ such that } k \geq i \text{ and } \langle\sigma, k\rangle \models \psi; \\
 & \quad \text{and for all } j, i \leq j < k,\ \langle\sigma, j\rangle \models \varphi \\
\langle\sigma, i\rangle \models (\varphi\,\mathcal{W}\,\psi) & \text{iff } \langle\sigma, i\rangle \models \varphi\,\mathcal{U}\,\psi \text{ or } \langle\sigma, i\rangle \models \Box\varphi
\end{array}
$$

A formula φ of PLTL is *satisfiable* in a model structure σ at the state s_i iff $\langle\sigma, i\rangle \models \varphi$. The formula φ is *satisfiable* iff there exists at least one model structure σ such that φ is satisfied in σ at s_0. A formula φ of PLTL is *valid* iff, for every model structure σ, φ is satisfied in σ at s_0, in which case we also write $\models \varphi$.

2.2.2 Resolution for PLTL

$\mathsf{R}_{\mathrm{PLTL}}$ operates on PLTL formulae in a certain normal form, called Separated Normal Form (SNF). This normal form consists of three types of clauses of the following forms:

$$
\begin{array}{ll}
\Box(\textbf{start} \Rightarrow \bigvee_{j=1}^{k} m_j) & \text{(initial clause)} \\
\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \bigcirc \bigvee_{j=1}^{k} m_j) & \text{(step clause)} \\
\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \Diamond l) & \text{(sometime clause)}
\end{array}
$$

where

1. **start** is a propositional constant that holds only at the beginning of time, i.e. $\langle\sigma, i\rangle \models \mathbf{start}$ iff $i = 0$;
2. $k \geq 0$, $n \geq 0$;
3. each $l_i, 1 \leq i \leq n$, is a literal and for any two literals $l_i, l_j, 1 \leq i < j \leq n, l_i \neq l_j$;
4. each $m_j, 1 \leq j \leq k$, is a literal and for any two literals $m_i, m_j, 1 \leq i < j \leq k, m_i \neq m_j$; and
5. l is a literal.

We use $\square(\mathbf{true} \Rightarrow \bigcirc \bigvee_{j=1}^{k} m_j)$ to denote a SNF clause $\square(\bigwedge_{i=1}^{n} l_i \Rightarrow \bigcirc \bigvee_{j=1}^{k} m_j)$ such that $n = 0$. Moreover, we use $\square(\bigwedge_{i=1}^{n} l_i \Rightarrow \bigcirc\mathbf{false})$ to denote a SNF clause $\square(\bigwedge_{i=1}^{n} l_i \Rightarrow \bigcirc \bigvee_{j=1}^{k} m_j)$ such that $k = 0$. Likewise, we use $\square(\mathbf{start} \Rightarrow \mathbf{false})$ to denote $\square(\mathbf{start} \Rightarrow \bigvee_{j=1}^{k} m_j)$ with $k = 0$. It is easy to see that $\square(\mathbf{start} \Rightarrow \mathbf{false})$ is not satisfiable. Thus, $\square(\mathbf{start} \Rightarrow \mathbf{false})$ is a *contradiction*. For a SNF clause $\Gamma = \square(P \Rightarrow D)$, we call P the left-hand side of Γ and D the right-hand side of Γ. As all clauses are of the form $\square(P \Rightarrow D)$, we often simply write $P \Rightarrow D$ instead. A formula $\diamond l$ is called an *eventuality*. Fisher et al. [38] define a procedure τ that transforms any PLTL formula φ into a set of SNF clauses $\tau(\varphi)$ and prove the following result.

Theorem 2.4 *[38] A formula φ of PLTL is satisfiable iff $\tau(\varphi)$ is satisfiable.*

The calculus $\mathsf{R}_{\mathrm{PLTL}}$ consists of two *step resolution rules* LSRES1 and LSRES2, one *eventuality resolution rule* LERES and one *rewrite rule* LRW. Firstly, we present the step resolution rules, which resolve two SNF clauses containing complementary literals on their right-hand sides. In the following, C and D are disjunctions of literals; P and Q are conjunctions of literals; and l is a literal.

LSRES1

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

LSRES2

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

Secondly, $\mathsf{R}_{\mathrm{PLTL}}$ contains a rewrite rule LRW, which rewrites a step clause with $\bigcirc\mathbf{false}$ as its right-hand side into an initial clause and a step clause.

$$\textbf{LRW} \quad \bigwedge_{i=1}^{n} m_i \Rightarrow \bigcirc\mathbf{false} \quad \longrightarrow \quad \left\{ \begin{array}{l} \mathbf{start} \Rightarrow \bigvee_{i=1}^{n} \neg m_i \\ \mathbf{true} \Rightarrow \bigcirc(\bigvee_{i=1}^{n} \neg m_i) \end{array} \right.$$

Finally, $\mathsf{R}_{\mathrm{PLTL}}$ contains a eventuality resolution rule LERES, which resolves formulae containing the temporal operators $\square$ and $\diamond$, respectively.

LERES

$$\begin{array}{c} P^{\dagger} \Rightarrow \bigcirc\square l \\ Q \Rightarrow \diamond\neg l \\ \hline Q \Rightarrow (\neg(P^{\dagger})\, \mathcal{W} \neg l) \end{array}$$

where $P^\dagger \Rightarrow \bigcirc\Box l$ represents a set of step clauses that together implies $\bigcirc\Box l$. It should be noted that the resolvent of LERES $Q \Rightarrow (\neg(P^\dagger)\,\mathcal{W}\,\neg l)$ is not a formula in SNF and, therefore, it will be transformed into SNF by the rules defined in the transformation procedure τ.

The most complex part of LERES is to find a suitable set of step clauses as premise of LERES and the algorithm for finding these step clauses is called *loop search algorithm*. Detailed descriptions of various loop search algorithms and their correctness proofs are given in [25, 26, 27].

Note that, for all resolvents derived by the resolution rules in $\mathsf{R}_{\mathrm{PLTL}}$, there are no duplicate literals on the left-hand side and the right-hand side. For instance, the resolvent of two SNF clauses $r \wedge p \Rightarrow \bigcirc(p \vee q \vee r)$ and $r \wedge q \Rightarrow \bigcirc(p \vee q \vee \neg r)$ is $r \wedge p \wedge q \Rightarrow \bigcirc(p \vee q)$. In analogy to the notions of a derivation and a refutation we have introduced for the calculus R_{PL}, we define the notions of a derivation and a refutation of the calculus $\mathsf{R}_{\mathrm{PLTL}}$ as follows.

Definition 2.10 *Derivation of* $\mathsf{R}_{\mathrm{PLTL}}$
A *derivation* from a set T of SNF clauses by $\mathsf{R}_{\mathrm{PLTL}}$ is a sequence $T_0, T_1, T_2, \ldots$ of sets of SNF clauses such that $T = T_0$ and $T_{t+1} = T_t \cup R_t$ where R_t is a set of SNF clauses obtained as the conclusion of an application of a rule of $\mathsf{R}_{\mathrm{PLTL}}$ to premises in T_t.

Definition 2.11 *Refutation of* $\mathsf{R}_{\mathrm{PLTL}}$
A *refutation* of a set T of SNF clauses by $\mathsf{R}_{\mathrm{PLTL}}$ is a derivation from T such that, for some $i \geq 0$, T_i contains a *contradiction*.

Definition 2.12 *Termination of a derivation*
A derivation *terminates* iff either a contradiction is derived or no new clauses can be derived by further applications of the resolution rules in $\mathsf{R}_{\mathrm{PLTL}}$.

Definition 2.13 *Saturation with respect to* $\mathsf{R}_{\mathrm{PLTL}}$
A set T of SNF clauses is *saturated with respect to* $\mathsf{R}_{\mathrm{PLTL}}$ iff all resolvents that can be derived by an application of a rule of $\mathsf{R}_{\mathrm{PLTL}}$ to premises in T are already contained in T.

Theorem 2.5 (Soundness and completeness of $\mathsf{R}_{\mathrm{PLTL}}$ [38]) *Let T be a finite set of SNF clauses and let $T_0, \ldots, T_n$ be a derivation from T such that $T_0 = T$ and T_n is saturated with respect to $\mathsf{R}_{\mathrm{PLTL}}$. Then T is unsatisfiable iff T_n contains a contradiction.*

We provide an example to demonstrate how a refutation can be found by applying the resolution rules in $\mathsf{R}_{\mathrm{PLTL}}$ to an unsatisfiable set of SNF clauses.

Example 2.2
Let T be a set of SNF clauses $\{\mathbf{start} \Rightarrow p \vee q,\ q \Rightarrow \bigcirc r,\ \mathbf{true} \Rightarrow \bigcirc\neg r,\ \mathbf{start} \Rightarrow \neg p\}$. Then the

derivation by $\mathsf{R}_{\mathrm{PLTL}}$ from T is as follows.

$$\begin{array}{rrcll}
1. & \mathbf{start} & \Rightarrow & p \lor q & \\
2. & q & \Rightarrow & \bigcirc r & \\
3. & \mathbf{true} & \Rightarrow & \bigcirc \neg r & \\
4. & \mathbf{start} & \Rightarrow & \neg p & \\
5. & q & \Rightarrow & \bigcirc \mathbf{false} & [2, 3, \mathrm{LSRES2}] \\
6. & \mathbf{start} & \Rightarrow & \neg q & [5, \mathrm{LRW}] \\
7. & \mathbf{true} & \Rightarrow & \bigcirc \neg q & [5, \mathrm{LRW}] \\
8. & \mathbf{start} & \Rightarrow & p & [1, 6, \mathrm{LSRES1}] \\
9. & \mathbf{start} & \Rightarrow & \mathbf{false} & [4, 8, \mathrm{LSRES1}]
\end{array}$$

where (i) $[c_1, c_2, \mathrm{LSRESi}]$ indicates the application of the ith step resolution rule to clauses c_1 and c_2; and (ii) $[c, \mathrm{LRW}]$ indicates the application of the rewrite rule LRW to clauses c.

Chapter 3

A refined resolution calculus for CTL

3.1 Introduction

Computation Tree Logic (CTL) [22] is a propositional branching-time temporal logic whose underlying model of time is a choice of possibilities branching into the future. There are many important applications that can be represented in and reasoned about in CTL such as the verification of digital circuits [24], analysis of real time and concurrent systems [53], XPath query processing [4], communication protocol verification [23], and Grid Component system verification [13].

As we can see from Section 1.1, there are many considerable advantages in being able to formally verify the correctness of computer-based systems. Furthermore, CTL can be used to represent a large number of systems. Therefore, it is definitely worth developing a reasoning procedure for CTL. In the following, we discuss a concrete example of communication protocol verification to demonstrate the use of CTL.

Alternating Bit Protocol, which is originally proposed in [12], is a network protocol and has been applied widely in the real world. The Alternating Bit Protocol (ABP) 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, which means that

1. the communication channel may lose messages; but

2. does not lose infinitely many messages, in other words, if a message is sent often enough, it will eventually reach its destination.

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.

The Receiver works in a complementary way. Initially the Receiver is waiting for messages. Once it receives a message with a control bit 0, it starts to repeatedly send the acknowledgement with the control bit 0. The Receiver will repeatedly send the acknowledgement including the control bit until it receives a message from the Transmitter with a complement control bit. The Receiver will then alternate the control bit and start sending the acknowledge including the new control bit.

This protocol can be represented formally in CTL and then we can reason about it using CTL theorem proving techniques. For example, we can prove that whether ABP has the following properties.

1. In all the possible future paths, the Receiver do not always send the acknowledgement with the control bit 0.

2. There exists a future path such that on this path whenever the Transmitter sends the message with the control bit 1, then eventually it receives the acknowledgement with the control bit 1.

Assume the CTL formula Φ_{spec} is the specification of ABP and Φ_{P_1}, Φ_{P_2} are the two properties above described in CTL, respectively. Using CTL theorem proving, we can formally prove whether $\Phi_{spec} \Rightarrow (\Phi_{P_1} \wedge \Phi_{P_2})$ is valid. If it is, we know that ABP has these two properties. Otherwise, it does not have. (After the language CTL, our resolution calculus for CTL and its implementation are formally explained, in Section 4.4 we will revisit this example in more details including how to translate the specification of ABP in English to the one in CTL.)

The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL introduced in this chapter is a refinement of an earlier resolution calculus [15] for CTL. The overall approach involves transformation to a normal form, called Separated Normal Form with Global Clauses for CTL, $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ for short, and the applications of *step* and *eventuality* resolution rules in $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ that deal with constraints on next states and on future states, respectively. We have improved the earlier calculus [15] in the following aspects. A technique introduced in [15] is the use of indices as part of a CTL normal form. We give a formal interpretation of indices and a formal semantics for the indexed normal form, $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$, which is missing from [15]. An ordering and a selection function are introduced into the calculus which allow us to reduce the number of possible applications of the inference rules during proof search. We show that our calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is sound, complete and terminating. Using our completeness proof we can show that two eventuality resolution rules in [15] are redundant. A detailed complexity analysis of the calculus is provided, which is absent for the earlier calculus. Finally, we have implemented $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ in our theorem prover CTL-RP whereas no implementation was provided for the earlier calculus in [15]. We also present the details of our prover CTL-RP and some experimental results in next chapter. It should be noted that the most of the content related to the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ in this chapter has been published in [78].

The rest of this chapter is organised as follows. We first present the syntax and semantics of CTL in Section 3.2 and then introduce a normal form for CTL, $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$, in Section 3.3. In Section 3.4 the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is presented. We provide proofs for soundness and completeness of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ in Section 3.5. In Section 3.6 we discuss the complexity of our calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. Finally, related work is discussed in Section 3.7 and conclusions are drawn in Section 3.8.

3.2 Syntax and semantics of CTL

The language of CTL is based on

- a set of atomic propositions $\mathsf{P_{PL}}$;
- propositional constants, **true** and **false**;
- boolean operators, $\wedge, \vee, \Rightarrow$, and $\neg$ ($\wedge$ and $\vee$ are associative and commutative); and
- 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); and the *universal path quantifier* **A** (for all future paths) and the *existential path quantifier* **E** (for some future path).

The set of *(well-formed) formulae of CTL* is inductively defined as follows:

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

Formulae of CTL over $\mathsf{P_{PL}}$ are typically 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 $L : S \rightarrow 2^{\mathsf{P_{PL}}}$ is an *interpretation function* mapping each state to the set of atomic propositions true at that state. These model structures are not required to be tree structures. However, CTL formulae can also be interpreted in tree model structures, which will be introduced later.

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 $j \geq i, (s_j, s_{j+1}) \in R$. A state $s' \in S$ is *reachable* from the state $s \in S$ iff there exists an infinite path χ_s such that $s' \in \chi_s$. If there exists two states $s, s' \in S$ such that $(s, s') \in R$, we say that s is a *predecessor* of s' and s' is a *successor* of s.

The satisfaction relation $\models$ between a pair consisting of a model structure M and a state $s_i \in S$, and a CTL formula is inductively defined 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 $\langle M, s_i\rangle \not\models \varphi$
$\langle M, s_i\rangle \models (\varphi \wedge \psi)$	iff $\langle M, s_i\rangle \models \varphi$ and $\langle M, s_i\rangle \models \psi$
$\langle M, s_i\rangle \models (\varphi \vee \psi)$	iff $\langle M, s_i\rangle \models \varphi$ or $\langle M, s_i\rangle \models \psi$
$\langle M, s_i\rangle \models (\varphi \Rightarrow \psi)$	iff $\langle M, s_i\rangle \not\models \varphi$ or $\langle M, s_i\rangle \models \psi$
$\langle M, s_i\rangle \models \mathbf{E}\bigcirc\psi$	iff there exists a path χ_{s_i} such that $\langle M, s_{i+1}\rangle \models \psi$

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

In addition, we use the following equivalences to define the remaining operators of CTL.

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

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

The satisfiability problem of CTL is known to be EXPTIME-complete [22, 31, 34].

Whether R is any total binary relation or a tree does not affect the set of valid CTL formulae [31, 33]. Therefore, in the following we restrict ourselves to model structures $M = \langle S, R, L, s_0 \rangle$ such that

- there exists a unique state s_0, called the *root*, such that every state $s \in S$ is reachable from state s_0 and there are no predecessors of s_0;
- for every state $s \in S$ except the root, state s has exactly one predecessor;
- for every infinite path χ_{s_0} and for every $i, j, 0 \leq i < j$, $s_i \neq s_j$.

For model structures $M = \langle S, R, L, s_0 \rangle$, it is also convenient to use definition of satisfiability and validity which are slightly different from Emerson's definition that we present earlier. In particular, we say a 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 s_0 is a model of φ. A CTL formula φ is valid iff for every model structure $M = \langle S, R, L, s_0 \rangle$, $M, s_0 \models \varphi$. Thus, our definition requires that if M is a model of φ, then φ must be satisfied at the root of M, namely state s_0, whereas Emerson's definition allows φ to be satisfied at any state of M. It is not hard to see that there exists a model for a CTL formula φ according to Emerson's definition iff there exists a model for φ according to our definition. The reason we add this restriction is that it can simplify the proof that our transformation rules preserve satisfiability.

3.3 Normal form

Our calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ operates on formulae in a clausal normal form, called Separated Normal Form with Global Clauses for CTL, denoted by $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$. The language of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is defined over an extension of CTL. That is the language is based on

- the language of CTL;
- a propositional constant **start**; and
- 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 five precedence groups, where (i) is highest and (v) is lowest: (i) $\mathbf{A}\bigcirc, \mathbf{E}\bigcirc, \mathbf{A}\Diamond, \mathbf{E}\Diamond, \mathbf{A}\Box, \mathbf{E}\Box, \mathbf{A}\,\mathcal{U}, \mathbf{E}\,\mathcal{U}, \mathbf{A}\,\mathcal{W}, \mathbf{E}\,\mathcal{W}, \mathbf{E}\bigcirc_{\langle ind\rangle}, \mathbf{E}_{\langle ind\rangle}\Diamond, \mathbf{E}_{\langle ind\rangle}\Box$, $\mathbf{E}_{\langle ind\rangle}\,\mathcal{U}$, $\mathbf{E}_{\langle ind\rangle}\,\mathcal{W}$, where $ind \in \mathsf{Ind}$; (ii) $\neg$; (iii) $\wedge$; (iv) $\vee$; and (v) $\Rightarrow$. Two operators in the same group have the same precedence. Higher precedence operators are applied before lower precedence operators. Then the language of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ *clauses* consists of formulae of the following forms:

$\mathbf{A}\Box(\mathbf{start} \Rightarrow \bigvee_{j=1}^{k} m_j)$	(initial clause)
$\mathbf{A}\Box(\mathbf{true} \Rightarrow \bigvee_{j=1}^{k} m_j)$	(global clause)
$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{A}\bigcirc \bigvee_{j=1}^{k} m_j)$	(**A**-step clause)
$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc \bigvee_{j=1}^{k} m_j)$	(**E**-step clause)
$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{A}\Diamond l)$	(**A**-sometime clause)
$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond l)$	(**E**-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. As all clauses are of the form $\mathbf{A}\Box(P \Rightarrow D)$, we often simply write $P \Rightarrow D$ instead. We assume that all $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses are always kept in condensed form, i.e. there are no duplicate literals in P or D. For example, a $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clause $r \wedge q \wedge q \Rightarrow \mathbf{A}\bigcirc(q \vee p \vee q)$ is always represented as $r \wedge q \Rightarrow \mathbf{A}\bigcirc(q \vee p)$. We call a clause which is either an initial, a global, an **A**-step, or an **E**-step clause a *determinate clause*. The formula $\mathbf{A}\Diamond l$ is called an **A**-*eventuality* and the formula $\mathbf{E}_{\langle ind\rangle}\Diamond l$ is called an **E**-*eventuality*.

The most important elements in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ are indices. These indices can be used to preserve a certain path context. For example,

- the formula $\mathbf{E}\bigcirc p \wedge \mathbf{E}\bigcirc\neg p$ is obviously satisfiable, as in general, these two existential path quantifiers can refer to two different paths; whereas
- the formula $\mathbf{E}_{\langle ind\rangle}\bigcirc p \wedge \mathbf{E}_{\langle ind\rangle}\bigcirc\neg p$ is unsatisfiable, as two existential quantifiers having the same index *ind* indicate p and $\neg p$ are satisfied at the same successor state of the current state.

Informally speaking, in the case that an index *ind* is associated with a next operator ($\bigcirc$), *ind* helps identify a particular successor state. For instance, if $\mathbf{E}_{\langle ind\rangle}\bigcirc p$ is satisfied at a state s, then there exists a state s' such that the edge from s to s' is labelled by *ind* and p holds at s'. In the other case that an index *ind* is associated with a long-term operator ($\Diamond, \Box, \mathcal{U}$ or $\mathcal{W}$), *ind* helps identify a particular path. For instance, if $\mathbf{E}_{\langle ind\rangle}\Diamond p$ is satisfied at state s, then there exists a state s' reachable from s such that each edge in the path from s to s' is labelled by *ind* and p holds at s'. This ability of indices is necessary for our transformation rule to preserve satisfiability.

3.3.1 Syntax and semantics of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$

To provide a semantics for $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$, we extend model structures $\langle S, R, L, s_0\rangle$ to $\langle S, R, L, [\_], s_0\rangle$ where $[\_] : \mathsf{Ind} \rightarrow 2^{(S\times S)}$ 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 exactly one state $s' \in S, (s, s') \in [ind]$ and $(s, s') \in R$. A state $s' \in S$ is an *ind-successor state* of state $s \in S$ iff $(s, s') \in [ind]$. An *infinite path* $\chi^{\langle ind\rangle}_{s_i}$ is an infinite sequence of states $s_i, s_{i+1}, s_{i+2}, \ldots$ such that for every $j \geq i, (s_j, s_{j+1}) \in [ind]$. An infinite path $\chi^{\langle ind\rangle}_{s_k} = s'_k, s'_{k+1}, \ldots$ is a *subpath* of $\chi^{\langle ind\rangle}_{s_i}$ iff there exists a natural number $l, l \geq i$ such that for every $j \geq 0$, $s'_{k+j} = s_{l+j}$. Note that since $[ind]$ is a function, for every state $s_i \in S$, there exists exactly one infinite path $\chi^{\langle ind\rangle}_{s_i}$ and for every state $s_j \in \chi^{\langle ind\rangle}_{s_i}$, $\chi^{\langle ind\rangle}_{s_j}$ is a subpath of $\chi^{\langle ind\rangle}_{s_i}$. The semantics of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ is then defined as shown below as an extension of the semantics of CTL defined in Section 3.2. Although the operators $\mathbf{E}_{\langle ind\rangle}\Box$, $\mathbf{E}_{\langle ind\rangle}\,\mathcal{U}$ and $\mathbf{E}_{\langle ind\rangle}\,\mathcal{W}$ do not appear in the normal form, we state their semantics, because they occur in the normal form transformation. (The semantics of the remaining operators is analogous to that given previously but in the extended model structure $\langle S, R, L, [\_], s_0\rangle$.)

$$
\begin{array}{ll}
\langle M, s_i\rangle \models \mathbf{start} & \text{iff } s_i = s_0 \\
\langle M, s_i\rangle \models \mathbf{E}_{\langle ind\rangle}\bigcirc\psi & \text{iff for the path } \chi^{\langle ind\rangle}_{s_i},\ \langle M, s_{i+1}\rangle \models \psi \\
\langle M, s_i\rangle \models \mathbf{E}_{\langle ind\rangle}\Diamond\psi & \text{iff } \langle M, s_i\rangle \models \mathbf{E}_{\langle ind\rangle}(\mathbf{true}\,\mathcal{U}\,\psi) \\
\langle M, s_i\rangle \models \mathbf{E}_{\langle ind\rangle}\Box\psi & \text{iff for every } s_j \in \chi^{\langle ind\rangle}_{s_i},\ \langle M, s_j\rangle \models \psi \\
\langle M, s_i\rangle \models \mathbf{E}_{\langle ind\rangle}(\varphi\,\mathcal{U}\,\psi) & \text{iff there exists } s_j \in \chi^{\langle ind\rangle}_{s_i} \text{ such that } \langle M, s_j\rangle \models \psi \text{ and} \\
 & \quad \text{for every } s_k \in \chi^{\langle ind\rangle}_{s_i}, \text{ if } i \leq k < j, \text{ then } \langle M, s_k\rangle \models \varphi \\
\langle M, s_i\rangle \models \mathbf{E}_{\langle ind\rangle}(\varphi\,\mathcal{W}\,\psi) & \text{iff } \langle M, s_i\rangle \models \mathbf{E}_{\langle ind\rangle}\Box\varphi \text{ or } \langle M, s_i\rangle \models \mathbf{E}_{\langle ind\rangle}(\varphi\,\mathcal{U}\,\psi)
\end{array}
$$

A $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ 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 $s_0 \in S$ is called a *model* of φ and we say that M satisfies φ. A $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ formula φ is *valid*, written $\models \varphi$, iff for every model structure $M = \langle S, R, L, [\_], s_0\rangle, M, s_0 \models \varphi$.

Figure 3.1 and Figure 3.2 depict example model structures satisfying the formulae $\mathbf{E}_{\langle ind_1\rangle}\bigcirc p$ and $\mathbf{E}_{\langle ind_1\rangle}\Diamond p$, respectively.

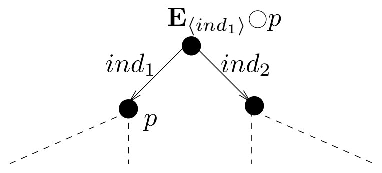


Figure 3.1: An example model structure of $\mathbf{E}_{\langle ind_1\rangle}\bigcirc p$

3.3.2 Transformation

We first introduce definitions of *indexed CTL formula* and *CTL clauses*, which will be used in our definition of the transformation from an arbitrary CTL formula into a set of formulae in normal form.

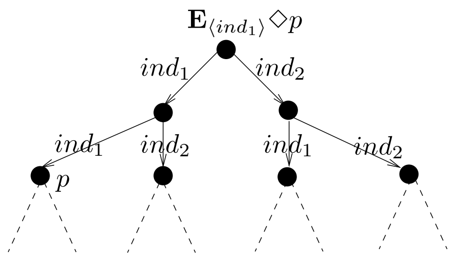


Figure 3.2: An example model structure of $\mathbf{E}_{\langle ind_1 \rangle}\Diamond p$

Definition 3.1 *Indexed CTL formula*
The set of *indexed CTL formulae* is inductively defined as follows:

1. **true**, **false** and **start** are indexed CTL formulae;

2. all atomic propositions in $\mathsf{P_{PL}}$ are indexed CTL formulae; and

3. if φ and ψ are indexed CTL formulae, then so are $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$, $\mathbf{A}\Box\varphi$, $\mathbf{A}\Diamond\varphi$, $\mathbf{A}\bigcirc\varphi$, $\mathbf{A}(\varphi\,\mathcal{U}\,\psi)$, $\mathbf{A}(\varphi\,\mathcal{W}\,\psi)$, $\mathbf{E}\Box\varphi$, $\mathbf{E}\Diamond\varphi$, $\mathbf{E}\bigcirc\varphi$, $\mathbf{E}(\varphi\,\mathcal{U}\,\psi)$, $\mathbf{E}(\varphi\,\mathcal{W}\,\psi)$, $\mathbf{E}_{\langle ind \rangle}\Box\varphi$, $\mathbf{E}_{\langle ind \rangle}\Diamond\varphi$, $\mathbf{E}_{\langle ind \rangle}\bigcirc\varphi$, $\mathbf{E}_{\langle ind \rangle}(\varphi\,\mathcal{U}\,\psi)$, and $\mathbf{E}_{\langle ind \rangle}(\varphi\,\mathcal{W}\,\psi)$, where *ind* is an arbitrary index in Ind.

Definition 3.2 *CTL clauses*
A CTL formula of the form $\mathbf{A}\Box(P \Rightarrow \varphi)$, where P is a conjunction of literals (possibly consisting of a single literal) or a propositional constant and φ is an arbitrary indexed CTL formula, is a *CTL clause* or a *clause*.

We now define 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.

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

$$
\begin{array}{cc}
(\varphi \wedge \mathbf{true}) \longrightarrow \varphi & (\varphi \wedge \mathbf{false}) \longrightarrow \mathbf{false} \\
(\varphi \vee \mathbf{true}) \longrightarrow \mathbf{true} & (\varphi \vee \mathbf{false}) \longrightarrow \varphi \\
\neg\mathbf{true} \longrightarrow \mathbf{false} & \neg\mathbf{false} \longrightarrow \mathbf{true}
\end{array}
$$

where φ is a CTL formula and $\vee$ and $\wedge$ are commutative and associative, plus the following rules which are based on the equivalences in [31].

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

where $\mathbf{P} \in \{\mathbf{A}, \mathbf{E}\}$ and $* \in \{\bigcirc, \Box, \Diamond\}$.

Let $\mathit{init}(\varphi)$ be the set of CTL clauses $\{\mathbf{A}\Box(\mathbf{start} \Rightarrow p), \mathbf{A}\Box(p \Rightarrow \mathit{simp}(\mathit{nnf}(\varphi)))\}$, where p is a new atomic proposition in $\mathsf{P_{PL}}$ that does not occur in φ.

Then the transformation of an arbitrary CTL formula φ into $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ consists of a sequence $T_0, T_1, \ldots, T_n$ of sets of CTL clauses such that (i) $T_0 = \mathit{init}(\varphi)$ and (ii) for every $t, 0 \leq t < n, T_{t+1} = (T_t \setminus \{\psi\}) \cup R_t$, where ψ is a formula in T_t not in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ and R_t is the result of applying a matching transformation rule to ψ. Moreover, for every $t, 0 \leq t < n$, T_t contains at least one formula not in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ while all formulae in T_n are in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$.

Note that for each rule of *Trans* containing a proposition p, p represents a new atomic proposition in $\mathsf{P_{PL}}$ which does not occur in T_t when we apply the rule to a clause in T_t. Furthermore, in the presentation of the rules, let

- q be an atomic proposition,
- l be a literal,
- 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:

$$\begin{array}{lc}
\mathit{Trans}(1) & q \Rightarrow \mathbf{E}\bigcirc\varphi \longrightarrow q \Rightarrow \mathbf{E}_{\langle \mathit{ind} \rangle}\bigcirc\varphi \\
\mathit{Trans}(2) & q \Rightarrow \mathbf{E}\Diamond\varphi \longrightarrow q \Rightarrow \mathbf{E}_{\langle \mathit{ind} \rangle}\Diamond\varphi \\
\mathit{Trans}(3) & q \Rightarrow \mathbf{E}\Box\varphi \longrightarrow q \Rightarrow \mathbf{E}_{\langle \mathit{ind} \rangle}\Box\varphi \\
\mathit{Trans}(4) & q \Rightarrow \mathbf{E}(\varphi_1 \mathcal{U}\, \varphi_2) \longrightarrow q \Rightarrow \mathbf{E}_{\langle \mathit{ind} \rangle}(\varphi_1 \mathcal{U}\, \varphi_2) \\
\mathit{Trans}(5) & q \Rightarrow \mathbf{E}(\varphi_1 \mathcal{W}\, \varphi_2) \longrightarrow q \Rightarrow \mathbf{E}_{\langle \mathit{ind} \rangle}(\varphi_1 \mathcal{W}\, \varphi_2)
\end{array}$$

  where *ind* is a new index.

- Boolean rules:

$$\mathit{Trans}(6) \quad q \Rightarrow \varphi_1 \wedge \varphi_2 \longrightarrow \begin{cases} q \Rightarrow \varphi_1 \\ q \Rightarrow \varphi_2 \end{cases}$$

$$\mathit{Trans}(7) \quad q \Rightarrow \varphi_1 \vee \varphi_2 \longrightarrow \begin{cases} q \Rightarrow \varphi_1 \vee p \\ p \Rightarrow \varphi_2 \end{cases}$$ if φ_2 is not a disjunction of literals.

$$\mathit{Trans}(8) \quad q \Rightarrow D \longrightarrow \mathbf{true} \Rightarrow \neg q \vee D$$

- Temporal operator rules:

$$\mathit{Trans}(9) \quad q \Rightarrow \mathbf{A}\bigcirc\varphi \longrightarrow \begin{cases} q \Rightarrow \mathbf{A}\bigcirc p \\ p \Rightarrow \varphi \end{cases}$$ if φ is not a disjunction of literals.

$$\mathit{Trans}(10) \quad q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi \longrightarrow \begin{cases} q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc p \\ p \Rightarrow \varphi \end{cases}$$ if φ is not a disjunction of literals.

$$\mathit{Trans}(11) \quad q \Rightarrow \mathbf{A}\Diamond\varphi \longrightarrow \begin{cases} q \Rightarrow \mathbf{A}\Diamond p \\ p \Rightarrow \varphi \end{cases}$$ if φ is not a literal.

$$\mathit{Trans}(12) \quad q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\varphi \longrightarrow \begin{cases} q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond p \\ p \Rightarrow \varphi \end{cases}$$ if φ is not a literal.

$$\mathit{Trans}(13) \quad q \Rightarrow \mathbf{A}(\varphi_1 \,\mathcal{U}\, \varphi_2) \longrightarrow \begin{cases} q \Rightarrow \mathbf{A}(\varphi_1 \,\mathcal{U}\, p) \\ p \Rightarrow \varphi_2 \end{cases}$$ if φ_2 is not a literal.

$$\mathit{Trans}(14) \quad q \Rightarrow \mathbf{A}(\varphi_1 \,\mathcal{W}\, \varphi_2) \longrightarrow \begin{cases} q \Rightarrow \mathbf{A}(\varphi_1 \,\mathcal{W}\, p) \\ p \Rightarrow \varphi_2 \end{cases}$$ if φ_2 is not a literal.

$$\mathit{Trans}(15) \quad q \Rightarrow \mathbf{E}_{\langle ind\rangle}(\varphi_1 \,\mathcal{U}\, \varphi_2) \longrightarrow \begin{cases} q \Rightarrow \mathbf{E}_{\langle ind\rangle}(\varphi_1 \,\mathcal{U}\, p) \\ p \Rightarrow \varphi_2 \end{cases}$$ if φ_2 is not a literal.

$$\mathit{Trans}(16) \quad q \Rightarrow \mathbf{E}_{\langle ind\rangle}(\varphi_1 \,\mathcal{W}\, \varphi_2) \longrightarrow \begin{cases} q \Rightarrow \mathbf{E}_{\langle ind\rangle}(\varphi_1 \,\mathcal{W}\, p) \\ p \Rightarrow \varphi_2 \end{cases}$$ if φ_2 is not a literal.

$$\mathit{Trans}(17) \quad q \Rightarrow \mathbf{A}\Box\varphi \longrightarrow \begin{cases} q \Rightarrow p \\ p \Rightarrow \varphi \\ p \Rightarrow \mathbf{A}\bigcirc p \end{cases}$$

$$\mathit{Trans}(18) \qquad q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Box\varphi \longrightarrow \begin{cases} q \Rightarrow p \\ p \Rightarrow \varphi \\ p \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc p \end{cases}$$

$$\mathit{Trans}(19) \qquad q \Rightarrow \mathbf{A}(\varphi\,\mathcal{U}\,l) \longrightarrow \begin{cases} q \Rightarrow l \vee p \\ p \Rightarrow \varphi \\ p \Rightarrow \mathbf{A}\bigcirc(l \vee p) \\ q \Rightarrow \mathbf{A}\Diamond l \end{cases}$$

$$\mathit{Trans}(20) \qquad q \Rightarrow \mathbf{E}_{\langle ind \rangle}(\varphi\,\mathcal{U}\,l) \longrightarrow \begin{cases} q \Rightarrow l \vee p \\ p \Rightarrow \varphi \\ p \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(l \vee p) \\ q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Diamond l \end{cases}$$

$$\mathit{Trans}(21) \qquad q \Rightarrow \mathbf{A}(\varphi\,\mathcal{W}\,l) \longrightarrow \begin{cases} q \Rightarrow l \vee p \\ p \Rightarrow \varphi \\ p \Rightarrow \mathbf{A}\bigcirc(l \vee p) \end{cases}$$

$$\mathit{Trans}(22) \qquad q \Rightarrow \mathbf{E}_{\langle ind \rangle}(\varphi\,\mathcal{W}\,l) \longrightarrow \begin{cases} q \Rightarrow l \vee p \\ p \Rightarrow \varphi \\ p \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(l \vee p) \end{cases}$$

Example 3.1
We demonstrate how we transform the CTL formula $\varphi_1 = \mathbf{A}\Box(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q))$ into an equi-satisfiable set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses using our transformation rules. First, we apply the function *init* to φ_1:

$$\Gamma_1 = \mathit{init}(\varphi_1) = \{\mathbf{A}\Box(\mathbf{start} \Rightarrow p_1), \mathbf{A}\Box(p_1 \Rightarrow \mathbf{A}\Box(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q)))\}.$$

Then we transform the set Γ_1 of clauses into a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses.

1.	$\mathbf{start} \Rightarrow p_1$	
2.	$p_1 \Rightarrow \mathbf{A}\Box(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q))$	
3.	$p_1 \Rightarrow p_2$	$\mathit{Trans}(17) \rightarrow 2$
4.	$p_2 \Rightarrow \mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q)$	$\mathit{Trans}(17) \rightarrow 2$
5.	$p_2 \Rightarrow \mathbf{A}\bigcirc p_2$	$\mathit{Trans}(17) \rightarrow 2$
6.	$\mathbf{true} \Rightarrow \neg p_1 \vee p_2$	$\mathit{Trans}(8) \rightarrow 3$
7.	$p_2 \Rightarrow \mathbf{E}_{\langle 1 \rangle}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q)$	$\mathit{Trans}(4) \rightarrow 4$
8.	$p_2 \Rightarrow q \vee p_3$	$\mathit{Trans}(20) \rightarrow 7$
9.	$p_3 \Rightarrow \mathbf{E}\bigcirc r$	$\mathit{Trans}(20) \rightarrow 7$
10.	$p_3 \Rightarrow \mathbf{E}_{\langle 1 \rangle}\bigcirc(q \vee p_3)$	$\mathit{Trans}(20) \rightarrow 7$
11.	$p_2 \Rightarrow \mathbf{E}_{\langle 1 \rangle}\Diamond q$	$\mathit{Trans}(20) \rightarrow 7$
12.	$\mathbf{true} \Rightarrow \neg p_2 \vee q \vee p_3$	$\mathit{Trans}(8) \rightarrow 8$
13.	$p_3 \Rightarrow \mathbf{E}_{\langle 2 \rangle}\bigcirc r$	$\mathit{Trans}(1) \rightarrow 9$

where the notation $\mathit{Trans}(x) \rightarrow y$ indicates that we apply the transformation rule $\mathit{Trans}(x)$ to the clause y. We obtain the set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses consisting of the following clauses, which is satisfiable iff the CTL formula $\mathbf{A}\Box(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q))$ is satisfiable.

$$\begin{array}{rrcl}
1. & \textbf{start} & \Rightarrow & p_1 \\
5. & p_2 & \Rightarrow & \mathbf{A}\bigcirc p_2 \\
6. & \textbf{true} & \Rightarrow & \neg p_1 \vee p_2 \\
10. & p_3 & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc(q \vee p_3) \\
11. & p_2 & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\Diamond q \\
12. & \textbf{true} & \Rightarrow & \neg p_2 \vee q \vee p_3 \\
13. & p_3 & \Rightarrow & \mathbf{E}_{\langle 2\rangle}\bigcirc r
\end{array}$$

Example 3.2
In the following, we transform the unsatisfiable CTL formula $\varphi_2 = \mathbf{E}\Box\neg l \wedge \mathbf{A}\Diamond l$ into a satisfiability equivalent set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. We apply the function *init* to the CTL formula φ_2:

$$\Gamma_2 = \mathit{init}(\varphi_2) = \{\mathbf{A}\Box(\textbf{start} \Rightarrow p_1), \mathbf{A}\Box(p_1 \Rightarrow \mathbf{E}\Box\neg l \wedge \mathbf{A}\Diamond l)\}.$$

Then we transform the set Γ_2 of clauses into a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses.

$$\begin{array}{rrcll}
1. & \textbf{start} & \Rightarrow & p_1 & \\
2. & p_1 & \Rightarrow & \mathbf{E}\Box\neg l \wedge \mathbf{A}\Diamond l & \\
3. & p_1 & \Rightarrow & \mathbf{E}\Box\neg l & \mathit{Trans}(6) \rightarrow 2 \\
4. & p_1 & \Rightarrow & \mathbf{A}\Diamond l & \mathit{Trans}(6) \rightarrow 2 \\
5. & p_1 & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\Box\neg l & \mathit{Trans}(3) \rightarrow 3 \\
6. & p_1 & \Rightarrow & p_2 & \mathit{Trans}(18) \rightarrow 5 \\
7. & p_2 & \Rightarrow & \neg l & \mathit{Trans}(18) \rightarrow 5 \\
8. & p_2 & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc p_2 & \mathit{Trans}(18) \rightarrow 5 \\
9. & \textbf{true} & \Rightarrow & \neg p_1 \vee p_2 & \mathit{Trans}(8) \rightarrow 6 \\
10. & \textbf{true} & \Rightarrow & \neg p_2 \vee \neg l & \mathit{Trans}(8) \rightarrow 7
\end{array}$$

Then the set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses consisting of the following clauses is satisfiable iff the CTL formula $\mathbf{E}\Box\neg l \wedge \mathbf{A}\Diamond l$ is satisfiable.

$$\begin{array}{rrcl}
1. & \textbf{start} & \Rightarrow & p_1 \\
4. & p_1 & \Rightarrow & \mathbf{A}\Diamond l \\
8. & p_2 & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc p_2 \\
9. & \textbf{true} & \Rightarrow & \neg p_1 \vee p_2 \\
10. & \textbf{true} & \Rightarrow & \neg p_2 \vee \neg l
\end{array}$$

We will return to this set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ later in this chapter.

In [15], Bolotov also provides a set of transformation rules and proves that their application to a CTL formula preserves satisfiability. The transformation rules presented above improve on Bolotov's transformation rules by introducing fewer new propositions and generating fewer clauses during the transformation process. For example, a clause $q \Rightarrow \mathbf{A}\Box r$ is transformed into the set of

$\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses

$$\mathbf{true} \Rightarrow \neg q \vee p \qquad \mathbf{true} \Rightarrow \neg p \vee r \qquad p \Rightarrow \mathbf{A}\bigcirc p$$

by our transformation rules, while Bolotov's transformation rules produce the following set of SNF clauses.

$$\begin{array}{rr} \mathbf{start} \Rightarrow \neg q \vee r & \mathbf{start} \Rightarrow \neg q \vee p \\ \mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg q \vee r) & \mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg q \vee p) \\ p \Rightarrow \mathbf{A}\bigcirc r & p \Rightarrow \mathbf{A}\bigcirc p \end{array}$$

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

Our clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL is based on, but not identical to, the resolution calculus in [15, 18]. The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ consists of

- eight *step* resolution rules SRES1 to SRES8,
- two *eventuality* resolution rules ERES1 and ERES2, and
- two *rewrite* rules RW1 and RW2.

Furthermore, all the rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ operate on $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. The calculus can be used to develop an EXPTIME decision procedure for the satisfiability problem of CTL, as will be shown in Chapter 4.

3.4.1 Step resolution

It is commonly agreed that search space for resolution for classical logic and first-order logic is very large and, consequently, in practice, the refinements for resolution are necessary. In particular, the ordering refinements and selection function refinements are utilised by many efficient theorem provers, for example, SPASS [54], Vampire [72] and Prover9 [55]. This principle that resolution calculi often require refinements in order to be efficient in practice is also true for non-classical logics, for instance CTL. Moreover, $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is designed in such a way that the step resolution can be emulated by first-order resolution (which is discussed in Section 4.2.3). Motivated by refinements of propositional and first-order resolution [11], we restrict the applicability of step resolution rules by means of an atom ordering and a selection function. These refinements have two advantages: (i) we can prove that they do not impair the completeness of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$; and (ii) the efforts of implementing them can be dramatically reduced by reusing some existing high performance first-order resolution prover, which contains these refinements.

To introduce the atom ordering and the selection function we use for step resolution, we give the following definitions first.

Definition 3.3 *Partial ordering*
A *partial ordering* R on a set S is the ordering such that

- for every element $s \in S$, $(s, s) \notin R$;

- for all elements s, t, u of S, if $(s, t), (t, u) \in R$, then $(s, u) \in R$.

Definition 3.4 *Total ordering*
A partial ordering R on a set S is a *total ordering* if for every pair of distinct elements s and t of S, $(s, t) \in R$ or $(t, s) \in R$.

Definition 3.5 *Well-founded ordering*
A partial ordering R on a set S is a *well-founded ordering* if every non-empty subset of S has a minimal element with respect to R.

Definition 3.6 *Multiset*
A *multiset* over a set S is a collection of elements from S in which each elements may occur any number of times. For example $M = \{1, 2, 2, 3, 3, 3\}$ is not a set but a multiset.

An *atom ordering* for $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is a well-founded and total ordering $\succ$ on the set $\mathsf{P_{PL}}$. 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 $q \succ p$.

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 in Ind, P and Q are conjunctions of literals, C and D are disjunctions of literals, neither of which contain duplicate literals, and l is a literal.

SRES1

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

SRES2

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

SRES3

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

SRES4

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

SRES5

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

SRES6

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

SRES7

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

SRES8

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

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

1. l must be strictly maximal with respect to C and no literal is selected in $C \vee l$, and
2. (i) $\neg l$ must be selected in $D \vee \neg l$ or (ii) 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

1. (i) l must be selected in $C \vee l$ or (ii) no literal is selected in $C \vee l$ and l is maximal with respect to C, and
2. $\neg l$ 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, i.e. having or not having the negation $\neg$ in front of l. If l in $C \vee l$ and $\neg l$ in $D \vee \neg l$ satisfy condition (C1) or condition (C2), then we 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:

$$\begin{array}{lrcl}\textbf{RW1} & \bigwedge_{i=1}^{n} m_i \Rightarrow \mathbf{A}\bigcirc\mathbf{false} & \longrightarrow & \mathbf{true} \Rightarrow \bigvee_{i=1}^{n} \neg m_i \\ \textbf{RW2} & \bigwedge_{i=1}^{n} m_i \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc\mathbf{false} & \longrightarrow & \mathbf{true} \Rightarrow \bigvee_{i=1}^{n} \neg m_i\end{array}$$

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

An example of how to apply step resolution and rewrite rules is given below.

Example 3.3
Consider the set T_3 of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses consisting of the clauses 1 to 5 shown below.

$$\begin{array}{lrl}1. & \mathbf{start} & \Rightarrow \underline{r} \\ 2. & \neg q & \Rightarrow \mathbf{E}_{\langle 1 \rangle}\bigcirc\underline{p} \\ 3. & r & \Rightarrow \mathbf{E}_{\langle 2 \rangle}\bigcirc(\neg q \vee \underline{p}) \\ 4. & u & \Rightarrow \mathbf{A}\bigcirc\underline{\neg p} \\ 5. & \mathbf{true} & \Rightarrow \underline{u}\end{array}$$

where the underlined literals indicate that those literals are eligible in the corresponding clauses.

We use the ordering $u \succ p \succ q \succ r$ on atomic propositions and the selection function S that maps every propositional disjunction C to the empty set. Then we are able to derive the contradiction

start $\Rightarrow$ **false** using step resolution and rewrite rules as follows.

$$\begin{array}{rrcll}
6. & \neg q \wedge u & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc\mathbf{false} & [2,4,\mathrm{SRES2}]\\
7. & \mathbf{true} & \Rightarrow & q \vee \underline{\neg u} & [6,\mathrm{RW2}]\\
8. & \mathbf{true} & \Rightarrow & \underline{q} & [5,7,\mathrm{SRES8}]\\
9. & r \wedge u & \Rightarrow & \mathbf{E}_{\langle 2\rangle}\bigcirc\underline{\neg q} & [3,4,\mathrm{SRES2}]\\
10. & r \wedge u & \Rightarrow & \mathbf{E}_{\langle 2\rangle}\bigcirc\mathbf{false} & [8,9,\mathrm{SRES7}]\\
11. & \mathbf{true} & \Rightarrow & \neg r \vee \underline{\neg u} & [10,\mathrm{RW2}]\\
12. & \mathbf{true} & \Rightarrow & \underline{\neg r} & [5,11,\mathrm{SRES8}]\\
13. & \mathbf{start} & \Rightarrow & \mathbf{false} & [1,12,\mathrm{SRES5}]
\end{array}$$

where the notations $[c_1, \ldots, c_n, \mathrm{SRESi}]$ and $[c_1, \ldots, c_n, \mathrm{RWi}]$ indicate that we apply the ith step resolution rule and the ith rewrite rule to clauses $c_1, \ldots, c_n$, respectively.

3.4.2 Eventuality resolution

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 satisfied, imply that $\Box l$ holds on (at least) one path.

ERES1

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

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 $\{\mathbf{A}\bigcirc\} \cup \{\mathbf{E}_{\langle ind\rangle}\bigcirc \mid ind \in \mathsf{Ind}\}$ and for every i, $1 \leq i \leq n$,

$$(\textstyle\bigwedge_{j=1}^{m_i} C^i_j) \Rightarrow l \tag{3.1}$$

and

$$(\textstyle\bigwedge_{j=1}^{m_i} C^i_j) \Rightarrow (\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} P^i_j) \tag{3.2}$$

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

Conditions (3.1) and (3.2) 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:

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

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. It allows us to represent all resolvents by ERES1 using a fixed set of propositions depending only on the initial set of clauses, i.e., n different $\mathbf{A}$-eventualities in the initial set of clauses require at most n new atomic propositions to represent resolvents by ERES1.

In the following we give a concrete example to demonstrate an application of ERES1.

Example 3.4

We resolve the $\mathbf{A}$-sometime clause $u \Rightarrow \mathbf{A}\Diamond\neg l$ with the following set $\Lambda_{\mathbf{E}\Box}$ of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses

$$\begin{array}{ll} p \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc l & q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc l \\ r \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc q & q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc p \\ & q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc r \end{array}$$

From $\Lambda_{\mathbf{E}\Box}$, we obtain that

- the conjunction of right-hand sides of clauses in the first column of $\Lambda_{\mathbf{E}\Box}$ excluding temporal operators implies l, i.e. $(l \wedge q) \Rightarrow l$;
- the conjunction of right-hand sides of clauses in the second column of $\Lambda_{\mathbf{E}\Box}$ excluding temporal operators implies l, i.e. $(l \wedge p \wedge r) \Rightarrow l$;
- the conjunction of right-hand sides of clauses in the first column of $\Lambda_{\mathbf{E}\Box}$ excluding temporal operators implies the disjunction of (i) the conjunction of left-hand sides of clauses in the first column of $\Lambda_{\mathbf{E}\Box}$ and (ii) the conjunction of the left-hand sides of clauses in the second column of $\Lambda_{\mathbf{E}\Box}$, i.e. $(l \wedge q) \Rightarrow ((p \wedge r) \vee q)$; and
- the conjunction of right-hand sides of clauses in the second column of $\Lambda_{\mathbf{E}\Box}$ excluding temporal operators implies the disjunction of (i) the conjunction of left-hand sides of clauses in the first column of $\Lambda_{\mathbf{E}\Box}$ and (ii) the conjunction of the left-hand sides of clauses in the second column of $\Lambda_{\mathbf{E}\Box}$, i.e. $(l \wedge p \wedge r) \Rightarrow ((p \wedge r) \vee q)$.

Therefore, $P^\dagger = ((p \wedge r) \vee q)$ and $P^\dagger \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$. We then can resolve the clause $u \Rightarrow \mathbf{A}\Diamond\neg l$ with $((p \wedge r) \vee q) \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$ as follows.

$$\begin{array}{r} ((p \wedge r) \vee q) \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l \\ u \Rightarrow \mathbf{A}\Diamond\neg l \\ \hline u \Rightarrow \mathbf{A}(\neg((p \wedge r) \vee q)\, \mathcal{W}\, \neg l) \end{array}$$

Then the resolvents of the application of ERES1 above in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ are as follows:

$$\begin{aligned}
w^{\mathbf{A}}_{\neg l} &\Rightarrow \mathbf{A}\bigcirc(\neg l \vee \neg p \vee \neg r)\\
w^{\mathbf{A}}_{\neg l} &\Rightarrow \mathbf{A}\bigcirc(\neg l \vee \neg q)\\
\mathbf{true} &\Rightarrow \neg u \vee \neg l \vee \neg p \vee \neg r\\
\mathbf{true} &\Rightarrow \neg u \vee \neg l \vee \neg q\\
\mathbf{true} &\Rightarrow \neg u \vee \neg l \vee w^{\mathbf{A}}_{\neg l}\\
w^{\mathbf{A}}_{\neg l} &\Rightarrow \mathbf{A}\bigcirc(\neg l \vee w^{\mathbf{A}}_{\neg l}).
\end{aligned}$$

Similar to ERES1, the intuition underlying the ERES2 rule below is to resolve an eventuality $\mathbf{E}_{\langle ind\rangle}\Diamond\neg l$, which states that $\Diamond\neg l$ is true on a path $\chi^{\langle ind\rangle}_{s_i}$, with a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses which together, provided that their combined left-hand sides were true, imply that $\Box l$ also holds on the path $\chi^{\langle ind\rangle}_{s_{i+1}}$.

ERES2

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

where $P^{\dagger} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\mathbf{E}_{\langle ind\rangle}\Box l)$ represents a set, $\Lambda^{ind}_{\mathbf{E}\Box}$, of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses which is analogous to the set $\Lambda_{\mathbf{E}\Box}$ but each $*$ is either empty or an operator in $\{\mathbf{A}\bigcirc, \mathbf{E}_{\langle ind\rangle}\bigcirc\}$ and for every i, $1 \le i \le n$,

$$(\textstyle\bigwedge_{j=1}^{m_i} C^i_j) \Rightarrow l \tag{3.3}$$

and

$$(\textstyle\bigwedge_{j=1}^{m_i} C^i_j) \Rightarrow (\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} P^i_j) \tag{3.4}$$

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

Again, conditions (3.3) and (3.4) ensure that the set $\Lambda^{ind}_{\mathbf{E}\Box}$ of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses implies the formula $P^{\dagger} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\mathbf{E}_{\langle ind\rangle}\Box l)$.

Similarly, we use an atomic proposition $w^{ind}_{\neg l}$ uniquely associated with $\mathbf{E}_{\langle ind\rangle}\Diamond\neg l$ to represent the resolvent of ERES2 as the following set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses:

$$\begin{aligned}
&\{w^{ind}_{\neg l} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\neg l \vee \textstyle\bigvee_{j=1}^{m_i} \neg P^i_j) \mid 1 \le i \le n\}\\
\cup\, &\{\mathbf{true} \Rightarrow \neg Q \vee \neg l \vee \textstyle\bigvee_{j=1}^{m_i} \neg P^i_j \mid 1 \le i \le n\}\\
\cup\, &\{\mathbf{true} \Rightarrow \neg Q \vee \neg l \vee w^{ind}_{\neg l},\ w^{ind}_{\neg l} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\neg l \vee w^{ind}_{\neg l})\}.
\end{aligned}$$

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

We show an example to demonstrate an application of ERES2.

Example 3.5

We resolve the **E**-sometime clause $u \Rightarrow \mathbf{E}_{\langle 1\rangle}\Diamond\neg l$ with the set $\Lambda^{1}_{\mathbf{E}\Box}$ of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses in the following.

$$\begin{aligned} q &\Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc l \\ q &\Rightarrow \mathbf{A}\bigcirc p \\ p &\Rightarrow \mathbf{A}\bigcirc q \end{aligned}$$

From $\Lambda^1_{\mathbf{E}\Box}$, we obtain that

- the conjunction of right-hand sides of clauses excluding temporal operators in $\Lambda^1_{\mathbf{E}\Box}$ implies l, i.e. $(l \wedge p \wedge q) \Rightarrow l$; and
- the conjunction of right-hand sides of clauses excluding temporal operators in $\Lambda^1_{\mathbf{E}\Box}$ implies the conjunction of left-hand sides of clauses in $\Lambda^1_{\mathbf{E}\Box}$, i.e. $(l \wedge p \wedge q) \Rightarrow (q \wedge p)$.

Therefore, $P^\dagger = (p \wedge q)$ and $P^\dagger \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc\mathbf{E}_{\langle 1\rangle}\Box l$. We then can resolve the clause $u \Rightarrow \mathbf{E}_{\langle 1\rangle}\Diamond\neg l$ with $(p \wedge q) \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc\mathbf{E}_{\langle 1\rangle}\Box l$ as follows.

$$\begin{array}{r} (p \wedge q) \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(\mathbf{E}_{\langle 1\rangle}\Box l) \\ u \Rightarrow \mathbf{E}_{\langle 1\rangle}\Diamond\neg l \\ \hline u \Rightarrow \mathbf{E}_{\langle 1\rangle}(\neg(p \wedge q)\,\mathcal{W}\,\neg l) \end{array}$$

Then the resolvents of the application of ERES2 above in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ are as follows:

$$\begin{aligned} w^1_{\neg l} &\Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(\neg l \vee \neg p \vee \neg q) \\ \mathbf{true} &\Rightarrow \neg u \vee \neg l \vee \neg p \vee \neg q \\ \mathbf{true} &\Rightarrow \neg u \vee \neg l \vee w^1_{\neg l} \\ w^1_{\neg l} &\Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(\neg l \vee w^1_{\neg l}). \end{aligned}$$

This completes the presentation of the resolution rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. We now introduce some useful definitions which are needed when we discuss the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ further.

Definition 3.7 *Saturation with respect to step resolution rules*
A set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is *saturated* with respect to step resolution rules, if all clauses that can be derived by an application of one of the step resolution rules SRES1 to SRES8 to premises in T are contained in T.

Definition 3.8 *Saturation with respect to* $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$
A set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is *saturated* with respect to $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ if all clauses that can be derived by an application of a rule of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ to premises in T are contained in T.

Definition 3.9 *Derivation*
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 $T_{t+1} = T_t \cup R_t$ where R_t is a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained as the conclusion of an application of a rule of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ to premises in T_t.

Definition 3.10 *Refutation*
A *refutation* of a set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses (by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$) is a derivation from T such that for some $i \geq 0$, T_i contains a *contradiction*, where a contradiction is either the formula **true** $\Rightarrow$ **false** or **start** $\Rightarrow$ **false**.

Definition 3.11 *Termination*
A derivation *terminates* iff either a contradiction is derived or no new clauses can be derived by any further application of resolution rules.

Next, we present an example of a refutation of a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses which involves both step resolution and eventuality resolution rules.

Example 3.6
In Example 3.2, we have seen that application of our transformation rules to the CTL formula $\varphi = \mathbf{E}\Box\neg l \wedge \mathbf{A}\Diamond l$ results in the following satisfiability equivalent set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses.

$$\begin{array}{rrcl}
1. & \mathbf{start} & \Rightarrow & p_1 \\
2. & p_1 & \Rightarrow & \mathbf{A}\Diamond l \\
3. & p_2 & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc p_2 \\
4. & \mathbf{true} & \Rightarrow & \neg p_1 \vee p_2 \\
5. & \mathbf{true} & \Rightarrow & \neg p_2 \vee \neg l
\end{array}$$

Using step resolution, eventuality resolution and rewrite rules with the ordering $l \succ p_1 \succ p_2 \succ w^A_l$ and the selection function S mapping every propositional disjunction C to an empty set, we are able to generate the following derivation.

$$\begin{array}{rrcll}
6. & w^{\mathbf{A}}_l & \Rightarrow & \mathbf{A}\bigcirc(l \vee \neg p_2) & [2,3,5,\mathrm{ERES1}] \\
7. & \mathbf{true} & \Rightarrow & \neg p_1 \vee l \vee \neg p_2 & [2,3,5,\mathrm{ERES1}] \\
8. & \mathbf{true} & \Rightarrow & \neg p_1 \vee l \vee w^{\mathbf{A}}_l & [2,3,5,\mathrm{ERES1}] \\
9. & w^{\mathbf{A}}_l & \Rightarrow & \mathbf{A}\bigcirc(l \vee w^{\mathbf{A}}_l) & [2,3,5,\mathrm{ERES1}] \\
10. & \mathbf{true} & \Rightarrow & \neg p_1 \vee \neg p_2 & [5,7,\mathrm{SRES8}] \\
11. & \mathbf{start} & \Rightarrow & \neg p_2 & [1,10,\mathrm{SRES5}] \\
12. & \mathbf{start} & \Rightarrow & p_2 & [1,4,\mathrm{SRES5}] \\
13. & \mathbf{start} & \Rightarrow & \mathbf{false} & [11,12,\mathrm{SRES4}]
\end{array}$$

where the notation $[c_1, \ldots, c_n, \mathrm{ERESi}]$ indicates that we apply the eventuality resolution rule ERESi to the clauses $c_1, \ldots, c_n$. Therefore, we have proved that $\mathbf{E}\Box\neg l \wedge \mathbf{A}\Diamond l$ is unsatisfiable.

3.4.3 Loop search

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 (3.1) and (3.2); and $\Lambda^{ind}_{\mathbf{E}\Box}$, satisfying conditions (3.3) and (3.4). Such sets of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses are also called $\mathbf{E}$-*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 [16]. 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, which is slightly different from the **E**-loop search algorithm in [16] due to the presence of global clauses, an ordering and a selection function we introduce into $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$.

In Section 3.7, we will discuss in more detail why an **A**-loop search algorithm is not required in our setting, while in Section 4.2.4 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 $A_1 \Rightarrow B_1$, $A_2 \Rightarrow B_2$, $A_3 \Rightarrow \mathbf{A}\bigcirc B_3$, $A_4 \Rightarrow \mathbf{A}\bigcirc B_4$, $A_5 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc B_5$, and $A_6 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc B_6$ are merged clauses, then so are $(A_1 \wedge A_2) \Rightarrow (B_1 \wedge B_2)$, $(A_1 \wedge A_4) \Rightarrow \mathbf{A}\bigcirc(B_1 \wedge B_4)$, $(A_1 \wedge A_6) \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \wedge B_6)$, $(A_3 \wedge A_4) \Rightarrow \mathbf{A}\bigcirc(B_3 \wedge B_4)$, $(A_3 \wedge A_6) \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_3 \wedge B_6)$, and $(A_5 \wedge A_6) \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_5 \wedge B_6)$.

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 $Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$, 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 with respect to step resolution rules, that is, the rules SRES1 to SRES8.

The algorithm proceeds by constructing a sequence $H_0, H_1, H_2, \ldots$ of formulae which approximate a loop formula. In more detail, the algorithm works as follows:

1. Search in T for merged clauses of the form $X_j \Rightarrow Y_j$, $X_j \Rightarrow \mathbf{A}\bigcirc Y_j$, and $X_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc Y_j$ such that $Y_j \Rightarrow l$ is provable (in propositional logic). Assuming there are n_0 such clauses, we build the first formula as follows:
$$H_0 = \textstyle\bigvee_{j=1}^{n_0} X_j$$
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 $n_0 = 0$), then no loop formula can be found and we return **false**.

2. Given a formula H_i, where $i \geq 0$, build the next formula H_{i+1} by looking in T for merged clauses of the form $A_j \Rightarrow (B_j \wedge l)$, $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_j \wedge l)$ such that $B_j \Rightarrow H_i$ is provable (in propositional logic). Assuming there are n_{i+1} such merged clauses, we build the formula H_{i+1} as follows:
$$H_{i+1} = \textstyle\bigvee_{j=1}^{n_{i+1}} A_j$$
Simplify H_{i+1} using boolean simplification.

3. Repeat the previous step until one of the conditions below is provable (in propositional logic).
   (a) $H_{i+1} \equiv \mathbf{true}$. A loop formula has been found. We return **true** and the algorithm terminates.
   (b) $H_{i+1} \equiv \mathbf{false}$ (that is, $n_{i+1} = 0$). 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 apply an eventuality resolution rule to an **E**-sometime clause $Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Diamond\neg l$, 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 apply an eventuality resolution rule to 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.

If the algorithm returns a formula $H_{i+1} \not\equiv \mathbf{false}$, then

- $H_{i+1} = \bigvee_{j=1}^{n_{i+1}} \bigwedge_{k=1}^{t_j} q_j^k$, for some literals q_j^k, $1 \leq j \leq n_{i+1}$, $1 \leq k \leq t_j$, and
- there exists the following set of clauses in T,

$$\begin{array}{ccc} P_1^1 \Rightarrow * C_1^1 & & P_1^l \Rightarrow * C_1^l \\ \vdots & & \vdots \\ P_{m_1}^1 \Rightarrow * C_{m_1}^1 & \cdots & P_{m_l}^l \Rightarrow * C_{m_l}^l \end{array}$$

  such that these clauses satisfy conditions (3.1) and (3.2) of ERES1 and (3.3) and (3.4) of ERES2 as well as the restrictions imposed on the form of $*$ and, moreover,

$$\bigvee_{r=1}^{l} \bigwedge_{s=1}^{m_r} P_s^r \equiv \bigvee_{j=1}^{n_{i+1}} \bigwedge_{k=1}^{t_j} q_j^k.$$

The proof of the correctness of this algorithm can be found in [16].

An important step in the algorithm is the task of "looking for merged clauses", which is again non-trivial. We will discuss this task in more detail in Section 4.2.4.

We give a number of examples to show how to apply the loop search algorithm and the complexity of those examples gradually increases.

Example 3.7
We apply the **E**-loop search algorithm to the set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses given below.

$$\begin{array}{rl} 1. & p \Rightarrow \mathbf{A}\bigcirc l \\ 2. & \mathbf{true} \Rightarrow p \\ 3. & u \Rightarrow \mathbf{A}\Diamond\neg l \end{array}$$

- Search in T for merged clauses of the form $X_j \Rightarrow Y_j$, $X_j \Rightarrow \mathbf{A}\bigcirc Y_j$, and $X_j \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc Y_j$ such that $Y_j \Rightarrow l$ is provable (in propositional logic). We find $p \Rightarrow \mathbf{A}\bigcirc l$, which results in

$$H_0 = p$$

  It should be noted that there are other ways of merging clauses. For example, by merging clause 1 and 2, we obtain the clause $p \Rightarrow \mathbf{A}\bigcirc(l \wedge p)$, which also results in $H_0 = p$.

- Given the formula H_0, we build the next formula H_1 by looking in T for merged clauses of the form $A_j \Rightarrow (B_j \wedge l)$, $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(B_j \wedge l)$ such that $B_j \Rightarrow H_0$ is provable (in propositional logic). By merging the clauses 1 and 2, we find $p \Rightarrow \mathbf{A}\bigcirc(p \wedge l)$, which results in

$$H_1 = p$$

- As $H_0 \equiv H_1$, the algorithm terminates and the loop formula $P^\dagger$ is p.

Example 3.8
Previously, we have used the set $\Lambda^1_{\mathbf{E}\Box}$ and the **E**-sometime clause $u \Rightarrow \mathbf{E}_{\langle 1\rangle}\Diamond\neg l$ in Example 3.5 to demonstrate an application of ERES2. Here we apply the loop search algorithm to the set $\Lambda^1_{\mathbf{E}\Box}$ to obtain the loop formula.

$$\begin{aligned}&1.\ q \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc l\\&2.\ q \Rightarrow \mathbf{A}\bigcirc p\\&3.\ p \Rightarrow \mathbf{A}\bigcirc q\end{aligned}$$

- Search in $\Lambda^1_{\mathbf{E}\Box}$ for merged clauses of the form $X_j \Rightarrow Y_j$, $X_j \Rightarrow \mathbf{A}\bigcirc Y_j$, and $X_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc Y_j$ such that $Y_j \Rightarrow l$ is provable (in propositional logic). We find $q \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc l$, which results in
$$H_0 = q$$
- Given the formula H_0, we build the next formula H_1 by looking for merged clauses of the form $A_j \Rightarrow (B_j \wedge l)$, $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_j \wedge l)$ such that $B_j \Rightarrow H_0$ is provable (in propositional logic). By merging the clauses 1 and 3, we find $p \wedge q \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(q \wedge l)$, which results in
$$H_1 = p \wedge q$$
It should be noted that there are other ways of merging the available clauses. For example, by merging clauses 1, 2, and 3, we obtain the clause $p \wedge q \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(q \wedge p \wedge l)$, which also results in $H_1 = p \wedge q$.
- Given the formula H_1, we build the next formula H_2 by looking for merged clauses of the form $A_j \Rightarrow (B_j \wedge l)$, $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_j \wedge l)$ such that $B_j \Rightarrow H_1$ is provable (in propositional logic). By merging the clauses 1, 2 and 3, we find $p \wedge q \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(p \wedge q \wedge l)$, which results in
$$H_2 = p \wedge q$$
- As $H_1 \equiv H_2$, the algorithm terminates and the loop formula $P^\dagger$ is $p \wedge q$.

Example 3.9
Previously we use the set $\Lambda_{\mathbf{E}\Box}$ and the **A**-sometime clause $u \Rightarrow \mathbf{A}\Diamond\neg l$ in Example 3.4 to demonstrate an application of ERES1. Here we apply the loop search algorithm to the set $\Lambda_{\mathbf{E}\Box}$ to obtain the loop formula.

$$\begin{aligned}&1.\ p \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc l\\&2.\ r \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc q\\&3.\ q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc l\\&4.\ q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc p\\&5.\ q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc r\end{aligned}$$

- Search in $\Lambda_{\mathbf{E}\Box}$ for merged clauses of the form $X_j \Rightarrow Y_j$, $X_j \Rightarrow \mathbf{A}\bigcirc Y_j$, and $X_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc Y_j$ such that $Y_j \Rightarrow l$ is provable (in propositional logic). We find $p \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc l$ and $q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc l$ and define H_0 as
$$H_0 = p \vee q$$

```
procedure $main$(N)
// N is a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses
begin
    New := $\{$C $|$ C is a determinate clause in N$\}$;
    ST  := $\{$C $|$ C is a sometime clause in N$\}$;
    Old := $\emptyset$;
    do
        Old := $resolution\_sres$(Old, New);
        New := $\emptyset$;
        if ($\perp \notin$ Old) then
           foreach $\mathbf{A}$-sometime clause and $\mathbf{E}$-sometime clause C in ST
               G := $resolution\_eres$(Old, C);
               if (G $\neq$ $\emptyset$) then
                  New := New $\cup$ G;
               end if
           end for
           New := New$\setminus$Old;
        end if
    while ($\perp \notin$ Old and New $\neq \emptyset$)
    $output$();
end
```

Figure 3.3: A decision procedure

- Given the formula H_0, we build the next formula H_1 by looking for merged clauses of the form $A_j \Rightarrow (B_j \wedge l)$, $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_j \wedge l)$ such that $B_j \Rightarrow H_0$ is provable (in propositional logic). By merging the clauses 1 and 2, we find $r \wedge p \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(q \wedge l)$ and by merging the clauses 3 and 4, we find $q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc(p \wedge l)$, which gives us

$$H_1 = q \vee (p \wedge r)$$

  It should be noted that there are other possible ways of merging the available clauses. For example, by merging clauses 3, 4, and 5, we obtain the clause $q \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(p \wedge r \wedge l)$, which also results in $H_1 = q \vee (p \wedge r)$.

- Given the formula H_1, we build the next formula H_2 by looking for merged clauses of the form $A_j \Rightarrow (B_j \wedge l)$, $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_j \wedge l)$ such that $B_j \Rightarrow H_1$ is provable (in propositional logic). We find $r \wedge p \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(q \wedge l)$ (by merging clause 1 and 2) and $q \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc(p \wedge r \wedge l)$ (by merging clause 3, 4 and 5), then

$$H_2 = q \vee (p \wedge r)$$

- As $H_1 \equiv H_2$, the algorithm terminates and the loop formula $P^{\dagger}$ is $q \vee (p \wedge r)$.

3.4.4 A decision procedure

We present a decision procedure based on the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for determining the satisfiability of a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses in Figure 3.3.

The procedure takes a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses `N` as input and then split `N` into the set `New` of determinate clauses and the set `ST` of sometime clauses (lines 4 and 5, respectively). The set `Old` is initially set to be empty (line 6). We then enter the main loop of the procedure which will be repeated until either the contradiction $\bot$ (i.e. **start** $\Rightarrow$ **false** or **true** $\Rightarrow$ **false**) has been derived or we cannot derive any new clauses (line 7 to 19). We saturate the set `New` $\cup$ `Old` using the step resolution rules and the resulting set of clauses becomes the set `Old` (line 8). If we have not derived the contradiction yet, then we try to apply eventuality resolution rules to each of the sometime clauses (lines 11 to 16). The union of all the resolvents generated by applications of the eventuality resolution rules becomes the set of new clauses `New`. Some of these resolvents may be redundant. Therefore, we eliminate clauses from `New` which are already in `Old` (line 17). Finally, after the main loop terminates, we print out the satisfiability of `N` (line 20).

3.5 Correctness of the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$

3.5.1 Correctness of the transformation to $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$

In [15, 38], Bolotov has developed a set of transformation rules to transform an arbitrary CTL formula into a set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses, which preserves satisfiability. However an analysis of the complexity of the transformation and a proof that the transformation is terminating are absent. Our transformation rules have been developed based on those in [15, 38], but are greatly improved for ease of implementation and efficiency.

In the following we show that our transformation

1. preserves satisfiability,
2. is terminating, and
3. allows only a polynomially bounded number of transformation rule application.

Lemma 3.1 *Let T be a set of CTL formulae, and let $M = \langle S, R, L, [\_], s_0\rangle$ be a model structure such that T is satisfiable in M. Let $p \in \mathsf{P}_{\mathsf{PL}}$ be an atomic proposition not occurring in T, and let $M' = \langle S, R, L', [\_], s_0\rangle$ be a model structure identical to M except for the truth value assigned by L' to p 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'. $\Box$

Lemma 3.2 *A CTL formula φ is satisfiable iff the set of formulae $\{\mathbf{A}\Box(\mathbf{start} \Rightarrow p), \mathbf{A}\Box(p \Rightarrow \varphi)\}$, where $p \in \mathsf{P}_{\mathsf{PL}}$ does not occur in φ, is satisfiable.*

Proof. Assume $\{\mathbf{A}\square(\mathbf{start} \Rightarrow p), \mathbf{A}\square(p \Rightarrow \varphi)\}$ is satisfiable in a model $M = \langle S, R, L, [\_], s_0\rangle$, 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 $M = \langle S, R, L, [\_], s_0\rangle$, 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 3.1, $M', s_0 \models \varphi$. 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 $\{\mathbf{A}\square(\mathbf{start} \Rightarrow p), \mathbf{A}\square(p \Rightarrow \varphi)\}$. $\square$

Now we show all of our transformation rules preserve satisfiability.

Lemma 3.3 *Let T be a set of CTL clauses, and let $M = \langle S, R, L, [\_], s_0\rangle$ be a model structure such that T is satisfiable in M. Let $\mathrm{Ind}(T)$ be the set of indices occurring in T and ind be an index, which is not in the set of indices $\mathrm{Ind}(T)$, i.e. ind does not occur in T. Let $M' = \langle S, R, L, [\_]', s_0\rangle$ be a model structure identical to M except that $[ind]'$ is an arbitrary function on S. 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 3.4 *Let $T_t = \Delta \cup \{\psi\}$, where $\psi = \mathbf{A}\square(q \Rightarrow \mathbf{E}\bigcirc\varphi)$, be a set of CTL clauses, and let $\mathrm{Ind}(T_t)$ be the set of indices occurring in T_t. Let T_{t+1} be the set of CTL clauses obtained by an application of Trans(1) to the formula ψ in T_t, that is, $T_{t+1} = \Delta \cup R_t$, where $R_t = \{\mathbf{A}\square(q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi)\}$ and $ind \notin \mathrm{Ind}(T_t)$. Then T_t is satisfiable iff T_{t+1} is satisfiable.*

Proof. Assume a model $M = \langle S, R, L, [\_], s_0\rangle$ satisfies T_{t+1}, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\square(q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi)$. From the semantics of $\wedge$, $M, s_0 \models \Delta$ and $M, s_0 \models \mathbf{A}\square(q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi)$. From the semantics of $\mathbf{A}\square$, for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M, s_j \models (q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi)$. From the semantics of $\Rightarrow$ and $\mathbf{E}_{\langle ind\rangle}\bigcirc$, for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M, s_j \models q$ implies that there exists a state s' such that $(s_j, s') \in [ind]$ and $M, s' \models \varphi$. From the semantics of $\mathbf{E}\bigcirc$, for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M, s_j \models q$ implies that $M, s_j \models \mathbf{E}\bigcirc\varphi$. 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_{t+1} is satisfiable, then so is T_t.

Next we prove the 'only if' part. Assume a model $M = \langle S, R, L, [\_], s_0\rangle$ satisfies T_t, 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 by the semantics of $\Rightarrow, \mathbf{A}\square$ and $\mathbf{E}\bigcirc$, for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M, s_j \models q$ implies that there exists a path χ_{s_j} such that there exists a state $s' \in \chi_{s_j}$, $(s_j, s') \in R$ and $M, s' \models \varphi$. Let the model $M' = \langle S, R, L, [\_]', s_0\rangle$ be identical to M except that $ind \in \mathrm{Ind}(T_{t+1})$ and for every path χ_{s_0} and for every state $s_j \in \chi_{s_0}$

1. if $M', s_j \models q$, then let s' be an arbitrary state with $(s_j, s') \in R$ and $M, s' \models \varphi$ and let $(s_j, s') \in [ind]$; and

2. if $M', s_j \not\models q$, then let s' be an arbitrary state with $(s_j, s') \in R$ and let $(s_j, s') \in [ind]$.

Since we have restricted ourselves to tree models and $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{E}\bigcirc\varphi)$, $[ind]$ is well-defined and a total function.

From the semantics of $\mathbf{E}_{\langle ind\rangle}\bigcirc$, for every path χ_{s_0} and every state $s_j \in \chi_{s_0}, M', s_j \not\models q$ or $M', s_j \models \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi$. From the semantics of $\vee, \Rightarrow, \mathbf{A}\Box$, $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi)$. Moreover, Lemma 3.3 shows that $M', s_0 \models \Delta$. Therefore, from the semantics of $\wedge$, $M', s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi)$. Thus, if T_t is satisfiable, then so is T_{t+1}. □

In the following we show the transformation rule *Trans*(6) preserves satisfiability.

Lemma 3.5 *Let* $T_t = \Delta \cup \{\psi\}$, *where* $\psi = \mathbf{A}\Box(q \Rightarrow \varphi_1 \wedge \varphi_2)$, *and* $T_{t+1} = \Delta \cup R_t$, *where* $R_t = \{\mathbf{A}\Box(q \Rightarrow \varphi_1), \mathbf{A}\Box(q \Rightarrow \varphi_2)\}$, *be two sets of CTL clauses such that* T_{t+1} *is obtained by an application of Trans*(6) *to the formula* ψ *in* T_t. *Then* T_t *is satisfiable iff* T_{t+1} *is satisfiable.*

Proof. Assume $T_t = \Delta \wedge \mathbf{A}\Box(q \Rightarrow \varphi_1 \wedge \varphi_2)$ 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 logical connectives involved, 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 $\langle M, s_0\rangle \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \varphi_1) \wedge \mathbf{A}\Box(q \Rightarrow \varphi_2)$

Therefore, T_t is satisfiable iff T_{t+1} is satisfiable. □

Next, we show the transformation rule *Trans*(8) preserves satisfiability.

Lemma 3.6 *Let* $T_t = \Delta \cup \{\psi\}$, *where* $\psi = \mathbf{A}\Box(q \Rightarrow D)$, *and* $T_{t+1} = \Delta \cup R_t$, *where* $R_t = \{\mathbf{A}\Box(\mathbf{true} \Rightarrow \neg q \vee D)\}$, *be two sets of CTL clauses such that* T_{t+1} *is obtained by an application of Trans*(8) *to the formula* ψ *in* T_t. *Then* T_t *is satisfiable iff* T_{t+1} *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_t is actually equivalent to T_{t+1}. □

The next rule we consider is the rule *Trans*(9).

Lemma 3.7 *Let* $T_t = \Delta \cup \{\psi\}$, *where* $\psi = \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$, *and* $T_{t+1} = \Delta \cup R_t$, *where* $R_t = \{\mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc p), \mathbf{A}\Box(p \Rightarrow \varphi)\}$ *and* $p \in \mathsf{P}_{\mathsf{PL}}$ *does not occur in* T_t, *be two sets of CTL clauses such that* T_{t+1} *is obtained by an application of Trans*(9) *to the formula* ψ *in* T_t. *Then* T_t *is satisfiable iff* T_{t+1} *is satisfiable.*

Proof. We first show the 'if' part. Assume T_{t+1} is satisfiable in a model structure $M = \langle S, R, L, [\_], s_0\rangle$, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc p) \wedge \mathbf{A}\Box(p \Rightarrow \varphi)$. From the semantics of $\wedge$ and $\mathbf{A}\Box$, we obtain that **(1)** $M, s_0 \models \Delta$ and **(2)** for each path χ_{s_0} and for each $s_j \in \chi_{s_0}, M, s_j \models \neg q$ or for each path χ_{s_j}, $M, s_{j+1} \models p$ and **(3)** for each path χ_{s_0} and for each $s_k \in \chi_{s_0}$, $M, s_k \models p \Rightarrow \varphi$.

According to (2), 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 (3) 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 (1), we obtain $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$. Therefore, if T_{t+1} is satisfiable, so is T_t.

Next, we prove the 'only if' part. Assume that T_t is satisfiable in a model structure $M = \langle S, R, L, [\_], s_0\rangle$, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$. Let M' to be a model structure identical to M except that for every state $s_i \in S$, p is true at a state s_i iff φ is true at s_i. By definition of M', we have that $\mathbf{A}\Box(p \Leftrightarrow \varphi)$ holds in M', that is, $M', s_0 \models \mathbf{A}\Box(p \Leftrightarrow \varphi)$. 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, $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$. From $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc\varphi)$ and the semantics of $\Rightarrow$, $\vee$ and $\mathbf{A}\Box$, for each path χ_{s_0} and for each state $s_j \in \chi_{s_0}, M', s_j \not\models q$ or $M', s_j \models \mathbf{A}\bigcirc\varphi$. From the semantics of $\mathbf{A}\bigcirc$, for each path χ_{s_0} and for each state $s_j \in \chi_{s_0}$, $M', s_j \not\models q$ or for each path χ_{s_j}, $M', s_{j+1} \models \varphi$. From $M', s_{j+1} \models \varphi$ and $M', s_0 \models \mathbf{A}\Box(p \Leftrightarrow \varphi)$, we obtain $M', s_{j+1} \models p$. So, for each path χ_{s_0} and for each state $s_j \in \chi_{s_0}, \mathcal{M}', s_j \not\models q$ or $M', s_j \models \mathbf{A}\bigcirc p$. Therefore, from the semantics of $\Rightarrow$ and $\mathbf{A}\Box$, $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\bigcirc p)$. Also, by Lemma 3.1, $M', s_0 \models \Delta$. Thus, if T_t is satisfiable, so is T_{t+1}. □

Lemma 3.8 *The CTL formula* $\mathbf{A}\Box\varphi \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box\varphi$ *is valid.*

Proof. Let $M = \langle S, R, L, [\_], s_0\rangle$ be an arbitrary model structure and s be an arbitrary state in S.

1. If $M, s \not\models \mathbf{A}\Box\varphi$, then $M, s \models \mathbf{A}\Box\varphi \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box\varphi$.
2. If, on the other hand, $M, s \models \mathbf{A}\Box\varphi$, then from the semantics of $\mathbf{A}\Box$ for every path χ_s and every state $s' \in \chi_s, M, s' \models \varphi$. Therefore, for every successor state s'' of s, for every path $\chi_{s''}$, and for every state $s''' \in \chi_{s''}$, we obtain that $M, s''' \models \varphi$. Thus, from the semantics of $\mathbf{A}\Box$, we obtain that for every successor state s'' of s, $M, s'' \models \mathbf{A}\Box\varphi$. From the semantics of $\mathbf{A}\bigcirc$, we obtain that $M, s \models \mathbf{A}\bigcirc\mathbf{A}\Box\varphi$. Thus, $M, s \models \mathbf{A}\Box\varphi \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box\varphi$.

As M is an arbitrary model structure and s is an arbitrary state in S, $\mathbf{A}\Box\varphi \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box\varphi$ is valid. □

Now we prove that the transformation rule *Trans*(17) preserves satisfiability.

Lemma 3.9 *Let* $T_t = \Delta \cup \{\psi\}$*, where* $\psi = \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Box\varphi)$*, and* $T_{t+1} = \Delta \cup R_t$*, where* $R_t = \{\mathbf{A}\Box(q \Rightarrow p), \mathbf{A}\Box(p \Rightarrow \varphi), \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc p)\}$ *and* $p \in \mathsf{P}_{\mathsf{PL}}$ *does not occur in* T_t*, be two sets of CTL clauses such that* T_{t+1} *is obtained by an application of Trans*(17) *to the formula* ψ *in* T_t*. Then* T_t *is satisfiable iff* T_{t+1} *is satisfiable.*

Proof. Assume T_t is satisfiable in $M = \langle S, R, L, [\_], s_0\rangle$, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Box\varphi)$. Let $M' = \langle S, R, L', [\_], s_0\rangle$ be identical to M except that $M', s \models p$ iff $M', s \models \mathbf{A}\Box\varphi$, for every state s in S. Thus, we know that **(1)** $M', s \models p \Leftrightarrow \mathbf{A}\Box\varphi$. By Lemma 3.1, $M', s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Box\varphi)$. From the semantics of $\wedge$, we obtain that **(2)** $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Box\varphi)$. From (2), (1) and the semantics of $\Rightarrow$ and $\mathbf{A}\Box$, as s is an arbitrary state in S, we obtain that $M', s_0 \models \mathbf{A}\Box(q \Rightarrow p)$. Moreover, from (1) and the semantics of $\mathbf{A}\Box$, we obtain that $M', s \models (p \Rightarrow \varphi)$. As s is an arbitrary state, we obtain $M', s_0 \models \mathbf{A}\Box(p \Rightarrow \varphi)$. From (1) and by Lemma 3.8, we obtain that $M', s \models (p \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box\varphi)$. From (1) and the semantics of $\mathbf{A}\bigcirc$, $M', s \models (\mathbf{A}\bigcirc\mathbf{A}\Box\varphi \Rightarrow \mathbf{A}\bigcirc p)$. From the semantics of $\Rightarrow$, we obtain that $M', s \models (p \Rightarrow \mathbf{A}\bigcirc p)$. As s is an arbitrary state, we obtain that $M', s_0 \models \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc p)$.

Therefore, $M', s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow p) \wedge \mathbf{A}\Box(p \Rightarrow \varphi) \wedge \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc p)$. We prove that if T_t is satisfiable, so is T_{t+1}.

Next we prove the 'if' part. Assume T_{t+1} is satisfiable in $M = \langle S, R, L, [\_], s_0\rangle$, i.e. **(3)** $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow p) \wedge \mathbf{A}\Box(p \Rightarrow \varphi) \wedge \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc p)$. Let s be an arbitrary state in S. If $M, s \not\models p$, then $M, s \models (p \Rightarrow \mathbf{A}\Box p)$. If, on the other hand, $M, s \models p$, then by an inductive augment we can conclude from (3) and the semantics of $\mathbf{A}\Box$ and $\mathbf{A}\bigcirc$ that for every path χ_s and for every state $s_i \in \chi_s$, $M, s_i \models p$. Thus, we obtain that **(4)** $M, s \models (p \Rightarrow \mathbf{A}\Box p)$. From (3), we know that in every state $p \Rightarrow \varphi$ holds and, thus, from (4) and the semantics of $\Rightarrow$ and $\mathbf{A}\Box$, we obtain that **(5)** $M, s \models (p \Rightarrow \mathbf{A}\Box\varphi)$. From (3), (5) and the semantics of $\Rightarrow$ and $\mathbf{A}\Box$, we obtain that $M, s \models (q \Rightarrow \mathbf{A}\Box\varphi)$. As s is an arbitrary state, we obtain that $M, s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Box\varphi)$. From (3) and the semantics of $\wedge$, we obtain that $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Box\varphi)$. Thus, if T_{t+1} is satisfiable, so is T_t. □

The next rule we prove to preserve satisfiability is *Trans*(19).

Lemma 3.10 *Let $T_t = \Delta \cup \{\psi\}$, where $\psi = \mathbf{A}\Box(q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l))$, and $T_{t+1} = \Delta \cup R_t$, where $R_t = \{\mathbf{A}\Box(q \Rightarrow l \vee p), \mathbf{A}\Box(p \Rightarrow \varphi), \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(l \vee p)), \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Diamond l)\}$ and $p \in \mathsf{P_{PL}}$ does not occur in T_t, be two sets of CTL clauses such that T_{t+1} is obtained by an application of Trans(19) to the formula ψ in T_t. Then T_t is satisfiable iff T_{t+1} is satisfiable.*

Proof. Assume T_t is satisfiable in the model structure $M = \langle S, R, L, [\_], s_0\rangle$, at the state s_0, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l))$. Let $M' = \langle S, R, L', [\_], s_0\rangle$ be identical to M except that for every state $s_i \in S$, $M', s_i \models p$ iff $M', s_i \models \mathbf{A}(\varphi \mathcal{U} l) \wedge \neg l$. Thus it holds that **(1)** $M', s_i \models p \Leftrightarrow \mathbf{A}(\varphi \mathcal{U} l) \wedge \neg l$.

By Lemma 3.1, we have $M', s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l))$ and consequently $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l))$. From the semantics of $\mathbf{A}\Box$, we have, for every state $s_i \in S$, **(2)** $M', s_i \models q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l)$.

1. From (2) and propositional reasoning, we obtain that **(3)** $M', s_i \models (q \wedge \neg l) \Rightarrow (\mathbf{A}(\varphi \mathcal{U} l)) \wedge \neg l)$. Together with (1), (3) give us $M', s_i \models (q \wedge \neg l \Rightarrow p)$. Thus, $M', s_i \models (q \Rightarrow l \vee p)$. From the semantics of $\mathbf{A}\Box$, $M', s_0 \models \mathbf{A}\Box(q \Rightarrow l \vee p)$.

2. From (1), we have $M', s_i \models (p \Rightarrow \neg l)$ and from the semantics of $\mathbf{A}\mathcal{U}$, (1) also implies that $M', s_i \models (p \Rightarrow \varphi \vee l)$. Thus, we have $M', s_i \models (p \Rightarrow \varphi)$. From the semantics of $\mathbf{A}\Box$, we obtain that $M', s_0 \models \mathbf{A}\Box(p \Rightarrow \varphi)$.

3. From (1) and the semantics of $\mathbf{A}\mathcal{U}$, if $M', s_i \models p$, then for every path χ_{s_i}, there exists a state $s_j \in \chi_{s_i}, j > i$ such that $M', s_j \models l$ and for every state $s_k \in \chi_{s_i}, i \leq k < j, M, s_k \models \varphi$. Thus, we know that for every successor state s_{i+1} of s_i, $M', s_{i+1} \models l \vee (\mathbf{A}(\varphi \mathcal{U} l)) \wedge \neg l)$. From (1), $M', s_{i+1} \models l \vee p$. Thus, from the semantics of $\mathbf{A}\bigcirc$, $M', s_i \models \mathbf{A}\bigcirc(l \vee p)$. Therefore, from the semantics of $\Rightarrow$, we have $M', s_i \models p \Rightarrow \mathbf{A}\bigcirc(l \vee p)$. From the semantics of $\mathbf{A}\Box$, we have $M', s_0 \models \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(l \vee p))$.

4. From (2) and the semantics of $\mathbf{A}\mathcal{U}$ and $\mathbf{A}\Box$, we obtain that if $M', s_i \models q$, then for every path χ_{s_i}, there exists a state $s_j \in \chi_{s_i}$ such that $M', s_j \models l$ and for every state $s_k \in \chi_{s_i}, i \leq k < j, M, s_k \models \varphi$. Thus, $M', s_i \models (q \Rightarrow \mathbf{A}\Diamond l)$. From the semantics of $\mathbf{A}\Box$, we have $M', s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Diamond l)$.

Thus, if T_t is satisfiable, then so is T_{t+1}.

Next, we prove 'if' part. Assume that T_{t+1} is satisfiable in the model structure $M = \langle S, R, L, [\_], s_0\rangle$, i.e. $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow l \vee p) \wedge \mathbf{A}\Box(p \Rightarrow \varphi) \wedge \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(l \vee p)) \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Diamond l)$, and consequently **(4)** $M, s_0 \models \mathbf{A}\Box(q \Rightarrow l \vee p)$, **(5)** $M, s_0 \models \mathbf{A}\Box(p \Rightarrow \varphi)$, **(6)** $M, s_0 \models \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(l \vee p))$, and **(7)** $M, s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Diamond l)$. We need to show that $M, s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l))$.

Let s_i be an arbitrary state in S.

- If $M, s_i \not\models q$, then $M, s_i \models q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l))$;
- if, on the other hand, $M, s_i \models q$, then
  - if $M, s_i \models l$, then, from the semantics of $\mathbf{A}\mathcal{U}$ and $\Rightarrow$, we have $M, s_i \models q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l))$;
  - if, on the other hand, $M, s_i \not\models l$, then from (4) and propositional reasoning, $M, s_i \models p$. By an inductive argument, we can conclude from (6) that for every path χ_{s_i}, either **(8)** for every state $s_j \in \chi_{s_i}, M, s_j \models p \wedge \neg l$ or **(9)** there exists a state s_j such that $M, s_j \models l$ and for every state $s_k, i \leq k < j, M, s_k \models p \wedge \neg l$. From (7) and the semantics of $\mathbf{A}\Box$ and $\mathbf{A}\Diamond$, as $M, s_i \models q$, the possibility of (8) can be eliminated and this leaves (9) as the only possibility. Therefore, we obtain that $M, s_i \models q \Rightarrow \mathbf{A}(p \mathcal{U} l)$. From (5), the semantics of $\mathbf{A}\Box$ and propositional reasoning, we have $M, s_i \models q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l)$.

From the semantics of $\mathbf{A}\Box$, we have $M, s_0 \models \mathbf{A}\Box(q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l)))$. Therefore, $M, s_0 \models \Delta \wedge \mathbf{A}\Box(q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l)))$. Thus, if T_{t+1} is satisfiable, then so is T_t. □

Theorem 3.1 *Let $T_t = \Delta \cup \{\psi\}$ and $T_{t+1} = \Delta \cup R_t$ be two sets of CTL clauses such that T_{t+1} is obtained by an application of a transformation rule of the form $\psi \rightarrow R_t$ in the set Trans to the formula ψ in T_t. Then T_t is satisfiable iff T_{t+1} is satisfiable.*

Proof. To prove this theorem, we need to show that every transformation rule in the set *Trans* preserves satisfiability.

Lemma 3.4, 3.5, 3.6, 3.7, 3.9 and 3.10 prove that the transformation rules *Trans*(1), *Trans*(6), *Trans*(8), *Trans*(9), *Trans*(17) and *Trans*(19) preserve satisfiability, respectively.

The proofs that *Trans*(2) to *Trans*(5) preserve satisfiability are analogous to the proof for *Trans*(1) in Lemma 3.4. The proof that *Trans*(7) preserves satisfiability is analogous to the proof for *Trans*(6) in Lemma 3.5. The proofs that *Trans*(10) to *Trans*(16) preserve satisfiability are analogous to the proof for *Trans*(9) in Lemma 3.7. The proof that *Trans*(18) preserves satisfiability is analogous to the proof for *Trans*(17) in Lemma 3.9. The proofs that *Trans*(20) to *Trans*(22) preserve satisfiability are analogous to the proof for *Trans*(19) in Lemma 3.10. □

Weight functions for CTL formulae

Generally speaking, the purpose of our transformation rules is to keep the operators our resolution rules can operate, eliminate unwanted operators, move the formula to the side we favour, and rewrite a complex formula into a few simpler formulae. To show that the transformation terminates, we assign weights to CTL clauses and sets of CTL clauses. The intuition of how the weights are assigned is that we assign the lower weights to the operators and formulae we favour and higher weights to the operators and formulae we do not want in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$. We give a few example to explain.

1. We prefer all the existential path quantifiers are indexed. Consequently, the weights of $\mathbf{E}\bigcirc, \mathbf{E}\Diamond, \mathbf{E}\Box, \mathbf{E}\,\mathcal{U}, \mathbf{E}\,\mathcal{W}$ are higher than the weights of $\mathbf{E}_{\langle ind\rangle}\bigcirc, \mathbf{E}_{\langle ind\rangle}\Diamond, \mathbf{E}_{\langle ind\rangle}\Box, \mathbf{E}_{\langle ind\rangle}\,\mathcal{U}$, $\mathbf{E}_{\langle ind\rangle}\,\mathcal{W}$, respectively.
2. The unwanted operators are $\Box, \mathcal{U}, \mathcal{W}$. Therefore, their weights are higher than the weights of operators $\bigcirc$ and $\Diamond$.
3. We prefer the propositions to occur at the right-hand side of a clause. Thus, the propositions at the left-hand side of a clause have higher weights compared to the propositions at the right-hand side of a clause.
4. The formula $\mathbf{A}\Diamond\varphi$, where φ is not a literal, is obviously more complex than the formula $\mathbf{A}\Diamond l$. Thus, $\mathbf{A}\Diamond\varphi$ has a higher weight.

Therefore, to show the termination, as any weight of a formula can not be a negative number, we just need to prove that every application of a transformation rule strictly reduces the weight of a set of CTL clauses.

We define the following three weight functions:

1. $w(\Gamma)$, which assigns a weight to a CTL clause Γ;
2. $w(\mathsf{L}, \varphi)$, which assigns a weight to a CTL formula φ occurring on the left-hand side of a CTL clause; and
3. $w(\mathsf{R}, \varphi)$, which assigns a weight to a CTL formula φ occurring on the right-hand side of a CTL clause.

Except for the case for atomic propositions, $w(\mathsf{L}, \varphi)$ and $w(\mathsf{R}, \varphi)$ are defined analogously. Therefore, to ease the following definition, we use $w(x, \varphi)$ where a case of definition applies to both $w(\mathsf{L}, \varphi)$ and $w(\mathsf{R}, \varphi)$. The inductive definition of three weight functions is as follows.

For every CTL clause $\Gamma = \mathbf{A}\Box(\varphi_1 \Rightarrow \varphi_2)$, the weight $w(\Gamma)$ of Γ is defined as follows.

1. $w(\mathbf{A}\Box(\varphi_1 \Rightarrow \varphi_2)) = w(\mathsf{L}, \varphi_1) + w(\mathsf{R}, \varphi_2) + 1$;
2. $w(x, \mathbf{start}) = 1$;
3. $w(x, \mathbf{true}) = w(x, \mathbf{false}) = 1$;
4. $w(\mathsf{L}, p) = 5$;
5. $w(\mathsf{R}, p) = 1$;
6. $w(x, \neg\varphi) = w(x, \varphi)$;
7. $w(x, \varphi_1 \wedge \varphi_2) = w(x, \varphi_1) + w(x, \varphi_2) + 7$;
8. $w(x, \varphi_1 \vee \varphi_2) = w(x, \varphi_1) + w(x, \varphi_2) + 1$, where both φ_1 and φ_2 are disjunctions of literals;

9. $w(x, \varphi_1 \vee \varphi_2) = w(x, \varphi_1) + w(x, \varphi_2) + 9$, where only one of φ_1 and φ_2 is a disjunction of literals;

10. $w(x, \varphi_1 \vee \varphi_2) = w(x, \varphi_1) + w(x, \varphi_2) + 17$, where neither of φ_1 and φ_2 are a disjunctions of literals;

11. $w(x, \mathbf{A}\Box\varphi) = w(x, \mathbf{E}_{\langle ind\rangle}\Box\varphi) = w(x, \varphi) + 16$;

12. $w(x, \mathbf{E}\Box\varphi) = w(x, \varphi) + 17$;

13. $w(x, \mathbf{A}\Diamond\varphi) = w(x, \mathbf{E}_{\langle ind\rangle}\Diamond\varphi) = w(x, \varphi) + 9$, where φ is not a literal;

14. $w(x, \mathbf{A}\Diamond l) = w(x, \mathbf{E}_{\langle ind\rangle}\Diamond l) = w(x, l) + 1$;

15. $w(x, \mathbf{E}\Diamond\varphi) = w(x, \varphi) + 10$;

16. $w(x, \mathbf{A}\bigcirc\varphi) = w(x, \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi) = w(x, \varphi) + 9$, where φ is not a disjunction of literals;

17. $w(x, \mathbf{A}\bigcirc\varphi) = w(x, \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi) = w(x, \varphi) + 1$, where φ is a disjunction of literals;

18. $w(x, \mathbf{E}\bigcirc\varphi) = w(x, \varphi) + 10$;

19. $w(x, \mathbf{A}(\varphi_1 \mathcal{U} \varphi_2)) = w(x, \mathbf{E}_{\langle ind\rangle}(\varphi_1 \mathcal{U} \varphi_2)) = w(x, \varphi_1) + w(x, \varphi_2) + 46$, where φ_2 is not a literal;

20. $w(x, \mathbf{E}(\varphi_1 \mathcal{U} \varphi_2)) = w(x, \varphi_1) + w(x, \varphi_2) + 47$;

21. $w(x, \mathbf{A}(\varphi \mathcal{U} l)) = w(x, \mathbf{E}_{\langle ind\rangle}(\varphi \mathcal{U} l)) = w(x, \varphi) + w(x, l) + 38$;

22. $w(x, \mathbf{A}(\varphi_1 \mathcal{W} \varphi_2)) = w(x, \mathbf{E}_{\langle ind\rangle}(\varphi_1 \mathcal{W} \varphi_2)) = w(x, \varphi_1) + w(x, \varphi_2) + 46$, where φ_2 is not a literal;

23. $w(x, \mathbf{E}(\varphi_1 \mathcal{W} \varphi_2)) = w(x, \varphi_1) + w(x, \varphi_2) + 47$;

24. $w(x, \mathbf{A}(\varphi \mathcal{W} l)) = w(x, \mathbf{E}_{\langle ind\rangle}(\varphi \mathcal{W} l)) = w(x, \varphi) + w(x, l) + 38$;

Note that a disjunction of literals can consist of a single literal. For every set Δ of CTL clauses,

$$w(\Delta) = \sum_{\Gamma \in \Delta} w(\Gamma).$$

In the following, we prove that each application of a transformation rule to a clause Γ in a set T of CTL clauses results in a set T' of CTL clauses that strictly weights less than T. First, we consider the transformation rule *Trans*(1).

Lemma 3.11 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \mathbf{A}\Box(q \Rightarrow \mathbf{E}\bigcirc\varphi)$, be a set of CTL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma'\}$, where $\Gamma' = \mathbf{A}\Box(q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi)$, be a set of CTL clauses such that T_{t+1} is obtained by an application of Trans(1) to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma') > 0$. According to the definition of the weight function for CTL clauses, we have

$$\begin{aligned} w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, \mathbf{E}\bigcirc\varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 10 + 1 \\ &= w(\mathsf{R}, \varphi) + 16; \end{aligned}$$

if φ is not a disjunction of literals, then

$$\begin{aligned} w(\Gamma') &= w(\mathsf{L}, q) + w(\mathsf{R}, \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 9 + 1 \\ &= w(\mathsf{R}, \varphi) + 15; \end{aligned}$$

or if φ is a disjunction of literals, then

$$\begin{aligned} w(\Gamma') &= w(\mathsf{L}, q) + w(\mathsf{R}, \mathbf{E}_{\langle ind\rangle}\bigcirc\varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 1 + 1 \\ &= w(\mathsf{R}, \varphi) + 7. \end{aligned}$$

Therefore, $w(T_t) - w(T_{t+1}) = w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma')$ is 1 or 9, which is greater than 0. □

Lemma 3.12 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \mathbf{A}\square(q \Rightarrow \varphi_1 \wedge \varphi_2)$, be a set of CTL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma_1, \Gamma_2\}$, where $\Gamma_1 = \mathbf{A}\square(q \Rightarrow \varphi)$ and $\Gamma_2 = \mathbf{A}\square(q \Rightarrow \varphi_2)$, be a set of CTL clauses such that T_{t+1} is obtained by an application of Trans(6) to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma_1) - w(\Gamma_2) > 0$. According to the definition of the weight function for CTL clauses, we have

$$\begin{aligned} w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, \varphi_1 \wedge \varphi_2) + 1 \\ &= 5 + w(\mathsf{R}, \varphi_1) + w(\mathsf{R}, \varphi_2) + 7 + 1 \\ &= w(\mathsf{R}, \varphi_1) + w(\mathsf{R}, \varphi_2) + 13 \end{aligned}$$

and

$$\begin{aligned} w(\Gamma_1) &= w(\mathsf{L}, q) + w(\mathsf{R}, \varphi_1) + 1 \\ &= 5 + w(\mathsf{R}, \varphi_1) + 1 \\ &= w(\mathsf{R}, \varphi_1) + 6 \end{aligned}$$

and

$$\begin{aligned} w(\Gamma_2) &= w(\mathsf{L}, q) + w(\mathsf{R}, \varphi_2) + 1 \\ &= 5 + w(\mathsf{R}, \varphi_2) + 1 \\ &= w(\mathsf{R}, \varphi_2) + 6 \end{aligned}$$

Therefore, $w(T_t) - w(T_{t+1}) = (w(\Delta) + w(\Gamma)) - (w(\Delta) + w(\Gamma_1) + w(\Gamma_2)) = (w(\Delta) + w(\mathsf{R}, \varphi_1) + w(\mathsf{R}, \varphi_2) + 13) - (w(\Delta) + w(\mathsf{R}, \varphi_1) + 6 + w(\mathsf{R}, \varphi_2) + 6) = 1 > 0$. □

Lemma 3.13 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \mathbf{A}\Box(q \Rightarrow D)$ and D is a disjunction of literals, be a set of CTL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma'\}$, where $\Gamma' = \mathbf{A}\Box(\mathbf{true} \Rightarrow \neg q \vee D)$, be a set of CTL clauses such that T_{t+1} is obtained by an application of Trans(8) to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma') > 0$. According to the definition of the weight function for CTL clauses, we have

$$\begin{aligned} w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, D) + 1 \\ &= 5 + w(\mathsf{R}, D) + 1 \\ &= w(\mathsf{R}, D) + 6 \end{aligned}$$

and

$$\begin{aligned} w(\Gamma') &= w(\mathsf{L}, \mathbf{true}) + w(\mathsf{R}, \neg q \vee D) + 1 \\ &= 1 + w(\mathsf{R}, \neg q) + w(\mathsf{R}, D) + 1 + 1 \\ &= 1 + w(\mathsf{R}, q) + w(\mathsf{R}, D) + 1 + 1 \\ &= 1 + 1 + w(\mathsf{R}, D) + 1 + 1 \\ &= w(\mathsf{R}, D) + 4 \end{aligned}$$

Therefore, $w(T_t) - w(T_{t+1}) = w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma') = 2 > 0$. □

Lemma 3.14 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \mathbf{A}\Box(q \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc\varphi)$, be a set of CTL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma_1, \Gamma_2\}$, where $\Gamma_1 = \mathbf{A}\Box(q \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc p)$ and $\Gamma_2 = \mathbf{A}\Box(p \Rightarrow \varphi)$, be a set of CTL clauses such that T_{t+1} is obtained by an application of Trans(10) to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma_1) - w(\Gamma_2) > 0$.

According to the definition of the weight function for CTL clauses, we have

$$\begin{aligned} w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, \mathbf{E}_{\langle ind \rangle} \bigcirc \varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 9 + 1 \\ &= w(\mathsf{R}, \varphi) + 15 \end{aligned}$$

and

$$\begin{aligned} w(\Gamma_1) &= w(\mathsf{L}, q) + w(\mathsf{R}, \mathbf{E}_{\langle ind \rangle} \bigcirc p) + 1 \\ &= 5 + w(\mathsf{R}, p) + 1 + 1 \\ &= 5 + 1 + 1 + 1 \\ &= 8 \end{aligned}$$

and

$$\begin{aligned} w(\Gamma_2) &= w(\mathsf{L}, p) + w(\mathsf{R}, \varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 1 \\ &= w(\mathsf{R}, \varphi) + 6 \end{aligned}$$

Therefore, $w(T_t) - w(T_{t+1}) = w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma_1) - w(\Gamma_2) = 1 > 0$. □

Lemma 3.15 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Box\varphi)$, be a set of CTL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma_1, \Gamma_2, \Gamma_3\}$, where $\Gamma_1 = \mathbf{A}\Box(q \Rightarrow p)$, $\Gamma_2 = \mathbf{A}\Box(p \Rightarrow \varphi)$ and $\Gamma_3 = \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc p)$, be a set of CTL clauses such that T_{t+1} is obtained by an application of Trans(17) to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma_1) - w(\Gamma_2) - w(\Gamma_3) > 0$. According to the definition of the weight function for CTL clauses, we have

$$\begin{aligned} w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, \mathbf{A}\Box\varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 16 + 1 \\ &= w(\mathsf{R}, \varphi) + 22 \end{aligned}$$

and

$$\begin{aligned} w(\Gamma_1) &= w(\mathsf{L}, q) + w(\mathsf{R}, p) + 1 \\ &= 5 + 1 + 1 \\ &= 7 \end{aligned}$$

and

$$\begin{aligned}w(\Gamma_2) &= w(\mathsf{L},p) + w(\mathsf{R},\varphi) + 1\\ &= 5 + w(\mathsf{R},\varphi) + 1\\ &= w(\mathsf{R},\varphi) + 6\end{aligned}$$

and

$$\begin{aligned}w(\Gamma_3) &= w(\mathsf{L},p) + w(\mathsf{R},\mathbf{A}\bigcirc p) + 1\\ &= 5 + w(\mathsf{R},p) + 1 + 1\\ &= 5 + 1 + 1 + 1\\ &= 8\end{aligned}$$

Therefore, $w(T_t) - w(T_{t+1}) = w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma_1) - w(\Gamma_2) - w(\Gamma_3) = 1 > 0$. □

Lemma 3.16 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \mathbf{A}\Box(q \Rightarrow \mathbf{A}(\varphi \mathcal{U} l))$, be a set of CTL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4\}$, where $\Gamma_1 = \mathbf{A}\Box(q \Rightarrow l \vee p)$, $\Gamma_2 = \mathbf{A}\Box(p \Rightarrow \varphi)$ $\Gamma_3 = \mathbf{A}\Box(p \Rightarrow \mathbf{A}\bigcirc(l \vee p))$ and $\Gamma_4 = \mathbf{A}\Box(q \Rightarrow \mathbf{A}\Diamond l)$, be a set of CTL clauses such that T_{t+1} is obtained by an application of Trans*(19) *to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma_1) - w(\Gamma_2) - w(\Gamma_3) - w(\Gamma_4) > 0$. According to the definition of the weight function for CTL clauses, we have

$$\begin{aligned}w(\Gamma) &= w(\mathsf{L},q) + w(\mathsf{R},\mathbf{A}(\varphi \mathcal{U} l)) + 1\\ &= 5 + w(\mathsf{R},\varphi) + w(\mathsf{R},l) + 38 + 1\\ &= 5 + w(\mathsf{R},\varphi) + 1 + 38 + 1\\ &= w(\mathsf{R},\varphi) + 45\end{aligned}$$

and

$$\begin{aligned}w(\Gamma_1) &= w(\mathsf{L},q) + w(\mathsf{R},l \vee p) + 1\\ &= 5 + w(\mathsf{R},l) + w(\mathsf{R},p) + 1 + 1\\ &= 5 + 1 + 1 + 1 + 1\\ &= 9\end{aligned}$$

and

$$\begin{aligned}w(\Gamma_2) &= w(\mathsf{L},p) + w(\mathsf{R},\varphi) + 1\\ &= 5 + w(\mathsf{R},\varphi) + 1\\ &= w(\mathsf{R},\varphi) + 6\end{aligned}$$

and

$$\begin{aligned} w(\Gamma_3) &= w(\mathsf{L}, p) + w(\mathsf{R}, \mathbf{A}\bigcirc(l \vee p)) + 1 \\ &= 5 + w(\mathsf{R}, l \vee p) + 1 + 1 \\ &= 5 + w(\mathsf{R}, l) + w(\mathsf{R}, p) + 1 + 1 + 1 \\ &= 5 + 1 + 1 + 1 + 1 + 1 \\ &= 10 \end{aligned}$$

and

$$\begin{aligned} w(\Gamma_4) &= w(\mathsf{L}, q) + w(\mathsf{R}, \mathbf{A}\Diamond l) + 1 \\ &= 5 + w(\mathsf{R}, l) + 1 + 1 \\ &= 5 + 1 + 1 + 1 \\ &= 8 \end{aligned}$$

Therefore, $w(T_t) - w(T_{t+1}) = w(\Delta) + w(\Gamma) - w(\Delta) - w(\Gamma_1) - w(\Gamma_2) - w(\Gamma_3) - w(\Gamma_4) = 12 > 0$. □

Theorem 3.2 *Let T_{t+1} be the set of CTL clauses obtained by an application of a transformation rule to a clause Γ in the set of CTL clauses T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. To show this theorem holds, we only need to prove that $w(T_t) - w(T_{t+1}) > 0$ for each transformation rule. For the transformation rules *Trans*(1), *Trans*(6), *Trans*(8), *Trans*(10), *Trans*(17), and *Trans*(19) we have already done so in Lemma 3.11, 3.12, 3.13, 3.14, 3.15, and 3.16, respectively. For the remaining transformation rules the result can be shown analogously. Below we only list the result of $w(T_t) - w(T_{t+1})$ for each rule.

Rule	$w(T_t) - w(T_{t+1})$	Rule	$w(T_t) - w(T_{t+1})$	Rule	$w(T_t) - w(T_{t+1})$
(1)	1 or 9	(2)	1 or 9	(3)	1
(4)	1 or 9	(5)	1 or 9	(6)	1
(7)	1	(8)	2	(9)	1
(10)	1	(11)	1	(12)	1
(13)	1	(14)	1	(15)	1
(16)	1	(17)	1	(18)	1
(19)	12	(20)	12	(21)	20
(22)	20				

□

Lemma 3.17 *Let T be a set of CTL clauses. If T contains a clause Γ which is not in* $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$*, then there exists a transformation rule, which can be applied to Γ in T.*

Proof. According to the syntax of CTL formulae and $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ formulae, the possible forms of formulae occurring on the right-hand side of a CTL clause are the following: **true**, **false**, p, $\neg\varphi$,

$(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$, $\mathbf{A}\Box\varphi$, $\mathbf{A}\Diamond\varphi$, $\mathbf{A}\bigcirc\varphi$, $\mathbf{A}(\varphi\,\mathcal{U}\,\psi)$, $\mathbf{A}(\varphi\,\mathcal{W}\,\psi)$, $\mathbf{E}\Box\varphi$, $\mathbf{E}\Diamond\varphi$, $\mathbf{E}\bigcirc\varphi$, $\mathbf{E}(\varphi\,\mathcal{U}\,\psi)$, $\mathbf{E}(\varphi\,\mathcal{W}\,\psi)$, $\mathbf{E}_{\langle ind \rangle}\Box\varphi$, $\mathbf{E}_{\langle ind \rangle}\Diamond\varphi$, $\mathbf{E}_{\langle ind \rangle}\bigcirc\varphi$, $\mathbf{E}_{\langle ind \rangle}(\varphi\,\mathcal{U}\,\psi)$, and $\mathbf{E}_{\langle ind \rangle}(\varphi\,\mathcal{W}\,\psi)$, where *ind* is an arbitrary index in Ind, p is a proposition and φ and ψ are CTL formulae. As we apply the functions *simp* and *nnf* at the beginning of the transformation, CTL formulae of the form $\mathbf{true}, \mathbf{false}, \neg\varphi$ (for a formula φ which is not a proposition), and $\varphi \Rightarrow \psi$ can not occur on the right-hand side of a CTL clause in T. For the remaining possible forms that Γ might take, the table below shows that if Γ is not a $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clause, then there exists a transformation rule which can be applied to Γ.

Form	*Trans*	Form	*Trans*	Form	*Trans*
$q \Rightarrow p$	(8)	$q \Rightarrow \mathbf{A}\Box\varphi$	(17)	$q \Rightarrow \mathbf{E}\Box\varphi$	(3)
$q \Rightarrow \neg p$	(8)	$q \Rightarrow \mathbf{A}\Diamond\varphi$	(11)	$q \Rightarrow \mathbf{E}\Diamond\varphi$	(2)
$q \Rightarrow \varphi \wedge \psi$	(6)	$q \Rightarrow \mathbf{A}\bigcirc\varphi$	(9)	$q \Rightarrow \mathbf{E}\bigcirc\varphi$	(1)
$q \Rightarrow \varphi \vee \psi$	(7) or (8)	$q \Rightarrow \mathbf{A}(\varphi\,\mathcal{U}\,\psi)$	(13) or (19)	$q \Rightarrow \mathbf{E}(\varphi\,\mathcal{U}\,\psi)$	(4)
		$q \Rightarrow \mathbf{A}(\varphi\,\mathcal{W}\,\psi)$	(14) or (21)	$q \Rightarrow \mathbf{E}(\varphi\,\mathcal{W}\,\psi)$	(5)
				$q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Box\varphi$	(18)
				$q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Diamond\varphi$	(12)
				$q \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc\varphi$	(10)
				$q \Rightarrow \mathbf{E}_{\langle ind \rangle}(\varphi\,\mathcal{U}\,\psi)$	(15) or (20)
				$q \Rightarrow \mathbf{E}_{\langle ind \rangle}(\varphi\,\mathcal{W}\,\psi)$	(16) or (22)

□

Theorem 3.3 *Let $T_0, T_1, \ldots$ be a sequence of sets of CTL clauses such that $T_0 = init(\varphi)$ for some CTL formula φ and T_{t+1} is obtained from T_t by applying a transformation rule to a clause in T_t. Then the sequence $T_0, T_1, \ldots$ terminates, i.e. there exists an index $n, n \geq 0$, such that no transformation rule can be applied to any clause in T_n. Furthermore, all clauses in T_n are in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$.*

Proof. Follows from Lemma 3.17 and Theorem 3.2. □

Using the following notation of the size of a CTL formula we are able to characterise the computational complexity of the normal form transformation.

Definition 3.12 *Size of a CTL formula*
Let φ and ψ be arbitrary CTL formulae; and p be an arbitrary atomic proposition in $\mathsf{P_{PL}}$. We inductively define the size *sz* of an arbitrary CTL formula as follows:

1. $sz(\mathbf{true}) = sz(\mathbf{false}) = sz(p) = 1$;
2. $sz(\neg\varphi) = sz(\mathbf{A}\Box\varphi) = sz(\mathbf{A}\Diamond\varphi) = sz(\mathbf{A}\bigcirc\varphi) = sz(\mathbf{E}\Box\varphi) = sz(\mathbf{E}\Diamond\varphi) = sz(\mathbf{E}\bigcirc\varphi) = sz(\varphi) + 1$; and
3. $sz(\varphi \wedge \psi) = sz(\varphi \vee \psi) = sz(\varphi \Rightarrow \psi) = sz(\mathbf{A}(\varphi\,\mathcal{U}\,\psi)) = sz(\mathbf{A}(\varphi\,\mathcal{W}\,\psi)) = sz(\mathbf{E}(\varphi\,\mathcal{U}\,\psi)) = sz(\mathbf{E}(\varphi\,\mathcal{W}\,\psi)) = sz(\varphi) + sz(\psi) + 1$.

Theorem 3.4 *Let φ be an arbitrary CTL formula and T_n be a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained from $T_0 = init(\varphi)$ by n applications of our transformation rules. Then n is linearly bounded in the size of φ and the set T_n can be computed in polynomial time in the size of φ.*

Proof. Let φ be of size m and we assume that φ is already in negation normal form. By the definition of the weight function, we know that the weight of $T_0 = init(\varphi)$ is $w(\mathbf{A}\Box(\mathbf{start} \Rightarrow p)) + w(\mathbf{A}\Box(p \Rightarrow \psi))$, where $\psi = simp(nnf(\varphi))$. It is not hard to see that the function *simp* only reduces the size of φ. Thus, the size of ψ is bounded by the size of φ. Furthermore,

$$\begin{aligned} w(\mathbf{A}\Box(\mathbf{start} \Rightarrow p)) &= w(\mathsf{L}, \mathbf{start}) + w(\mathsf{R}, p) + 1 \\ &= 1 + 1 + 1 \\ &= 3 \end{aligned}$$

and

$$\begin{aligned} w(\mathbf{A}\Box(p \Rightarrow \psi)) &= w(\mathsf{L}, p) + w(\mathsf{R}, \psi) + 1 \\ &= 5 + w(\mathsf{R}, \psi) + 1 \\ &= w(\mathsf{R}, \psi) + 6. \end{aligned}$$

Therefore, $w(T_0) = w(\mathsf{R}, \psi) + 9$. As the maximal weight for a constant, proposition, boolean operator or temporal operator is 47, then $w(\mathsf{R}, \psi)$ is bounded by $47m + 9$. Since, by Theorem 3.2, each application of a transformation rule to T_t results a T_{t+1} with $w(T_{t+1}) \leq w(T_t) - 1$, T_n can be computed in less than $47m + 9$ applications of the transformation rules.

Regarding the complexity of each application, we assume CTL clauses are stored in a tree data structure. For example, the tree in Figure 3.4 represents the CTL clause $p \Rightarrow \mathbf{E}\bigcirc(q_1 \vee \mathbf{A}\Box q_2)$. Then according to our transformation rules, by reusing the subtrees representing subformulae as appropriate, generating the results of clauses from the clause which the rule applies to can be accomplished in constant time in the size of φ. The pattern matching procedure determining that for a given CTL clause not in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ which rule to apply, requires linear time in the size of φ in the worst case.

Therefore, the set T_n can be computed in polynomial time in the size of φ. □

Theorem 3.5 *Let φ be an arbitrary CTL formula and T_n be a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained from $T_0 = init(\varphi)$ by a linearly bounded applications of our transformation rules in the size of φ. Then φ is satisfiable iff T_n is satisfiable.*

Proof. Follows from Theorem 3.3, Lemma 3.2, Theorem 3.1, and Theorem 3.4. □

3.5.2 Soundness and completeness

Theorem 3.6 (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.*

$$\begin{tikzcd}
 & \Rightarrow \arrow[dl] \arrow[dr] & & \\
p & & \mathbf{E}\bigcirc \arrow[d] & \\
 & & \vee \arrow[dl] \arrow[dr] & \\
 & q_1 & & \mathbf{A}\Box \arrow[d] \\
 & & & q_2
\end{tikzcd}$$

Figure 3.4: $p \Rightarrow \mathbf{E}\bigcirc(q_1 \vee \mathbf{A}\Box q_2)$ stored in a tree structure

Proof. Let $T_0, T_1, \ldots, T_n$ be a derivation from a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clause $T_0 = T$ 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_{t+1} from T_t for some $t \geq 0$. Assume T_t is satisfiable and $M = \langle S, R, L, [\_], s_0 \rangle$ is a model structure satisfying T_t.

First, we show that SRES1 is sound. Assume $\mathbf{A}\Box(P \Rightarrow \mathbf{A}\bigcirc(C \vee l))$ and $\mathbf{A}\Box(Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l))$ are in T_t. Let T_{t+1} be obtained by an application of SRES1 to $\mathbf{A}\Box(P \Rightarrow \mathbf{A}\bigcirc(C \vee l))$ and $\mathbf{A}\Box(Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l))$, that is, $T_{t+1} = T_t \cup \{\mathbf{A}\Box(P \wedge Q \Rightarrow \mathbf{A}\bigcirc(C \vee D))\}$. We show that M also satisfies T_{t+1}. Consider an arbitrary state $s \in S$. If $M, s \not\models P$ or $M, s \not\models Q$, then obviously $M, s \models P \wedge Q \Rightarrow \mathbf{A}\bigcirc(C \vee D)$. Assume that $M, s \models P$ and $M, s \models Q$. From $\mathbf{A}\Box(P \Rightarrow \mathbf{A}\bigcirc(C \vee l))$, $\mathbf{A}\Box(Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l))$ and the semantics of $\mathbf{A}\Box$, we obtain that $M, s \models P \Rightarrow \mathbf{A}\bigcirc(C \vee l)$ and $M, s \models Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l)$. From the semantics of $\Rightarrow$, we obtain that $M, s \models \mathbf{A}\bigcirc(C \vee l)$ and $M, s \models \mathbf{A}\bigcirc(D \vee \neg l)$. From the semantics of $\mathbf{A}\bigcirc$, we obtain that for all successors s' of state s, $M, s' \models C \vee l$ and $M, s' \models D \vee \neg l$. As l and $\neg l$ cannot both be true at state s', we conclude that $M, s' \models C \vee D$. From the semantics of $\mathbf{A}\bigcirc$, we have $M, s \models \mathbf{A}\bigcirc(C \vee D)$. Therefore, $M, s \models P \wedge Q \Rightarrow \mathbf{A}\bigcirc(C \vee D)$. As s is arbitrary, from the semantics of $\mathbf{A}\Box$, we have $M, s_0 \models \mathbf{A}\Box(P \wedge Q \Rightarrow \mathbf{A}\bigcirc(C \vee D))$.

We show that SRES2 is sound. Assume $\mathbf{A}\Box(P \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(C \vee l))$ and $\mathbf{A}\Box(Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l))$ are in T_t. Let T_{t+1} be obtained by an application of SRES2 to $\mathbf{A}\Box(P \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(C \vee l))$ and $\mathbf{A}\Box(Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l))$, that is, $T_{t+1} = T_t \cup \{\mathbf{A}\Box(P \wedge Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(C \vee D))\}$. We show that M also satisfies T_{t+1}. Consider an arbitrary state $s \in S$. If $M, s \not\models P$ or $M, s \not\models Q$, then obviously $M, s \models P \wedge Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(C \vee D)$. Assume that $M, s \models P$ and $M, s \models Q$. From $\mathbf{A}\Box(P \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(C \vee l))$, $\mathbf{A}\Box(Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l))$ and the semantics of $\mathbf{A}\Box$, we obtain that $M, s \models P \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(C \vee l)$ and $M, s \models Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l)$. From the semantics of $\Rightarrow$, we obtain that $M, s \models \mathbf{E}_{\langle ind \rangle}\bigcirc(C \vee l)$ and $M, s \models \mathbf{A}\bigcirc(D \vee \neg l)$. From the semantics of $\mathbf{A}\bigcirc$, we obtain that for all successors s' of state s, $M, s' \models D \vee \neg l$. From the semantics of $\mathbf{E}_{\langle ind \rangle}\bigcirc$, we have that for the successor s'' of s such that s'' is on the path $\chi_s^{\langle ind \rangle}$, $M, s'' \models C \vee l$. As s'' is a successor of s, $M, s'' \models D \vee \neg l$.

As l and $\neg l$ cannot both be true at state s'', we conclude that $M, s'' \models C \vee D$. From the semantics of $\mathbf{E}_{\langle ind\rangle}\bigcirc$, we have $M, s \models \mathbf{E}_{\langle ind\rangle}\bigcirc(C \vee D)$. Therefore, $M, s \models P \wedge Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(C \vee D)$. As s is arbitrary, from the semantics of $\mathbf{A}\square$, we have $M, s_0 \models \mathbf{A}\square(P \wedge Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(C \vee D))$.

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

Next we show SRES6 is sound. Assume $\mathbf{A}\square(\mathbf{true} \Rightarrow C \vee l)$ and $\mathbf{A}\square(Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l))$ are in T_t. Let T_{t+1} be obtained by an application of SRES6 to them, that is, $T_{t+1} = T_t \cup \{\mathbf{A}\square(Q \Rightarrow \mathbf{A}\bigcirc(C \vee D))\}$. We show that M also satisfies T_{t+1}. Consider an arbitrary state $s \in S$. If $M, s \not\models Q$, then $M, s \models Q \Rightarrow \mathbf{A}\bigcirc(C \vee D)$. If, on the other hand, $M, s \models Q$, then from $M, s_0 \models \mathbf{A}\square(Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l))$ and the semantics of $\mathbf{A}\square$, we obtain $M, s \models \mathbf{A}\bigcirc(D \vee \neg l)$. Consider an arbitrary successor state $s' \in S$ of s, we then have $M, s' \models D \vee \neg l$. By the assumption $M, s_0 \models \mathbf{A}\square(\mathbf{true} \Rightarrow C \vee l)$ and from the semantics of $\mathbf{A}\square, \mathbf{true}, \Rightarrow$, we also have $M, s' \models C \vee l$. As l and $\neg l$ can not both be true at the state s', we conclude $M, s' \models C \vee D$. As state s' is an arbitrary successor state of s, we obtain $M, s \models \mathbf{A}\bigcirc(C \vee D)$. As $M, s \models Q$, we have $M, s \models Q \Rightarrow \mathbf{A}\bigcirc(C \vee D)$. As state s is an arbitrary state, from the semantics of $\mathbf{A}\square$, we obtain $M, s_0 \models \mathbf{A}\square(Q \Rightarrow \mathbf{A}\bigcirc(C \vee D))$.

For the rule SRES3, the proof is analogous to that for SRES2; for the rules SRES4 and SRES8, the proofs are analogous to that for SRES5; and for the rule SRES7, the proof is analogous to that for SRES6.

Regarding RW1, from the semantics of $\mathbf{A}\bigcirc$ and **false** we obtain that the formula $\mathbf{A}\square(Q \Rightarrow \mathbf{A}\bigcirc\mathbf{false})$ is true iff $\mathbf{A}\square(Q \Rightarrow \mathbf{false})$ is true. This formula is propositionally equivalent to $\mathbf{A}\square(\neg Q)$ which in turn, by the semantics of $\Rightarrow$ and **true**, is equivalent to $\mathbf{A}\square(\mathbf{true} \Rightarrow \neg Q)$. The proof for RW2 is analogous.

Next, we show ERES1 is sound. Assume that $\mathbf{A}\square(Q \Rightarrow \mathbf{A}\diamond\neg l)$ is in T_t and there exists a set $\Lambda_{\mathbf{E}\square}$ of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses in T_t together implying $\mathbf{A}\square(P^\dagger \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\square l)$, where $P^\dagger$ is a disjunction of conjunctions of literals (defined in Section 3.4.2). Therefore, M satisfies $\mathbf{A}\square(P^\dagger \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\square l)$. We show that M also satisfies $\mathbf{A}\square(Q \Rightarrow \mathbf{A}(\neg(P^\dagger)\,\mathcal{W}\,\neg l))$. Consider an arbitrary state $s_i \in S$. If $M, s_i \not\models Q$, then obviously $M, s_i \models Q \Rightarrow \mathbf{A}(\neg(P^\dagger)\,\mathcal{W}\,\neg l)$. Assume $M, s_i \models Q$. From $\mathbf{A}\square(Q \Rightarrow \mathbf{A}\diamond\neg l)$ and the semantics of $\mathbf{A}\square$ and $\Rightarrow$, we obtain that $M, s_i \models \mathbf{A}\diamond\neg l$. If $M, s_i \models \neg l$, then by the semantics of $\mathbf{A}\,\mathcal{W}$, $M, s_i \models \mathbf{A}(\neg(P^\dagger)\,\mathcal{W}\,\neg l)$. From the semantics of $\Rightarrow$, $M, s_i \models Q \Rightarrow \mathbf{A}(\neg(P^\dagger)\,\mathcal{W}\,\neg l)$. If, on the other hand, $M, s_i \models l$, then from the semantics of $\mathbf{A}\diamond$, we know that for every path χ_{s_i}, there exists $s_j \in \chi_{s_i}, j > i$ such that $M, s_j \models \neg l$ and for every $k, i \leq k < j, M, s_k \models l$.

For an arbitrary state $s \in S$, if $M, s \models \mathbf{A}\diamond\neg l \wedge l$ then by the semantics of $\mathbf{A}\bigcirc$ and $\mathbf{A}\diamond$, $M, s \models \mathbf{A}\bigcirc\mathbf{A}\diamond\neg l$. Therefore, for all the successors s' of s, $M, s' \models \mathbf{A}\diamond\neg l$.

Due to the property above and $M, s_i \models \mathbf{A}\Diamond\neg l \wedge l$ and $M, s_k \models l$, by an inductive augment we can conclude that for every $k, i \leq k < j, M, s_k \models \mathbf{A}\Diamond\neg l$. As we know, $M, s_k \models l$. Therefore, $M, s_k \models \mathbf{A}\Diamond\neg l \wedge l$. From the semantics of $\mathbf{A}\Diamond$, for all paths χ_{s_k}, there exists $s_n \in \chi_{s_k}, n > k, M, s_n \models \neg l$. Next, we use a proof by contradiction to establish that for all $k, i \leq k < j, M, s_k \models \neg(P^\dagger)$. Assume that $P^\dagger$ holds at s_k. From the semantics of $\mathbf{A}\Box$ and $\Rightarrow$, we have $M, s_k \models \mathbf{E}\bigcirc\mathbf{E}\Box l$. From the semantics of $\mathbf{E}\bigcirc$ and $\mathbf{E}\Box$, we know that there exists a path χ_{s_k} such that for all states $s_m \in \chi_{s_k}, m > k, M, s_m \models l$. This is a contradiction. Therefore, $\neg(P^\dagger)$ must hold at all the states s_k. From the semantics of $\mathbf{A}\mathcal{U}$, we obtain that $M, s_i \models \mathbf{A}(\neg(P^\dagger)\,\mathcal{U}\,\neg l)$. From the semantics of $\mathbf{A}\mathcal{W}$, we obtain that $M, s_i \models \mathbf{A}(\neg(P^\dagger)\,\mathcal{W}\,\neg l)$. Thus, $M, s_i \models Q \Rightarrow \mathbf{A}(\neg(P^\dagger)\,\mathcal{W}\,\neg l)$. As s_i is arbitrary, from the semantics of $\mathbf{A}\Box$, $M, s_0 \models \mathbf{A}\Box(Q \Rightarrow \mathbf{A}(\neg(P^\dagger)\,\mathcal{W}\,\neg l))$. The proof for ERES2 is analogous. $\Box$

Our proof of the completeness of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ makes use of (reduced) labelled behaviour graphs, which will be defined later in this section. These graphs can be seen as finite representations of the set of all models of a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses.

First, we briefly discuss how our proof proceeds. We introduce the idea of augmentation, which was originally developed for a resolution calculus for PLTL [38]. Next, we create a finite labelled directed graph, called a labelled behaviour graph, for an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. To create a CTL model structure for a set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses from a labelled behaviour graph for T, some nodes and some subgraphs of a labelled behaviour 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, if an augmented set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is unsatisfiable, then its reduced labelled behaviour graph is empty. We also prove that each application of a deletion rule corresponds to a derivation from T by $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. Therefore, if T is unsatisfiable, its reduced labelled behaviour graph H_{red} is empty and the sequence of applications 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 $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. In the following, we show the detailed completeness proof.

Let T be a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained by applying the normal form transformation to a given CTL formula. Recall from Section 3.4.2 an application of ERES1 or ERES2 to the set T may introduce new propositions, for example $w^{\mathbf{A}}_{\neg l}$ and $w^{ind}_{\neg l}$, into T. Our completeness proof makes use of the labelled behaviour graph, whose construction is closely related to the set $\mathit{Prop}(T)$ of propositions occurring in T. For our completeness proof, it is very inconvenient if $\mathit{Prop}(T)$ may change during a derivation. Therefore, we introduce augmentation, which adds certain clauses associated with these new propositions into T right from the beginning, i.e. before any resolution rule is applied to T. In this way, we can be sure that no new propositions appear during the application of resolution rules. That is $\mathit{Prop}(T)$ stays the same, if T is augmented. Moreover, we also show that augmentation is correct.

We adapt the *augmentation* procedure used in [38] for PLTL to CTL to establish a relation between the new atomic propositions introduced by applications of ERES1 or ERES2 and eventualities associated with them.

Definition 3.13 *Augmentation*
Given a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clause T, we construct an augmented set $aug(T)$ as follows: the augmented set $aug(T)$ is the smallest set containing T and satisfying the following conditions:

- For every **A**-sometime clause in T, $Q \Rightarrow \mathbf{A}\Diamond\neg l$, $aug(T)$ 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 the proposition uniquely associated with $\mathbf{A}\Diamond\neg l$ (i.e. $w^{\mathbf{A}}_{\neg l}$ is the same proposition we used for ERES1).

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

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

  where $w^{ind}_{\neg l}$ is the proposition uniquely associated with $\mathbf{E}_{\langle ind\rangle}\Diamond l$ (i.e. $w^{ind}_{\neg l}$ is the same proposition we used for ERES2).

It must be noted that there is a minor mistake regarding augmentation in [15]. In [15], the new proposition for a sometime clause is uniquely associated with the tuple of the path quantifier **A** or **E**, the temporal operator $\Diamond$ and the literal, but not including the index if the clause has one. For example, given two **E**-sometime clauses $p \Rightarrow \mathbf{E}_{\langle 1\rangle}\Diamond l$ and $q \Rightarrow \mathbf{E}_{\langle 2\rangle}\Diamond l$, according to [15], the same proposition w_l is associated with both. However, this may lead to the augmentation of a satisfiable set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses being unsatisfiable, as the following example shows.

Example 3.10
Let T be the following set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses (which can be satisfied in the CTL model structure in Figure 3.5).

$$\begin{array}{rrl} 1. & q &\Rightarrow \mathbf{E}_{\langle 1\rangle}\Diamond\neg l \\ 2. & r &\Rightarrow \mathbf{E}_{\langle 2\rangle}\Diamond\neg l \\ 3. & \mathbf{start} &\Rightarrow r \\ 4. & \mathbf{start} &\Rightarrow l \\ 5. & \mathbf{start} &\Rightarrow p \\ 6. & p &\Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc p \\ 7. & p &\Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc l \end{array}$$

Following [15], its augmentation $aug(T)$ consists of T itself and the following clauses.

$$\begin{array}{rrll} 8. & \mathbf{true} &\Rightarrow \neg q \vee \neg l \vee w_{\neg l} & [1, \mathrm{AUG}] \\ 9. & w_{\neg l} &\Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(\neg l \vee w_{\neg l}) & [1, \mathrm{AUG}] \\ 10. & \mathbf{true} &\Rightarrow \neg r \vee \neg l \vee w_{\neg l} & [2, \mathrm{AUG}] \\ 11. & w_{\neg l} &\Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc(\neg l \vee w_{\neg l}) & [2, \mathrm{AUG}] \end{array}$$

where $[c, \mathrm{AUG}]$ indicates the application of augmentation to clause c. Then we are able to derive the following clauses.

$$\begin{array}{lrcll}
12. & w_{\neg l} & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc(\neg l \lor \neg p) & [1, 6, 7, \mathrm{ERES2}]\\
13. & \mathbf{true} & \Rightarrow & \neg q \lor \neg l \lor \neg p & [1, 6, 7, \mathrm{ERES2}]\\
14. & p \land w_{\neg l} & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc\neg l & [6, 12, \mathrm{SRES3}]\\
15. & p \land w_{\neg l} & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc\mathbf{false} & [7, 14, \mathrm{SRES3}]\\
16. & \mathbf{true} & \Rightarrow & \neg p \lor \neg w_{\neg l} & [15, \mathrm{RW2}]\\
17. & \mathbf{start} & \Rightarrow & \neg l \lor w_{\neg l} & [3, 10, \mathrm{SRES5}]\\
18. & \mathbf{start} & \Rightarrow & w_{\neg l} & [4, 17, \mathrm{SRES5}]\\
19. & \mathbf{start} & \Rightarrow & \neg w_{\neg l} & [5, 16, \mathrm{SRES5}]\\
20. & \mathbf{start} & \Rightarrow & \mathbf{false} & [18, 19, \mathrm{SRES4}]
\end{array}$$

Thus, we are able to derive a contradiction from this augmentation of T. As the model structure in Figure 3.5 satisfies the set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses, the set T is satisfiable and $aug(T)$ should also be satisfiable. Therefore, the definition of augmentation in [15] is incorrect.

In contrast, according to our definition of $aug(T)$, the augmentation of T consists of T plus the following clauses.

$$\begin{array}{lrcll}
8. & \mathbf{true} & \Rightarrow & \neg q \lor \neg l \lor w^1_{\neg l} & [1, \mathrm{AUG}]\\
9. & w^1_{\neg l} & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc(\neg l \lor w^1_{\neg l}) & [1, \mathrm{AUG}]\\
10. & \mathbf{true} & \Rightarrow & \neg r \lor \neg l \lor w^2_{\neg l} & [2, \mathrm{AUG}]\\
11. & w^2_{\neg l} & \Rightarrow & \mathbf{E}_{\langle 2\rangle}\bigcirc(\neg l \lor w^2_{\neg l}) & [2, \mathrm{AUG}]
\end{array}$$

With the ordering $r \succ p \succ l \succ w^1_{\neg l} \succ w^2_{\neg l}$ and the selection function S mapping every propositional disjunction C to an empty set, we are then able to derive the clauses below.

$$\begin{array}{lrcll}
12. & w^1_{\neg l} & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc(\neg l \lor \neg p) & [1, 6, 7, \mathrm{ERES2}]\\
13. & \mathbf{true} & \Rightarrow & \neg q \lor \neg l \lor \neg p & [1, 6, 7, \mathrm{ERES2}]\\
14. & p \land w^1_{\neg l} & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc\neg l & [6, 12, \mathrm{SRES3}]\\
15. & p \land w^1_{\neg l} & \Rightarrow & \mathbf{E}_{\langle 1\rangle}\bigcirc\mathbf{false} & [7, 14, \mathrm{SRES3}]\\
16. & \mathbf{true} & \Rightarrow & \neg p \lor \neg w^1_{\neg l} & [15, \mathrm{RW2}]\\
17. & \mathbf{start} & \Rightarrow & \neg l \lor w^2_{\neg l} & [3, 10, \mathrm{SRES5}]\\
18. & \mathbf{start} & \Rightarrow & w^2_{\neg l} & [4, 17, \mathrm{SRES5}]\\
19. & \mathbf{start} & \Rightarrow & \neg w^1_{\neg l} & [5, 16, \mathrm{SRES5}]
\end{array}$$

As we can see, $w^2_{\neg l}$ in clause 18 and $w^1_{\neg l}$ in clause 19 are different atomic propositions, so we are not able to derive a contradiction from clauses 18 and 19.

Next, we formally prove that our augmentation preserves satisfiability.

Lemma 3.18 *Let T be a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and M be a model structure satisfying T. If T' is a subset of T, then M also satisfies T'.*

Proof. Straightforward. □

Lemma 3.19 *Let T be a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. The augmented set $aug(T)$ is satisfiable iff T is satisfiable.*

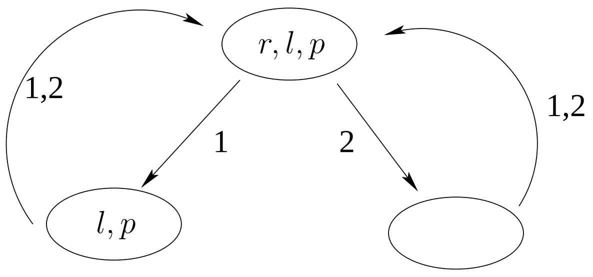


Figure 3.5: A model for T

Proof. As $T \subset aug(T)$, by Lemma 3.18 if $aug(T)$ is satisfiable and a model structure M satisfies $aug(T)$, then M also satisfies T and, thus, T is satisfiable.

Conversely, if T holds in a model structure M_1 at the state s_0, then M_1 can be extended to another model structure M_2 by giving $w^{\mathbf{A}}_{\neg l}$ the same truth value as $l \wedge \mathbf{A}\Diamond\neg l$ and $w^{ind}_{\neg l}$ the same truth value as $l \wedge \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$ in each state in M_2 for each eventuality $\neg l$ in T. We show that M_2 satisfies $aug(T)$ at the state s_0 of M_2.

Assume that $Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$ is in T. We show that M_2 satisfies $\mathbf{A}\Box(\mathbf{true} \Rightarrow \neg Q \vee \neg l \vee w^{ind}_{\neg l})$ and $\mathbf{A}\Box(w^{ind}_{\neg l} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\neg l \vee w^{ind}_{\neg l}))$ i.e. the two clauses added by augmentation for $Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$. Let s be an arbitrary state in M_2.

1. We know that $M_1, s_0 \models \mathbf{A}\Box(Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l)$. By the definition of M_2 and Lemma 3.1, we know that $M_2, s_0 \models \mathbf{A}\Box(Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l)$. From the semantics of $\mathbf{A}\Box$, $M_2, s \models Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$. By propositional reasoning, $M_2, s \models (Q \wedge l) \Rightarrow (l \wedge \mathbf{E}_{\langle ind\rangle}\Diamond\neg l)$. By the definition of M_2, $M_2, s \models (l \wedge \mathbf{E}_{\langle ind\rangle}\Diamond\neg l) \Rightarrow w^{ind}_{\neg l}$. Therefore, by the semantics of $\Rightarrow$, we obtain that $M_2, s \models (Q \wedge l) \Rightarrow w^{ind}_{\neg l}$. Thus, $M_2, s \models \mathbf{true} \Rightarrow \neg Q \vee \neg l \vee w^{ind}_{\neg l}$. Since s is an arbitrary state in M_2, from the semantics of $\mathbf{A}\Box$, we obtain that $M_2, s_0 \models \mathbf{A}\Box(\mathbf{true} \Rightarrow \neg Q \vee \neg l \vee w^{ind}_{\neg l})$.

2. From the definition of M_2, we know $M_2, s \models w^{ind}_{\neg l} \Rightarrow (l \wedge \mathbf{E}_{\langle ind\rangle}\Diamond\neg l)$.

   - If $M_2, s \not\models w^{ind}_{\neg l}$, then $M_2, s \models w^{ind}_{\neg l} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\neg l \vee w^{ind}_{\neg l})$.
   - If, on the other hand, $M_2, s \models w^{ind}_{\neg l}$, then $M_2, s \models l \wedge \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$. Thus, from the semantics of $\mathbf{E}_{\langle ind\rangle}\Diamond$, for the state s' with $(s, s') \in [ind]$, either $M_2, s' \models \neg l$ or $M_2, s' \models l \wedge \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$. By the definition of M_2, we know that $M_2, s' \models (l \wedge \mathbf{E}_{\langle ind\rangle}\Diamond\neg l) \Rightarrow w^{ind}_{\neg l}$. Thus, either $M_2, s' \models \neg l$ or $M_2, s' \models w^{ind}_{\neg l}$. From the semantics of $\mathbf{E}_{\langle ind\rangle}\bigcirc$ and $\vee$, $M_2, s \models \mathbf{E}_{\langle ind\rangle}\bigcirc(\neg l \vee w^{ind}_{\neg l})$. As $M_2, s \models w^{ind}_{\neg l}$, we obtain that $M_2, s \models w^{ind}_{\neg l} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\neg l \vee w^{ind}_{\neg l})$.

   As s is arbitrary, from the semantics of $\mathbf{A}\Box$, $M_2, s_0 \models \mathbf{A}\Box(w^{ind}_{\neg l} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\neg l \vee w^{ind}_{\neg l}))$.

The proof for $Q \Rightarrow \mathbf{A}\Diamond\neg l$ in T is analogous. Therefore, $aug(T)$ is satisfied in M_2 at state s_0. □

We now introduce the notion of a *labelled behaviour graph*. Given a set *Ind* of indices an *Ind-labelled graph* H is an ordered pair $H = (N, E)$, where N is a set of nodes and E is a set of directed edges in H of the form (n, ind, n'), where $n, n' \in N$ and $ind \in Ind$. If there exists an edge $(n, ind, n') \in E$ for some $ind \in Ind$, then n' is a *successor* of n and n is a *predecessor* of n'. If the label *ind* is also important for the relation of n and n' in the context, we also say that n' is an *ind-successor* of n and n is an *ind-predecessor* of n'. When the label on the edge is not important, we use (n, n') to denote an edge, which means the label can be any index in *Ind*.

Definition 3.14 *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 3.15 *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 3.16 *labelled behaviour graph*
Let T be an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and $\mathrm{Ind}(T)$ be the set of indices occurring in T. If $\mathrm{Ind}(T)$ is empty, then let $\mathrm{Ind}(T) = \{ind\}$, where ind is an arbitrary index in Ind. Given T and $\mathrm{Ind}(T)$, we construct a finite directed graph $G = (N, E)$ for T as follows.

The set of nodes N of G consists of all ordered tuples $n = (V, E_A, E_E)$, where

1. V is a valuation of the atomic propositions occurring in T;
2. E_A is a subset of $\{l \mid Q \Rightarrow \mathbf{A}\Diamond l \in T\}$; and
3. E_E is a subset of $\{l_{\langle ind \rangle} \mid Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Diamond l \in T\}$.

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 G 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 successor node of the node n and any valuation satisfies $R_A(n, T)$. Let $R_{ind}(n, T) = \{D \mid Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc D \in T$ 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

$$\mathsf{Ev}_{\mathbf{A}}(V, T) = \{l \mid Q \Rightarrow \mathbf{A}\Diamond l \in T \text{ and } V \models Q\}$$

and

$$\mathsf{Ev}_{\mathbf{E}}(V, T) = \{l_{\langle ind \rangle} \mid Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Diamond l \in T \text{ and } V \models Q\},$$

respectively. Let functions $\mathsf{Unsat}_{\mathbf{A}}(E_A, V)$ and $\mathsf{Unsat}_{ind}(E_E, V)$ be defined as

$$\mathsf{Unsat}_{\mathbf{A}}(E_A, V) = \{l \mid l \in E_A \text{ and } V \not\models l\}$$

and

$$\mathsf{Unsat}_{ind}(E_E, V) = \{l_{\langle ind \rangle} \mid l_{\langle ind \rangle} \in E_E \text{ and } V \not\models l\},$$

respectively. For a node $n = (V, E_A, E_E)$ in G, if $l \in E_A (l_{ind} \in E_E)$ and $V \models l$, then we say that l (l_{ind}) is satisfied in node n.

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 $E'_E = \mathsf{Unsat}_{ind}(E_E, V) \cup \mathsf{Ev}_{\mathbf{E}}(V', T)$. That is, there is an edge labelled with ind from (V, E_A, E_E) to (V', E'_A, E'_E) iff (i) V' satisfies all constraints imposed by **A**-step clauses, **E**-step clauses with the index ind, and global clauses whose left-hand sides are satisfied by V, (ii) E'_A consists of **A**-eventualities not satisfied by V plus additional **A**-eventualities triggered by V', and (iii) E'_E consists of **E**-eventualities with the index ind not satisfied by V plus additional **E**-eventualities with the index ind triggered by V'.

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 G. That is, initial nodes are those nodes such that (i) V satisfies all constraints imposed by initial clauses and global, (ii) E_A consists of **A**-eventualities triggered by V, and (iii) E_E consists of **E**-eventualities with the index ind triggered by V.

The *labelled behaviour graph* $H = (N', E')$ for an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses T is the subgraph of G such that the set $N' \subseteq N$ of nodes and the set $E' \subseteq E$ of edges are reachable from the initial nodes of G.

We now provide an example of a labelled behaviour graph obtained from an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses.

Example 3.11
For the augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses below, the labelled behaviour graph is shown in Figure 3.6.

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

Definition 3.17 *Path from a node n to a node n' through a graph*
A path from a node n_1 to a node n_k in a graph is a sequence of nodes $n_1, n_2, \ldots, n_k$ such that $(n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, n_k)$ are edges of the graph.

Definition 3.18 *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 3.19 *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 3.20 *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 3.20 *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 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 (i) there exists a clause $Q \Rightarrow \mathbf{A}\Diamond l \in T$ such that $V' \models Q$ or (ii) $l \in E_A$ 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 either from $\mathsf{Unsat}_{\mathbf{A}}(E_A, V)$ or from $\mathsf{Ev}_{\mathbf{A}}(V', T)$. 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 3.21 *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. Then, 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. The proof proceeds by induction over the nodes of a path $(n_0, n_1, \ldots)$ from an initial node $n_0 = (V^0, E^0_A, E^0_E)$ to the node $n = (V, E_A, E_E)$.

In the base case, n is an initial node. If $l \in E^0_A$, by the construction of initial nodes, there must be an $\mathbf{A}$-sometime clause $Q \Rightarrow \mathbf{A}\Diamond l \in T$ and $V^0 \models Q$. By augmentation, $\mathbf{true} \Rightarrow \neg Q \vee l \vee w^{\mathbf{A}}_l$. If $V^0 \not\models l$, we obtain that $V^0 \models w^{\mathbf{A}}_l$.

Otherwise we assume that the lemma holds from node n_0 to $n_i = (V^i, E^i_A, E^i_E), i > 0$, and we prove that it holds for node $n_{i+1} = (V^{i+1}, E^{i+1}_A, E^{i+1}_E)$. Based on the assumption of the lemma, $V^{i+1} \not\models l$ and $l \in E^{i+1}_A$. By Lemma 3.20, since $l \in E^{i+1}_A$, either

(1) there exists a clause $Q \Rightarrow \mathbf{A}\Diamond l \in T$ such that $V^{i+1} \models Q$ or

(2) $l \in E^i_A$ and $V^i \not\models l$.

In case (1), by augmentation, $\mathbf{true} \Rightarrow \neg Q \vee l \vee w^{\mathbf{A}}_l \in T$ and since $V^{i+1} \not\models l$, we obtain that $V^{i+1} \models w^{\mathbf{A}}_l$. In case (2) by the induction hypothesis we have $V^i \models w^{\mathbf{A}}_l$. By augmentation we have

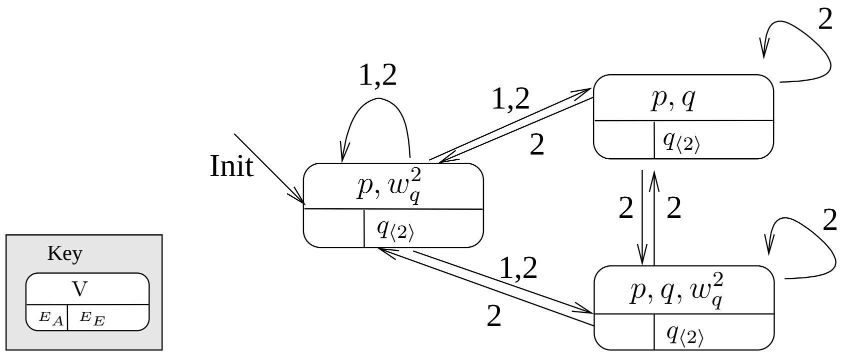


Figure 3.6: A labelled behaviour graph

$w_l^{\mathbf{A}} \Rightarrow \mathbf{A}\bigcirc(w_l^{\mathbf{A}} \vee l) \in T$. Thus by the construction of H, since $V^{i+1} \not\models l$, we have $V^{i+1} \models w_l^{\mathbf{A}}$. Thus, the lemma also holds for node n_{i+1}. □

Lemma 3.22 *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 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 $l_{\langle ind\rangle} \in E'_E$ then either (i) there exists a clause $Q \Rightarrow \mathbf{E}\Diamond l_{\langle ind\rangle} \in T$ such that $V' \models Q$ or (ii) $l_{\langle ind\rangle} \in E_E$ and $V \not\models l$.*

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

Lemma 3.23 *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. Then for every node $n = (V, E_A, E_E)$ in H, if $l_{\langle ind\rangle} \in E_E$ and $V \not\models l$, then $V \models w_l^{ind}$.*

Proof. The proof proceeds analogously to the proof of Lemma 3.21 and uses the fact that T contains the clause $w_l^{ind} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(w_l^{ind} \vee l)$. □

Lemma 3.24 *Let T be an augmented set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses and 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 $H' = (N', E')$ for 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 a 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 $n_0 = (V^0, E^0_A, E^0_E)$ in H'. By Definition 3.16, V^0 satisfies $R_0(T') \cup R_g(T')$. 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 and the sets E^0_A and E^0_E will be the same in both graphs. Therefore, n_0 is also an initial node in H.

Next we assume that every node $n_i = (V^i, E^i_A, E^i_E)$, 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 $n_{i+1} = (V^{i+1}, E^{i+1}_A, E^{i+1}_E)$ in H' with an incoming edge $(n_i, ind, n_{i+1}) \in E', ind \in \mathrm{Ind}(T)$ is also in H.

V^{i+1} satisfies $R_g(T') \cup R_A(n_i, T') \cup R_{ind}(n_i, T')$. Thus V^{i+1} also satisfies $R_g(T) \cup R_A(n_i, T) \cup R_{ind}(n_i, T)$, as $R_g(T) \subseteq R_g(T')$, $R_A(n_i, T) \subseteq R_A(n_i, T')$ and $R_{ind}(n_i, T) \subseteq R_{ind}(n_i, T')$. Furthermore as T and T' contain the same **A**- or **E**-sometime clauses in T, E^{i+1}_A and E^{i+1}_E 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 3.21 *Terminal node*
A node n in a labelled behaviour graph for an augmented set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is a terminal node iff there exists an index $ind \in \mathrm{Ind}(T)$ such that no edges labelled with ind depart from n.

Note that in the labelled behaviour graph shown in Figure 3.6, 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 3.22 *ind-labelled terminal subgraph for* $l_{\langle ind\rangle}$
For a labelled behaviour graph (N, E) for an augmented set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses, a subgraph (N', E') is an *ind*-labelled terminal subgraph for $l_{\langle ind\rangle}$ 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 $n = (V, E_A, E_E) \in N'$, $l_{\langle ind\rangle} \in E_E$ and $V \models \neg l$.

Definition 3.23 *Terminal subgraph for* l
For a labelled behaviour graph (N, E) for an augmented set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses, 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 $ind \in \mathrm{Ind}(T)$ 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 3.7 and Figure 3.8 show examples of an *ind*-labelled terminal subgraph for $q_{\langle 2\rangle}$ 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 3.25 *Given a labelled behaviour graph $H = (N, E)$ and a node $n = (V, E_A, E_E) \in N$, if, for every eventuality $l_{\langle ind\rangle} \in E_E$, $l_{\langle ind\rangle}$ 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 $l_{\langle ind\rangle}$ of H.*

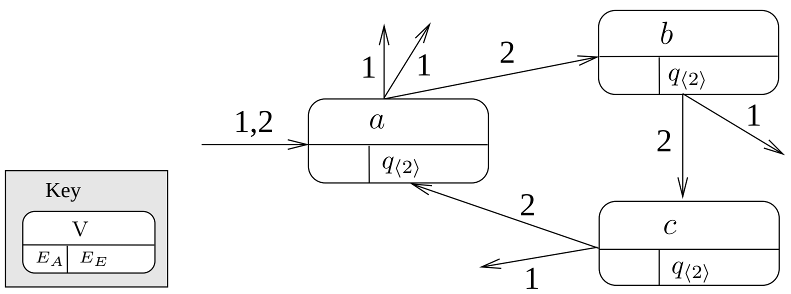


Figure 3.7: A 2-labelled terminal subgraph for $q_{\langle 2\rangle}$

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

Definition 3.24 *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'.
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 3.26 *If an augmented set of* $\text{SNF}^{\text{g}}_{\text{CTL}}$ *clauses T is unsatisfiable, then its reduced labelled behaviour graph H is empty.*

Proof. Proving this lemma is equivalent to proving that, if H is not empty, then T is satisfiable. By the definition of the satisfiability of a CTL formula, it is also equivalent to proving that if H is not empty, then a CTL model structure satisfying T can be constructed from H. Therefore, we assume that the reduced labelled behaviour graph $H = (N, E)$ of T is non-empty and we show how to construct a CTL model structure $M = \langle S, R, L, [\_], s_0 \rangle$ satisfying T from H.

According to the definition of a CTL model structure and the semantics of $\text{SNF}^{\text{g}}_{\text{CTL}}$ clauses, the following properties 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,

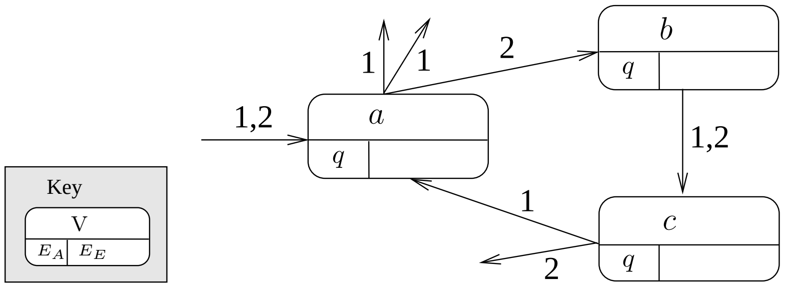


Figure 3.8: A terminal subgraph for q

- $L(s_i)$ 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; and
- 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 $P \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond l \in T$ and every state $s \in S$, if $M, s \models P$, then the path $\chi_s^{\langle ind\rangle}$ 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 χ_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 $H = (N, E)$ and a mapping h from M to H.

Let cs be a function such that $cs(n)$, for every node $n = (V, E_A, E_E)$, is a fresh state s such that $L(s) = V$. In addition, by $RP(s_n)$ we denote a reverse path consisting of a finite sequence $s_n, s_{n-1}, \ldots, s_0$ 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, $0 \leq i \leq n-1$, $(s_i, s_{i+1}) \in R$.

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 $(n_1, n_2, \ldots, n_k)$ ordered in an arbitrary but fixed order ($k > 0$ as otherwise n would be a terminal node in H). Let S^{RP} be the 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 $s_{i+1} = cs(n_1)$ and $h(s_{i+1}) = n_1$;
- 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 assume $h(s) = n_m \in \{n_1, n_2, \ldots, n_k\}, 1 \leq m \leq k$, then
  - $s_{i+1} = cs(n_{m+1})$ and $h(s_{i+1}) = n_{m+1}$, if $m \neq k$;
  - $s_{i+1} = cs(n_1)$ and $h(s_{i+1}) = n_1$, if $m = k$.

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

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 $P \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond l$ is in T and let s be an arbitrary state in S such that $M, s \models P$. We need to show that the path $\chi_s^{\langle ind\rangle}$ contains a state s' such that $l \in L(s')$. We give a proof by contradiction.

Assume l does not hold on $\chi_s^{\langle ind \rangle}$. We know the path $\chi_s^{\langle ind \rangle}$ is an infinite sequence, whereas the set of nodes in H is finite, which implies that there are nodes $\{n_1^t, n_2^t, \ldots, n_k^t\} \in H, k \geq 1$, that are used infinitely often to construct the path $\chi_s^{\langle ind \rangle}$. As we assume that l does not hold on $\chi_s^{\langle ind \rangle}$, we obtain that, for every state $s'' \in \chi_s^{\langle ind \rangle}$, $M, s'' \not\models l$. Therefore, for every node $h(s'') = (V'', E''_A, E''_E)$, $V'' \not\models l$. Moreover, by the construction of H and M, $P \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc l \in T$ and $M, s \models P$, we obtain that $l_{\langle ind \rangle} \in E''_E$. Therefore, for $1 \leq i \leq k$, $n_i^t = (V_i^t, {E_A}_i^t, {E_E}_i^t)$, $V_i^t \not\models l$ and $l_{\langle ind \rangle} \in {E_E}_i^t$. By the way we construct M, all the ind-successors of each node in the set $\Delta = \{n_1^t, n_2^t, \ldots, n_k^t\}$ are also in Δ. 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 $l_{\langle ind \rangle}$ 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 the path $\chi_s^{\langle ind \rangle}$ 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. $\square$

Lemma 3.27 *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 $\mathbf{A}\square$ 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 $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$. Then, we can use step resolution rules SRES4, SRES5, and SRES8 on this set T to derive a contradiction, namely either $\mathbf{start} \Rightarrow \mathbf{false}$ or $\mathbf{true} \Rightarrow \mathbf{false}$. $\square$

Lemma 3.28 *If the unreduced labelled behaviour graph for an augmented set of* $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ *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 by 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 3.27 there exists a refutation of T using step resolution rules SRES4, SRES5, and SRES8. $\square$

Theorem 3.7 (Completeness of $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$) *If a finite augmented set T of* $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ *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 arbitrary augmented unsatisfiable set of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ 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 3.28 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 3.26, 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 3.24. So, there is an index $ind \in \mathrm{Ind}(T)$ such that n has no any ind-successors. Then n is a terminal node $n = (V, E_A, E_E)$. Consider $W = \{D \mid P \Rightarrow \mathbf{A}\bigcirc D \in T$ and $V \models P\} \cup \{D' \mid P' \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc D' \in T$, 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 3.16, W must be unsatisfiable, for otherwise there would exist a node n' that could serve as an ind-successor of n.

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 $Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\mathbf{false}$ with $ind \in \mathrm{Ind}(T)$, 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 $i, 1 \le i \le n, N_i = N_{i-1} \cup \{C_i\}$, 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 $N'_0 = \{P \Rightarrow \mathbf{A}\bigcirc D \in T$ and $V \models P\} \cup \{P' \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc D' \in T, ind \in \mathrm{Ind}(T)$, and $V \models P'\} \cup \{\mathbf{true} \Rightarrow D'' \in T\}$ and for every $i, 1 \le i \le n, N'_i = N'_{i-1} \cup \{C'_i\}$, where C'_i is either $\mathbf{true} \Rightarrow C_i$, $P_i \Rightarrow \mathbf{A}\bigcirc C_i$ with $V \models P_i$ or $P_i \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C_i, ind \in \mathrm{Ind}(T)$ with $V \models P_i$. C'_n is either $\mathbf{true} \Rightarrow \mathbf{false}$, $P_n \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$ or $P_n \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\mathbf{false}, ind \in \mathrm{Ind}(T)$ with $V \models P_n$.

We prove the base case first. Let $N_1 = W \cup \{C_1\}$. 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 $P_1 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C_1, ind \in \mathrm{Ind}(T)$ 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 $C_1 = B_1 \vee B_2$ is derived from two clauses $B_1 \vee l$ and $B_2 \vee \neg l$, then by the construction of W we are able to find a clause $G = P_0 \Rightarrow \mathbf{A}\bigcirc(B_1 \vee l)$, $P_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee l)$ or $\mathbf{true} \Rightarrow B_1 \vee l$ and $G' = P_0 \Rightarrow \mathbf{A}\bigcirc(B_2 \vee \neg l)$, $P_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_2 \vee \neg l)$ or $\mathbf{true} \Rightarrow B_2 \vee \neg l$ 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.

$$\begin{array}{rl}
\text{From } G = & P_0 \Rightarrow \mathbf{A}\bigcirc(B_1 \vee l) \\
\text{and } G' = & P'_0 \Rightarrow \mathbf{A}\bigcirc(B_2 \vee \neg l) \\
\hline
\multicolumn{2}{c}{\text{we can derive } C'_1 = P_0 \wedge P'_0 \Rightarrow \mathbf{A}\bigcirc(B_1 \vee B_2) \text{ by SRES1}}
\end{array}$$

$$\begin{array}{rl}
\text{From } G = & P_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee l) \\
\text{and } G' = & P'_0 \Rightarrow \mathbf{A}\bigcirc(B_2 \vee \neg l) \\
\hline
\multicolumn{2}{c}{\text{we can derive } C'_1 = P_0 \wedge P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee B_2) \text{ by SRES2}}
\end{array}$$

$$\begin{array}{rl}
\text{From } G = & P_0 \Rightarrow \mathbf{A}\bigcirc(B_1 \vee l) \\
\text{and } G' = & P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_2 \vee \neg l) \\
\hline
\multicolumn{2}{c}{\text{we can derive } C'_1 = P_0 \wedge P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee B_2) \text{ by SRES2}}
\end{array}$$

$$\begin{array}{rl}
\text{From } G = & P_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee l) \\
\text{and } G' = & P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_2 \vee \neg l) \\
\hline
\multicolumn{2}{l}{\text{we can derive } C'_1 = P_0 \wedge P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee B_2) \text{ by SRES3}}
\end{array}$$

$$\begin{array}{rl}
\text{From } G = & \mathbf{true} \Rightarrow B_1 \vee l \\
\text{and } G' = & P'_0 \Rightarrow \mathbf{A}\bigcirc(B_2 \vee \neg l) \\
\hline
\text{we can derive } C'_1 = & P'_0 \Rightarrow \mathbf{A}\bigcirc(B_1 \vee B_2) \text{ by SRES6}
\end{array}$$

$$\begin{array}{rl}
\text{From } G = & P_0 \Rightarrow \mathbf{A}\bigcirc(B_1 \vee l) \\
\text{and } G' = & \mathbf{true} \Rightarrow B_2 \vee \neg l \\
\hline
\text{we can derive } C'_1 = & P_0 \Rightarrow \mathbf{A}\bigcirc(B_1 \vee B_2) \text{ by SRES6}
\end{array}$$

$$\begin{array}{rl}
\text{From } G = & \mathbf{true} \Rightarrow B_1 \vee l \\
\text{and } G' = & P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_2 \vee \neg l) \\
\hline
\text{we can derive } C'_1 = & P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee B_2) \text{ by SRES7}
\end{array}$$

$$\begin{array}{rl}
\text{From } G = & P_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee l) \\
\text{and } G' = & \mathbf{true} \Rightarrow B_2 \vee \neg l \\
\hline
\text{we can derive } C'_1 = & P_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_1 \vee B_2) \text{ by SRES7}
\end{array}$$

$$\begin{array}{rl}
\text{From } G = & \mathbf{true} \Rightarrow B_1 \vee l \\
\text{and } G' = & \mathbf{true} \Rightarrow B_2 \vee \neg l \\
\hline
\text{we can derive } C'_1 = & \mathbf{true} \Rightarrow B_1 \vee B_2 \text{ by SRES8}
\end{array}$$

where l is eligible in $B_1 \vee l$ and $\neg l$ is eligible in $B_2 \vee \neg l$ for a given atom ordering $\succ$ and a given selection function S of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ as otherwise we would not have been able to derive $C_1 = B_1 \vee B_2$ on the propositional level using ordered resolution with selection given the ordering $\succ$ and the selection function S.

Because $C_1 = B_1 \vee B_2$, C'_1 is one of the clauses $P_0 \wedge P'_0 \Rightarrow \mathbf{A}\bigcirc C_1$, $P_0 \wedge P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C_1$, $P_0 \Rightarrow \mathbf{A}\bigcirc C_1$, $P'_0 \Rightarrow \mathbf{A}\bigcirc C_1$, $P_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C_1$, $P'_0 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C_1$, or $\mathbf{true} \Rightarrow B_1 \vee B_2$. It is easy to see that since $V \models P_0$ and $V \models P'_0$, we have $V \models P_1$. Furthermore, the 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, let $N_{i+1} = N_i \cup \{C_i\}$, 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 $P_i \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C_i, ind \in \mathrm{Ind}(T)$ 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 $C'_n = P_n \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$, $P_n \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\mathbf{false}$ or $\mathbf{true} \Rightarrow \mathbf{false}$ from T. From $P_n \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$ or $P_n \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\mathbf{false}$ we can obtain the clause $\mathbf{true} \Rightarrow \neg P_n$ in normal form using RW1 or RW2.

By Lemma 3.24, the labelled behaviour graph H' for N'_n is a subgraph of H. In particular, every node in H' has to satisfy $\neg P_n$. 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 3.24 is applicable to H. Then there must exist an eventuality $l_{\langle ind \rangle}$ (or l), where $l_{\langle ind \rangle}$ (or l) is not satisfied in an *ind*-labelled terminal subgraph for $l_{\langle ind \rangle}$ (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 $l_{\langle ind \rangle}$.** Let $Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Diamond l$ be a clause in T and $H' = (N', E')$ be an *ind*-labelled terminal subgraph for $l_{\langle ind \rangle}$ of the behaviour graph H. For each $n = (V, E_A, E_E) \in N'$, let *loop*(n) be the set consisting of all global, **A**-step, and **E**-step clauses labelled with *ind* in T whose left-hand sides are satisfied by V, and let $F_n \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc G_n$ be the clause, which is the result of merging all clauses in *loop*$(n) \cup \{\mathbf{true} \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc\mathbf{true}\}$. To show the set $\bigcup_{n \in N'}$ *loop*(n) is an **E**-loop in $\neg l$, we must check the following two conditions.

  - For each $n \in N'$, we must have $\models G_n \Rightarrow \neg l$. In the following, we use a proof by contradiction to establish it. By the construction of H, G_n is the only constraint on the valuations of *ind*-successors of n. Therefore, if the implication $G_n \Rightarrow \neg l$ is not valid, then, by the construction of H, there must be an *ind*-successor $n' = (V', E'_A, E'_E)$ of n, such that $V' \models l$. By property (ITS2), every *ind*-successor of n is in H'. By property (ITS3), for every *ind*-successor $n_i = (V^i, E^i_A, E^i_E)$ of the node n, $V^i \models \neg l$. Therefore, we obtain a contradiction. So, $G_n \Rightarrow \neg l$ is valid.
  - For each $n \in N'$ we must have $\models G_n \Rightarrow \bigvee_{n' \in N'} F_{n'}$. Let $\{n_1, n_2, \ldots, n_k\}, k \geq 1$ be the set of *ind*-successors of the node n. We show that the assumption that $G_n \Rightarrow \bigvee_{n' \in N'} F_{n'}$ is not valid leads to a contradiction. By property (ITS2), every *ind*-successor of n is also in N'. Thus, to prove $\models G_n \Rightarrow \bigvee_{n' \in N'} F_{n'}$, it is enough to prove that $\models G_n \Rightarrow \bigvee_{i=1}^{k} F_{n_i}$. By the construction of H, G_n is the only constraint on the valuations of *ind*-successors of n. Therefore, if $G_n \Rightarrow \bigvee_{i=1}^{k} F_{n_i}$ is not valid, then, by the construction of H, there must be an *ind*-successor $n'' = (V'', E''_A, E''_E)$ of n, such that $V'' \models \neg(\bigvee_{i=1}^{k} F_{n_i})$, namely $V'' \models \neg F_{n_1} \wedge \ldots \wedge \neg F_{n_k}$. As we know, for every *ind*-successor $n_i = (V^i, E^i_A, E^i_E), 1 \leq i \leq k$, of the node n, $V^{n_i} \models F_{n_i}$. Therefore, n'' can not be in the set $\{n_1, \ldots, n_k\}$. This is a contradiction. Thus, $G_n \Rightarrow \bigvee_{i=1}^{k} F_{n_i}$ is valid.

  Since we show that the set $\bigcup_{n \in N'}$ *loop*(n) is an **E**-loop in $\neg l$, we are able to use it in an application of ERES2 with the eventuality $l_{\langle ind \rangle}$ occurring in $Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Diamond l \in T$. Let L be defined as

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

  Then $T' = T \cup \{w_l^{ind} \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(\neg L \vee l),\ \mathbf{true} \Rightarrow \neg Q \vee \neg L \vee l\}$ 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', (i) $V \models L$; and (ii) by property (ITS3), $V \models \neg l$ and $l_{ind} \in E_E$. Therefore, $V \models \neg(\neg L \vee l)$. Moreover, by Lemma 3.23, $V \models w_l^{ind}$. Recall that through augmentation the set T contains clauses

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

By Lemma 3.22 either **(1)** there is an edge $(n', ind, n) \in E$, where $n' = (V', E'_A, E'_E)$, $l_{\langle ind \rangle} \in E'_E$, and $V' \models \neg l$; or **(2)** $V \models Q, V \models \neg l$.

1. In the case of (1), we have $V' \models w_l^{ind}$ by Lemma 3.23. So, for the aforementioned resolvent $w_l^{ind} \Rightarrow \mathbf{E}_{\langle ind \rangle} \bigcirc (\neg L \vee l)$, V' satisfies w_l^{ind} but V does not satisfy $(\neg L \vee l)$. Thus, the labelled behaviour graph for T' does not contain an edge (n', ind, n). By the construction of the labelled behaviour graph for T', n is not a node in it.
2. In the case of (2), we have $V \models Q, V \models L, V \models \neg l$. Thus, 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 $l_{\langle ind \rangle}$.

$\square$

The behaviour graph construction and the use of deletion rules to remove parts of the behaviour graph which cannot contribute to the model structure for a given clause set is commonly used in completeness proofs for temporal and modal resolution calculi [38, 56] and resembles the tableau construction and marking procedure for propositional dynamic logic PDL in [60].

3.5.3 Termination

In this section, we show that all the derivations obtained by applying rules of the calculus $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$ to an arbitrary finite set of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses terminate.

Theorem 3.8 *Any derivation from a finite set T of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses by the calculus $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$ terminates.*

Proof. Assume T is augmented. Let T be constructed from a set Θ of n atomic propositions and a set *Ind* of m indices. Then the number of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses constructed from Θ 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 can be at most $(m+1)2^{4n} + (m \cdot n + n + 1)2^{2n+1}$ different $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses. Any derivation from a set of $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses by the calculus $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$ will terminate when either no more new clauses can be derived or a contradiction is obtained. Since there are only a finitely bounded number of different $\mathrm{SNF}_{\mathrm{CTL}}^{\mathrm{g}}$ clauses, one of these two conditions will eventually be true. $\square$

3.6 Complexity

The satisfiability problem of CTL is known to be EXPTIME-complete [22, 31, 34]. Next we consider the complexity of the decision procedure based on $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$ and presented in Section 3.4.4.

Theorem 3.9 *The complexity of the $\mathsf{R}_{\mathrm{CTL}}^{\succ,S}$-based decision procedure is in EXPTIME.*

Proof. Assume that the set N of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is augmented and is constructed from a set Θ 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, compare literals in constant time and check the identity of indices in constant time. From the proof of Theorem 3.8, 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 $\mathsf{C} = \frac{\mathsf{B}(\mathsf{B}-1)}{2}$ combinations 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 $\mathsf{D} = \mathsf{B} \cdot (8n^2 + 1)$. 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. According to the proof of Theorem 3.8, there can be at most different B determinate clauses. Therefore, the complexity of saturating a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses by step resolution is the order of $(\mathsf{C} \cdot \mathsf{A} \cdot \mathsf{D}) \cdot \mathsf{B}$. That is the complexity of *resolution_sres* (line 8) is in 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 [16]. The CTL loop search algorithm is a variation of the PLTL loop search algorithm [27] which has been shown to be in EXPTIME and we can show that the complexity of the CTL loop search algorithm from [16] is also in EXPTIME. Generally speaking, each iteration of the CTL loop search algorithm is a saturation of the clause set, which is in EXPTIME, and there may be an exponential number of iterations required. Therefore, the complexity of *resolution_eres* (line 12) is in EXPTIME. According to the proof of Theorem 3.8, there can be at most distinct $n \cdot 2^{2n+1}$ **A**-sometime clauses and $m \cdot n \cdot 2^{2n+1}$ **E**-sometime clauses. Thus, the number of iterations for the for-loop (line 11 to 16) is at most $(n \cdot 2^{2n+1} + m \cdot n \cdot 2^{2n+1})$.

As we know, the cost of deciding whether a clause is existing in the set `Old` is D and the number of clauses in the set `New` at line 17 is at most B. Therefore, the cost of the operation `New` $\setminus$ `Old` is $\mathsf{D} \cdot \mathsf{B}$ (line 17). Analogously we can obtain that the cost of the operation `New` $\cup$ `G` (line 14) is bounded by $\mathsf{D} \cdot \mathsf{B}$.

Thus, the complexity of each iteration of the do-while-loop (line 8 to 18) is in EXPTIME. From the proof of Theorem 3.8, there can be at most B different determinate clauses. The number of iterations of the do-while-loop is at most B. Therefore, the complexity of the decision procedure is in EXPTIME. □

3.7 Related work

3.7.1 Comparison between $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ and the previous resolution calculus

$\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is based on Bolotov's resolution calculus for CTL [15]. For instance, the use of indices to transform CTL formulae into Separated Normal Form for CTL was introduced in [15]. However, no formal interpretation was given for indices and no formal semantics stated for $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$. In this thesis, we provide a formal semantics for $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$.

Compared to the definition of $\mathrm{SNF}_{\mathrm{CTL}}$ in [15], we use an additional type of clause, namely *global clauses*. Our definition of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ provides several advantages over [15]. 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 $m_1 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc l$ and $m_2 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\neg l$ we can derive $m_1 \wedge m_2 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\mathbf{false}$. Both $m_1 \wedge m_2 \Rightarrow \mathbf{A}\bigcirc\mathbf{false}$ and $m_1 \wedge m_2 \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc\mathbf{false}$ 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 [15] does not allow for such clauses, in the approach taken in [15] 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 [15] 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 [15] we will have fewer clauses transformed from the original CTL formula and generated by resolution.

In [15], Bolotov provides a set of rules for the transformation of CTL formulae into a clausal normal form. The transformation rules are shown to preserve satisfiability. However the proof for termination of the transformation process and the proof that the result of the process is indeed in normal form are absent and the complexity of the process is not studied. In contrast, we have provided the corresponding proofs and an analysis of the complexity for our transformation process.

Below, we provide a concrete example to demonstrate that our transformation rules are more efficient in terms of new atomic propositions introduced and the number of clauses generated during the transformation by comparing two transformation processes to the same CTL formula $\mathbf{A}\square(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q))$ we have used in Example 3.1.

Example 3.12

1. We apply the function *init* to the CTL formula $\varphi = \mathbf{A}\square(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q))$:

$$\Gamma = \mathit{init}(\varphi) = \{\mathbf{A}\square(\mathbf{start} \Rightarrow p_1), \mathbf{A}\square(p_1 \Rightarrow \mathbf{A}\square(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q)))\}$$

2. We transform the set Γ of clauses into a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses using Bolotov's transformation rules as given in Appendix A[1]. In the following table, $\mathit{Trans}(x) \rightarrow y$ indicates applying the

[1]The set of the transformation rules presented in Appendix A, which has been used by the earlier versions of our CTL prover CTL-RP, is slightly different from Bolotov's transformation rules. The only difference is that the rules in Appendix A do not rewrite a global clause into an initial clause and an **A**-step clause. Therefore, it is slightly better than Bolotov's transformation rules in terms of the number of clauses generated during the transformation.

transformation rule *Trans*(x) to the yth clause in the table.

1.	$\mathbf{start} \Rightarrow p_1$	
2.	$p_1 \Rightarrow \mathbf{A}\Box(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q))$	
3.	$p_1 \Rightarrow \mathbf{A}\Box p_2$	$Trans(8) \rightarrow 2$
4.	$p_2 \Rightarrow \mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q)$	$Trans(8) \rightarrow 2$
5.	$p_1 \Rightarrow p_2 \wedge p_3$	$Trans(11) \rightarrow 3$
6.	$p_3 \Rightarrow \mathbf{A}\bigcirc(p_2 \wedge p_3)$	$Trans(11) \rightarrow 3$
7.	$p_2 \Rightarrow \mathbf{E}_{\langle 1\rangle}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q)$	$Trans(3) \rightarrow 4$
8.	$p_1 \Rightarrow p_2$	$Trans(4) \rightarrow 5$
9.	$p_1 \Rightarrow p_3$	$Trans(4) \rightarrow 5$
10.	$p_3 \Rightarrow \mathbf{A}\bigcirc p_4$	$Trans(7) \rightarrow 6$
11.	$p_4 \Rightarrow p_2 \wedge p_3$	$Trans(7) \rightarrow 6$
12.	$p_2 \Rightarrow \mathbf{E}_{\langle 1\rangle}(p_5\,\mathcal{U}\,q)$	$Trans(9) \rightarrow 7$
13.	$p_5 \Rightarrow \mathbf{E}\bigcirc r$	$Trans(9) \rightarrow 7$
14.	$\mathbf{true} \Rightarrow \neg p_1 \vee p_2$	$Trans(6) \rightarrow 8$
15.	$\mathbf{true} \Rightarrow \neg p_1 \vee p_3$	$Trans(6) \rightarrow 9$
16.	$p_4 \Rightarrow p_2$	$Trans(4) \rightarrow 11$
17.	$p_4 \Rightarrow p_3$	$Trans(4) \rightarrow 11$
18.	$p_2 \Rightarrow q \vee (p_5 \wedge p_6)$	$Trans(14) \rightarrow 12$
19.	$p_6 \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc(q \vee (p_5 \wedge p_6))$	$Trans(14) \rightarrow 12$
20.	$p_2 \Rightarrow \mathbf{E}_{\langle 1\rangle}\Diamond q$	$Trans(14) \rightarrow 12$
21.	$p_5 \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc r$	$Trans(1) \rightarrow 13$
22.	$\mathbf{true} \Rightarrow \neg p_4 \vee p_2$	$Trans(6) \rightarrow 16$
23.	$\mathbf{true} \Rightarrow \neg p_4 \vee p_3$	$Trans(6) \rightarrow 17$
24.	$p_2 \Rightarrow q \vee p_7$	$Trans(5) \rightarrow 18$
25.	$p_7 \Rightarrow p_5 \wedge p_6$	$Trans(5) \rightarrow 18$
26.	$p_6 \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc p_8$	$Trans(7) \rightarrow 19$
27.	$p_8 \Rightarrow q \vee (p_5 \wedge p_6)$	$Trans(7) \rightarrow 19$
28.	$\mathbf{true} \Rightarrow \neg p_2 \vee q \vee p_7$	$Trans(6) \rightarrow 24$
29.	$p_7 \Rightarrow p_5$	$Trans(4) \rightarrow 25$
30.	$p_7 \Rightarrow p_6$	$Trans(4) \rightarrow 25$
31.	$p_8 \Rightarrow q \vee p_9$	$Trans(5) \rightarrow 27$
32.	$p_9 \Rightarrow p_5 \wedge p_6$	$Trans(5) \rightarrow 27$
33.	$\mathbf{true} \Rightarrow \neg p_7 \vee p_5$	$Trans(6) \rightarrow 29$
34.	$\mathbf{true} \Rightarrow \neg p_7 \vee p_6$	$Trans(6) \rightarrow 30$

35.	$\mathbf{true} \Rightarrow \neg p_8 \vee q \vee p_9$	$\mathit{Trans}(6) \rightarrow 31$
36.	$p_9 \Rightarrow p_5$	$\mathit{Trans}(4) \rightarrow 32$
37.	$p_9 \Rightarrow p_6$	$\mathit{Trans}(4) \rightarrow 32$
38.	$\mathbf{true} \Rightarrow \neg p_9 \vee p_5$	$\mathit{Trans}(6) \rightarrow 36$
39.	$\mathbf{true} \Rightarrow \neg p_9 \vee p_6$	$\mathit{Trans}(6) \rightarrow 37$

3. We obtain the set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses consisting of the following clause, which is satisfiable iff the CTL formula $\mathbf{A}\Box(\mathbf{E}(\mathbf{E}\bigcirc r\,\mathcal{U}\,q))$ is satisfiable.

1. $\mathbf{start} \Rightarrow p_1$
10. $p_3 \Rightarrow \mathbf{A}\bigcirc p_4$
14. $\mathbf{true} \Rightarrow \neg p_1 \vee p_2$
15. $\mathbf{true} \Rightarrow \neg p_1 \vee p_3$
20. $p_2 \Rightarrow \mathbf{E}_{\langle 1\rangle}\Diamond q$
21. $p_5 \Rightarrow \mathbf{E}_{\langle 2\rangle}\bigcirc r$
22. $\mathbf{true} \Rightarrow \neg p_4 \vee p_2) \rightarrow 16$
23. $\mathbf{true} \Rightarrow \neg p_4 \vee p_3$
26. $p_6 \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc p_8$
28. $\mathbf{true} \Rightarrow \neg p_2 \vee q \vee p_7$
33. $\mathbf{true} \Rightarrow \neg p_7 \vee p_5$
34. $\mathbf{true} \Rightarrow \neg p_7 \vee p_6$
35. $\mathbf{true} \Rightarrow \neg p_8 \vee q \vee p_9$
38. $\mathbf{true} \Rightarrow \neg p_9 \vee p_5$
39. $\mathbf{true} \Rightarrow \neg p_9 \vee p_6$

In Example 3.1 we have used our own transformation rules to transform the set Γ into a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses. The table below compares the results with respect to the number of clauses generated during the transformation process, the number of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses obtained at the end of the process, and the number of new propositions occurring in these clauses.

	Example 3.1	Example 3.12
No. of clauses generated totally	13	39
No. of clauses in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$	7	15
No. of new propositions introduced	3	9

This example illustrates the improvements that our transformation rules can offer compared with the transformation rules in [15]. In addition, as we have implemented both transformation processes in our prover CTL-RP, which will be discussed in the next chapter, we have experimented with them on many examples and the result of this empirical study also indicates that our transformation is more efficient. More details can be found in Section 4.4.

$$\textbf{TRES1}\quad \frac{\begin{array}{l} P^{\dagger} \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l \\ q \Rightarrow \mathbf{A}\Diamond\neg l \end{array}}{q \Rightarrow \mathbf{A}(\neg P^{\dagger}\,\mathcal{W}\,\neg l)} \qquad \textbf{TRES2}\quad \frac{\begin{array}{l} P^{\dagger} \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l \\ q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l \end{array}}{q \Rightarrow \mathbf{E}_{\langle ind\rangle}(\neg P^{\dagger}\,\mathcal{U}\,\neg l)}$$

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

Figure 3.9: Redundant eventuality resolution rules

Another difference to [15] is the approach taken in our completeness proof. The proof in [15] tries to relate 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 the deletion process. In contrast, 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 [38] for proving completeness of a clausal resolution for PLTL and related to the concept of a labelled behaviour graph used [29]. However, our labelled behaviour graph differs in its construction to capture the semantics of indices in $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$. We believe our completeness proof 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 and the labelled behaviour graph for a set T of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses saturated under $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ that contains no contradiction, is a reduced labelled behaviour graph. Hence, we could potentially use the labelled behaviour graph construction to generate counter models for failed proofs. Our labelled behaviour graph can be easily extended so that it can be used in completeness proofs of resolution calculi for the combination of CTL and other logics, for example, the combination of CTL and modal logic KD45 [29].

Furthermore, in the resolution calculus for CTL presented in [15, 29] step resolution is not constrained by an ordering and a selection function. Therefore, the step resolution rules in [15, 29] 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, respectively. The other two eventuality resolution rules are given in Figure 3.9. Using our completeness proof we can prove that the two eventuality resolution rules TRES1 and TRES2 in [15, 29] are redundant.

We give a brief explanation why this is the case. Informally, 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 [15, 29], $\mathbf{A}\bigcirc\mathbf{A}\Box l$ 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 [15] 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 [15]. In this thesis, we prove that a decision procedure based on $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is of the order EXPTIME.

3.7.2 Other approaches for the satisfiability problem of CTL

There are other approaches, which can also be used to solve the satisfiability problem of CTL, namely automata techniques [71] and tableau calculi [3, 34, 31, 10]. We give a brief discussion on these two approaches.

The automata-based decision procedure for CTL [71] consists of two separated phases. Firstly, the decision procedure constructs an automaton A for a given CTL formula φ. This constructed automaton has a very useful property that it accepts certain infinite tree models of φ iff the formula φ is satisfiable. In the second phase, the decision procedure checks whether there exists a tree model accepted by A, i.e. whether the language accepted by A is non-empty. If there is one tree model accepted by A, then φ is satisfiable. Otherwise, φ is unsatisfiable. Constructing the automaton requires exponential time in the size of φ while checking the emptiness of the language accepted by the automaton requires polynomial time in the size of A.

For a tableau calculus for CTL, usually two types of rules are defined for the calculus, namely the construction rules and the deletion rules. For the construction rules, they are used to expand an arbitrary CTL formula φ into a (possibly cyclic) graph such that each node of this graph is a set of CTL formulae. The deletion rules of a tableau calculus for CTL determine which nodes should be removed from the constructed tableau. A node n is eliminated if any one of the following condition is satisfied: (i) there are inconsistencies in the node, for example p and $\neg p$; (ii) the node has certain requirements on its successors and those requirements are not fulfilled by its successors; or (iii) the node contains unrealised eventualities, for example, $\mathbf{E}\Diamond p$ or $\mathbf{A}(p\,\mathcal{U}\,q)$ in the node n and can not be realised in any node reachable from the node n as well. Generally speaking, the deletion rules remove all the nodes which are not able to be used to construct a CTL model satisfying the CTL formula φ.

If the tableau has no nodes left after the deletion process, the tableau is a *closed tableau* for CTL and the CTL formula φ is unsatisfiable, otherwise the tableau is an *open tableau* for CTL and φ is satisfiable.

3.8 Conclusions

CTL [31] was introduced by Emerson et al in the 1980s and now is a well-known branching-time temporal logic for the specification and verification of computational systems. Approaches to the satisfiability problem in CTL include automata techniques [71], tableau calculi [3, 34] and a resolution calculus [15], developed by Bolotov.

Bolotov's calculus for CTL is based on the ideas underlying a resolution calculus for PLTL [38]. Here, we have provided a refined clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL. Compared with [15], we use an ordering and a selection function to restrict the applicability of step resolution rules and we have fewer eventuality resolution rules. We present a new completeness proof based on labelled behaviour graphs. Our completeness proof demonstrates a closer relationship between applications of resolution rules and deletions on labelled behaviour graphs. The proof shows that the additional eventuality resolution rules in [15], which are the most costly rules, are redundant. In addition, we prove that the complexity of a $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$-based decision procedure for CTL is EXPTIME.

Chapter 4

CTL-RP: A resolution theorem prover for CTL

4.1 Introduction

There are many CTL theorem proving methods, including resolution-based [15], tableau-based [3, 34, 62], and automata-based [36] methods. However, only the method in [3] has recently been implemented in the Tableau Workbench [2]. Due to the branching structure of CTL models and the presence of fixpoint operators in the logic, implementing an efficient theorem proving method for CTL is conceptually complex.

Here we present the first implemented resolution theorem prover for CTL, CTL-RP, which realises the sound and complete clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ introduced in Chapter 3. Moreover, $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is designed such that it allows the use of classical first-order resolution techniques to implement its rules. We take advantage of this fact in the development of our prover CTL-RP by basing its implementation on the first-order theorem prover SPASS [54, 75]. Therefore, our approach dramatically reduces the implementation effort. We experimentally compare the only theorem prover for CTL that we know of, namely the Tableau Workbench, with CTL-RP and the results obtained from this empirical study show good performance of CTL-RP.

The overall approach we take is as follows. The clausal normal form for CTL, on which the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ operates, consists of six types of clauses: initial clauses, global clauses, **A**-step clauses, **E**-step clauses, **A**-sometime clauses and **E**-sometime clauses. The calculus itself consists of step resolution, eventuality resolution and rewrite rules. In our implementation of the calculus, we transform all clauses except **A**- and **E**-sometime clauses into first-order clauses. This enables us to emulate the step resolution rules by first-order ordered resolution with selection. As to implementation of the first-order calculus, we use the theorem prover SPASS. To fully realise $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ we have to extend SPASS with an implementation of the eventuality resolution rules. As we will see, the rewrite rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ become superfluous in this approach. It should be noted that the most of the content related to the implementation of the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ in this chapter has been published in [79].

The rest of the chapter is organised as follows. In Section 4.2 firstly we introduce first-order ordered resolution with selection. Secondly, we define how to transform initial clauses, global clauses, and step clauses into first-order clauses. Thus we are able to apply first-order resolution rules to those clauses. Thirdly, we show our approach to the implementation of the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ using first-order resolution techniques. Furthermore, we list all the algorithms involved in the decision procedure CTL-RP implements. In Section 4.3 we discuss related work, in particular, the resolution-based prover TRP++ for Propositional Linear-time Temporal Logic (PLTL) which also uses first-order techniques and the tableau-based prover TWB for CTL. In Section 4.4, we experimentally compare CTL-RP with the tableau theorem prover TWB. Finally, we draw conclusions in Section 4.5.

4.2 Implementation of the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$

In order to realise the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ and its associated decision procedure for CTL, we adopt an approach analogous to that used in [48] for the implementation of a resolution calculus [38] for PLTL.

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 the first-order ordered resolution with selection calculus [11] to emulate step resolution. **A**-sometime clauses 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**-sometime clauses and **E**-sometime clauses, respectively, and use the loop search algorithm presented in Section 3.4 to find suitable premises for these rules. We also 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.

4.2.1 Preliminaries of first-order ordered resolution with selection

In order to present our approach of using first-order techniques to carry out resolution for CTL, we first introduce several notions following their definition in [11, 47, 67, 51].

The language of first-order logic is based on

- the boolean operators $\neg$ (not), $\land$ (and), $\lor$ (or), and $\Rightarrow$ (implies);
- the universal quantifier $\forall$ (for all) and the existential quantifier $\exists$ (there exists);
- a countable set V of *variables*;
- a countable set F of *function symbols* such that each function symbol in F is associated with an *arity* $n \in \mathbb{N}_0$;
- a countable set P of *predicate symbols* such that each predicate symbol in P is associated with an *arity* $n \in \mathbb{N}_0$;

We assume that the sets V, F and P are pairwise disjoint. The pair (F, P) is called a *signature*.

Definition 4.1 *Term*
A *term* is either a variable in V or an expression $f(t_1, \ldots, t_n)$ where $f \in \mathsf{F}$ is a function symbol of arity n and $t_1, \ldots, t_n$ are terms. Function symbols of arity zero are also called *constant symbols*. For a term $t = f(t_1, \ldots, t_n)$ the terms $t_1, \ldots, t_n$ 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, that is, it is an element of $\mathsf{T}(\mathsf{F}, \emptyset)$.

Definition 4.2 *Depth of a term*
The *depth* $dp(t)$ of a term is inductively defined as

1. if t is a variable or a constant then $dp(t) = 1$, and
2. if $t = f(t_1, \ldots, t_n)$, then

$$dp(t) = 1 + \max(\{dp(t_i) \mid 1 \leq i \leq n\})$$

For example, $dp(a) = 1$; $dp(f(x, y)) = 2$; and $dp(r(h(x, g(x)))) = 4$, where a is a constant, x and y are variables, and f, g, r and s are function symbols.

Definition 4.3 *Atom*
An *atom* is an expression $p(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are terms in $\mathsf{T}(\mathsf{F}, \mathsf{V})$ and $p \in \mathsf{P}$ is a predicate symbol of arity n. Predicate symbols of arity zero are also called *propositions*.

Definition 4.4 *Literal*
A *literal* is an expression A (a *positive literal*) or $\neg A$ (a *negative literal*) where A is an atom. For a literal $L = (\neg)p(t_1, \ldots, t_n)$ the terms $t_1, \ldots, t_n$ are the *arguments* of L. A literal is *ground* if all its arguments are ground.

Definition 4.5 *Depth of a literal*
The *depth* $dp(L)$ of a literal $L = (\neg)p(t_1, \ldots, t_n)$ 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. For example, $dp(p(h(x, y), r(h(x, g(x))))) = 4$, where x and y are variables; g, h, and r are function symbols; and p is a predicate.

Definition 4.6 *First-order clause*
A first-order *clause* $C = L_1 \vee L_2 \vee \ldots \vee L_n$ 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 and we write $D \subseteq C$ if D is a subclause of C. A *strict subclause* D of a clause C is a subclause of C not identical to C.

Definition 4.7 *Depth of a first-order clause*
The *depth* $dp(C)$ of a first-order clause C is given by $\max(\{dp(L) \mid L \in C\})$ if C is non-empty, and $dp(C) = 0$ otherwise. For example, $dp(p(x) \vee q(f(x, y))) = 2$, where x and y are variables; and p and q are predicates.

Definition 4.8 *Expression*
An expression is a term, an atom or a literal.

Definition 4.9 *Substitution*
A *substitution* is a mapping from variables to terms which is the identity mapping almost everywhere. A substitution σ can be represented as a finite set of pairs $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$, where $x_i \neq t_i$ for all i, $1 \leq i \leq n$, and $x_i \neq x_j$ for all i, j, $1 \leq i < j \leq n$. A substitution $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ is said to be a *ground substitution*, if every $t_i, 1 \leq i \leq n$, is a ground term. The value of a substitution σ for a variable x is denoted by $x\sigma$. A substitution can be extended to a mapping from terms to terms. Analogously, for atoms, literals, and clauses. The result of the *application* of a substitution σ to an expression E is denoted by $E\sigma$.

Definition 4.10 *Composition of substitutions*
The *composition* $\theta\sigma$ of the substitutions σ and θ is defined as $x\theta\sigma = (x\theta)\sigma$ for all variables x in V. A substitution σ is *idempotent* if $\sigma\sigma = \sigma$.

Definition 4.11 *Variable renaming*
A *variable renaming* is an injective substitution σ such that $x\sigma$ is a variable for every variable $x \in \mathsf{V}$. An expression E' is a *variant* of an expression E if there exists a variable renaming σ such that $E' = E\sigma$. We say E and E' are *identical modulo variable renaming.* We usually consider clauses C and D to be identical if they are identical modulo variable renaming.

Definition 4.12 *Instance*
An expression E' is an *instance* of an expression E if there exists a substitution σ such that $E' = E\sigma$.

Definition 4.13 *Tautology*
A first-order clause is a tautology iff it contains a pair of complementary literals.

Definition 4.14 *Subsumption*
A first-order clause C subsumes a first-order clause D if there exists a substitution σ such that $C\sigma \subseteq D$.

Definition 4.15 *Unifier*
A substitution σ is a *unifier* of expressions $E_1, \ldots, E_n$ if $E_i\sigma = E_j\sigma$ for all i, j, $1 \leq i, j \leq n$, and $E_1, \ldots, E_n$ are said to be *unifiable*. A unifier σ is a *most general unifier* of $E_1, \ldots, E_n$ if for every unifier θ of $E_1, \ldots, E_n$ there exists a substitution ρ such that $x\theta = x(\sigma\rho)$ for every variable x occurring in $E_1, \ldots, E_n$.

Definition 4.16 *Condensation*
A *condensation* $\mathit{Cond}(C)$ of a clause C is a minimal subclause of C such that there exists a substitution σ such that $C\sigma$ contains the same set of literals as $\mathit{Cond}(C)$. A clause C is *condensed* if there exists no condensation of C which is a strict subclause of C.

For example, let $C_1 = f(x) \vee f(x) \vee g(x)$ and $C_2 = h(x) \vee h(y) \vee g(x)$ be first-order clauses, then $\mathit{Cond}(C_1) = f(x) \vee g(x)$ and $\mathit{Cond}(C_2) = h(x) \vee g(x)$. Note that the condensation $\mathit{Cond}(C)$ of a clause C is either identical to C or is a strict subclause of C which subsumes C according to the definition of subsumption we give above.

Definition 4.17 *Atom ordering*
An *atom ordering* $\succ_{FOL}$ is a well-founded, total ordering on ground atoms. For two non-ground atoms A and B, we define $A \succ_{FOL} B$ iff $A\sigma \succ_{FOL} B\sigma$ 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.

Definition 4.18 *Selection function*
A *selection function* S_{FOL} assigns to each clause C a possibly empty set of occurrences of negative literals in C. These negative literals in $S_{FOL}(C)$ are called *selected* (in the clause C).

The resolution calculus $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ is parameterised by an atom ordering $\succ_{FOL}$ and a selection function S_{FOL}, and consists of the following two inference rules[1]:

- Ordered resolution with selection

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

  where

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

- Ordered positive factoring with selection

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

  where

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

We now give a number of important definitions related to the calculus $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$.

Definition 4.19 *Redundancy*
Given a set N of clauses, a ground clause C is *redundant with respect to* N if there are ground

[1]We use $\mathtt{I}$ to indicate inference rules.

instances $C_1\sigma, \ldots, C_n\sigma$ of clauses in N such that $C_1\sigma, \ldots, C_n\sigma \models C$ and for each $i, 1 \leq i \leq n$, $C \succ_{FOL} C_i\sigma$. A non-ground clause C is *redundant with respect to* N if every ground instance of C is redundant with respect to N.

Definition 4.20 *Saturation*
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.

Definition 4.21 *Derivation*
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 $\mathsf{R}_{\text{FOL}}^{\succ_{FOL}, S_{FOL}}$ 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* $N_\infty = \bigcup_{j\geq 0} \bigcap_{k \geq j} N_k$ is contained in some set N_j.

Theorem 4.1 *[11]* $\mathsf{R}_{\text{FOL}}^{\succ_{FOL}, S_{FOL}}$ *is a sound and complete refutation calculus: for a set of clauses* N_0 *and a 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.*

During the implementation of CTL-RP, we have found that a particular redundancy elimination rule, called *matching replacement resolution*, can significantly improve the efficiency of our prover.

Matching replacement resolution[2]

$$\mathtt{R} \quad \frac{C \vee A \qquad \neg B \vee D}{C \vee A \qquad D}$$

where $A\sigma = B$ and $C\sigma \subseteq D$.

For example, let the two first-order clauses $\Gamma_1 = q(x) \vee p(x)$ and $\Gamma_2 = \neg p(x) \vee q(x) \vee r(x)$ be in the set N, then N can be reduced to the set $N' = N \cup \{q(x) \vee r(x)\} \setminus \{\Gamma_2\}$, by applying the matching replacement resolution rule to Γ_1 and Γ_2.

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

In order to represent every determinate $\text{SNF}_{\text{CTL}}^{\text{g}}$ clause by a first-order clause we uniquely associate every atomic proposition 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}.

[2]We use $\mathtt{R}$ to indicate redundancy elimination rules.

The constant 0 represents the state $s_0 \in S$ in a 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 $Q_p(\mathsf{0})$ 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 a 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 $P \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C$ 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 (which is equivalent to **false**) in an **A**-step clause $P \Rightarrow \mathbf{A}\bigcirc C$ or an **E**-step clause $P \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C$. In our calculus, 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 disjunction C, the $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses $P \Rightarrow \mathbf{A}\bigcirc C$, $P \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc C$, 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.

We now give a number of concrete examples:

$\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses	First-order representation
start $\Rightarrow p \vee \neg r$	$Q_p(\mathsf{0}) \vee \neg Q_r(\mathsf{0})$
$r \Rightarrow \mathbf{E}_{\langle 1\rangle}\bigcirc p$	$\neg Q_r(x) \vee Q_p(\mathsf{app}(s_1, x))$
$\neg p \wedge q \Rightarrow \mathbf{A}\bigcirc$**false**	$Q_p(x) \vee \neg Q_q(x)$
true $\Rightarrow p \vee \neg q$	$Q_p(x) \vee \neg Q_q(x)$
start $\Rightarrow$ **false**	$\perp$

4.2.3 Implementing step resolution

The representation of determinate $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses by first-order clauses allows for all our step resolution rules to be implemented using the first-order ordered resolution with selection calculus presented in Section 4.2.1.

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

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

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

$$\begin{aligned} Q_q(\mathsf{app}(s_{ind}, x)) &\succ_{FOL} Q_p(\mathsf{app}(s_{ind'}, x)) \\ Q_q(x) &\succ_{FOL} Q_p(x) \end{aligned}$$

provided $q \succ p$.

Regarding the first-order selection function S_{FOL}, we use the close correspondence between determinate $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ 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 in C by S;
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 $\lceil \neg P \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s, x))$ by S_{FOL}, with C being non-empty, iff $\neg l$ is selected in C by S;
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 multiple occurrences of the same literal, 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 clause 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.

Theorem 4.2 *Let $\succ$ and S be an atom ordering and a selection function, respectively, for $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ and let $\succ_{FOL}$ and S_{FOL} be a corresponding atom ordering and a corresponding selection function,*

An application of SRES1

$$\begin{array}{c} P \Rightarrow \mathbf{A}\bigcirc(C \lor l) \\ Q \Rightarrow \mathbf{A}\bigcirc(D \lor \neg l) \\ \hline P \land Q \Rightarrow \mathbf{A}\bigcirc(C \lor D) \end{array} \quad \begin{array}{l} \text{where } l \text{ is eligible in } C \lor l \\ \text{and } \neg l \text{ is eligible in } D \lor \neg l \end{array}$$

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

$$\begin{array}{c} \lceil \neg P \rceil(x) \lor \lceil C \rceil(\mathsf{app}(w,x)) \lor Q_l(\mathsf{app}(w,x)) \\ \lceil \neg Q \rceil(y) \lor \lceil D \rceil(\mathsf{app}(z,y)) \lor \neg Q_l(\mathsf{app}(z,y)) \\ \hline \mathrm{Cond}(\lceil \neg P \rceil(x) \lor \lceil \neg Q \rceil(x) \lor \lceil C \rceil(\mathsf{app}(w,x)) \lor \lceil D \rceil(\mathsf{app}(w,x))) \end{array}$$

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

Figure 4.1: Emulating SRES1 inferences in first-order logic

An application of SRES2

$$\begin{array}{c} P \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(C \lor l) \\ Q \Rightarrow \mathbf{A}\bigcirc(D \lor \neg l) \\ \hline P \land Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc(C \lor D) \end{array} \quad \begin{array}{l} \text{where } l \text{ is eligible in } C \lor l \\ \text{and } \neg l \text{ is eligible in } D \lor \neg l \end{array}$$

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

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

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 $\sigma = \{y \leftarrow x, z \leftarrow s_{ind}\}$.

Figure 4.2: Emulating SRES2 inferences in first-order logic

respectively, for $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$. Let Γ_1 and Γ_2 be two determinate clauses. Then a determinate clause Γ_3 is derivable from Γ_1 and Γ_2 by SRES1 to SRES8 in $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ iff there exists a clause C derivable from $\lceil\Gamma_1\rceil$ and $\lceil\Gamma_2\rceil$ by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ 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 $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ 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 $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$. Figures 4.1 to 4.8 show this relationship for inferences by SRES1 to SRES8, respectively.

We also show that provided we keep first-order clauses in condensed form, $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ 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 clause C^i, a global clause C^g, an **A**-step clause C^A or an **E**-step clause C^E. We can distinguish ten different types of inference, C^iC^g, 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, where C^iC^g indicates an inference

An application of SRES3

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

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

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

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 $\sigma = \{y \leftarrow x\}$. Note that for any two literals $Q_l(\mathsf{app}(s_{ind}, x))$ and $Q_l(\mathsf{app}(s_{ind'}, y))$, they are only unifiable if $ind = ind'$.

Figure 4.3: Emulating SRES3 inferences in first-order logic

An application of SRES4

$$\frac{\begin{array}{l} \mathbf{start} \Rightarrow C \vee l \\ \mathbf{start} \Rightarrow D \vee \neg l \end{array}}{\mathbf{start} \Rightarrow C \vee D} \qquad \begin{array}{l} \text{where } l \text{ is eligible in } C \vee l \\ \text{and } \neg l \text{ is eligible in } D \vee \neg l \end{array}$$

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

$$\frac{\begin{array}{l} \lceil C \rceil(0) \vee Q_l(0) \\ \lceil D \rceil(0) \vee \neg Q_l(0) \end{array}}{\text{Cond}(\lceil C \rceil(0) \vee \lceil D \rceil(0))}$$

where the definition of $\succ_{FOL}$ and S_{FOL} ensures that $Q_l(0)$ and $\neg Q_l(0)$ are both eligible in their respective clauses.

Figure 4.4: Emulating SRES4 inferences in first-order logic

involving an initial clause and a global clause, C^iC^A indicates an inference involving an initial clause and an A-step clause, etc. SRES1 to SRES8 map to eight of these types of inference. The two not covered by SRES1 to SRES8 are C^iC^A and C^iC^E, that is, $\mathsf{R}^{\succ,S}_{\text{CTL}}$ does not allow inferences between an initial clause and an **A**-step clause or between an initial clause and an **E**-step clause. We show that $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\text{FOL}}$ also does not allow the corresponding first-order inferences of C^iC^A and C^iC^E.

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

1. Let $Q_q(0) \vee \mathsf{C}$ represent an initial clause and let $\neg Q_q(\mathsf{app}(s, x)) \vee \mathsf{D}$ represent an **A**-step clause. Here, $Q_q(0)$ and $\neg Q_q(\mathsf{app}(s, x))$ are not unifiable.

2. Let $\lceil \Gamma_1 \rceil = Q_q(0) \vee \mathsf{C}$ represent an initial clause Γ_1 and let $\lceil \Gamma_2 \rceil = \neg Q_q(x) \vee (\neg)Q_p(\mathsf{app}(s, x)) \vee \mathsf{D}$ represent an **A**-step clause Γ_2. $\neg Q_q(x)$ is not eligible in $\lceil \Gamma_2 \rceil$: 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.

An application of SRES5

$$\frac{\begin{array}{c}\mathbf{true} \Rightarrow C \vee l\\ \mathbf{start} \Rightarrow D \vee \neg l\end{array}}{\mathbf{start} \Rightarrow C \vee D} \quad \begin{array}{l}\text{where } l \text{ is eligible in } C \vee l\\ \text{and } \neg l \text{ is eligible in } D \vee \neg l\end{array}$$

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

$$\frac{\begin{array}{l}\lceil C\rceil(x) \vee Q_l(x)\\ \lceil D\rceil(0) \vee \neg Q_l(0)\end{array}}{\mathrm{Cond}(\lceil C\rceil(0) \vee \lceil D\rceil(0))}$$

where the definition of $\succ_{FOL}$ and S_{FOL} ensures that $Q_l(x)$ and $\neg Q_l(0)$ are both eligible in their respective clauses for the substitution $\sigma = \{x \leftarrow 0\}$.

Figure 4.5: Emulating SRES5 inferences in first-order logic

An application of SRES6

$$\frac{\begin{array}{c}\mathbf{true} \Rightarrow C \vee l\\ Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l)\end{array}}{Q \Rightarrow \mathbf{A}\bigcirc(C \vee D)} \quad \begin{array}{l}\text{where } l \text{ is eligible in } C \vee l\\ \text{and } \neg l \text{ is eligible in } D \vee \neg l\end{array}$$

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

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

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

Figure 4.6: Emulating SRES6 inferences in first-order logic

3. Let $\neg Q_q(0) \vee \mathsf{C}$ represent an initial clause and let $Q_q(\mathsf{app}(s,x)) \vee \mathsf{D}$ represent an **A**-step clause. Again $\neg Q_q(0)$ and $Q_q(\mathsf{app}(s,x))$ are not unifiable.

4. Let $\lceil\Gamma_1\rceil = \neg Q_q(0) \vee \mathsf{C}$ represent an initial clause Γ_1 and let $\lceil\Gamma_2\rceil = Q_q(x) \vee (\neg)Q_p(\mathsf{app}(s,x)) \vee \mathsf{D}$ represent an **A**-step clause Γ_2. $Q_q(x)$ is not eligible in $\lceil\Gamma_2\rceil$: 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 selects negative literals.

For an inference of type C^iC^E, the proof is analogous. Therefore, inferences of type C^iC^A and C^iC^E are not possible in $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$, just as they are not possible in $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$.

Furthermore, all inferences by SRES1 to SRES8 involve only literals on the right-hand sides of determinate clauses. We prove that inferences in $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$ are subject to an analogous restriction. This is again due to the ordering $\succ_{FOL}$ and the selection function S_{FOL}.

We show it holds for inferences of type C^AC^A.

- Let $\lceil\Gamma_1\rceil = \neg Q_q(x) \vee (\neg)Q_{q_1}(\mathsf{app}(s,x)) \vee \mathsf{C}$ represent an **A**-step clause Γ_1 and let $\lceil\Gamma_2\rceil = Q_q(x) \vee (\neg)Q_{q_2}(\mathsf{app}(s,x)) \vee \mathsf{D}$ represent another **A**-step clause Γ_2. $Q_q(x)$ is not eligible in

An application of SRES7

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

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

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

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 $\sigma = \{x \leftarrow \mathsf{app}(s_{ind}, y)\}$.

Figure 4.7: Emulating SRES7 inferences in first-order logic

An application of SRES8

$$\frac{\begin{array}{l}\mathbf{true} \Rightarrow C \vee l \\ \mathbf{true} \Rightarrow D \vee \neg l\end{array}}{\mathbf{true} \Rightarrow C \vee D} \quad \begin{array}{l}\text{where } l \text{ is eligible in } C \vee l \\ \text{and } \neg l \text{ is eligible in } D \vee \neg l\end{array}$$

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

$$\frac{\begin{array}{l}\lceil C\rceil(x) \vee Q_l(x) \\ \lceil D\rceil(y) \vee \neg Q_l(y)\end{array}}{\text{Cond}(\lceil C\rceil(x) \vee \lceil D\rceil(x))}$$

where the definition of $\succ_{FOL}$ and S_{FOL} ensures that $Q_l(x)$ and $\neg Q_l(y)$ are both eligible in their respective clauses for the substitution $\sigma = \{y \leftarrow x\}$.

Figure 4.8: Emulating SRES8 inferences in first-order logic

$\lceil\Gamma_2\rceil$: 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 selects negative literals.

- Let $\lceil\Gamma_1\rceil = \neg Q_q(\mathsf{app}(s, x)) \vee \mathsf{C}$ represent an **A**-step clause Γ_1 and let $\lceil\Gamma_2\rceil = Q_q(x) \vee (\neg)Q_p(\mathsf{app}(s, x)) \vee \mathsf{D}$ represent another **A**-step clause Γ_2. $Q_q(x)$ is not eligible in $\lceil\Gamma_2\rceil$: 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 selects negative literals.

- Let $\lceil\Gamma_1\rceil = Q_q(\mathsf{app}(s, x)) \vee \mathsf{C}$ represent an **A**-step clause Γ_1 and let $\lceil\Gamma_2\rceil = \neg Q_q(x) \vee (\neg)Q_p(\mathsf{app}(s, x)) \vee \mathsf{D}$ represent another **A**-step clause Γ_2. $\neg Q_q(x)$ is not eligible in $\lceil\Gamma_2\rceil$: Condition (i) in the definition of $\succ_{FOL}$ ensures that $(\neg)Q_p(\mathsf{app}(s, x))\sigma \succ_{FOL} \neg Q_q(x)\sigma$ for any substitution σ and S_{FOL} is defined in such a way that $\neg Q_q(x)$ is not selected.

It also holds for inferences of type $C^E C^E$.

- Let $\lceil\Gamma_1\rceil = \neg Q_q(x) \vee (\neg)Q_{q_1}(\mathsf{app}(s_{ind}, x)) \vee \mathsf{C}$ represent an **E**-step clause Γ_1 and let $\lceil\Gamma_2\rceil = Q_q(x) \vee (\neg)Q_{q_2}(\mathsf{app}(s_{ind}, x)) \vee \mathsf{D}$ represent another **E**-step clause Γ_2. $Q_q(x)$ is not eligible in

```
procedure $eres$(T, C)
// T is a saturated set of determinate clauses
// C is a sometime clause $Q \Rightarrow \mathbf{A}\Diamond\neg l$ or $Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$
begin
    if C is an $\mathbf{A}$-sometime clause then
      SOS := {D | D is a global or step clause in T};
    else if C is an $\mathbf{E}$-sometime clause then
      SOS := {D | D is a global, $\mathbf{A}$-step, or $\mathbf{E}$-step clause with
      the index $ind$ in T};
    end if
    i := 0;
    $\mathtt{H}_{-1}(x)$ := $\mathbf{true}$;
    do
        Goals := $\{ls(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee \neg \mathtt{H}_{i-1}(x)\sigma\}$, where $\sigma = \{x \leftarrow \mathsf{app}(s,x)\}$;
        $\mathtt{T}_1$ := $resolution\_sos$(SOS, Goals);
        $\mathtt{T}_2$ := $\{G(x) \mid G(x) \vee ls(x) \in \mathtt{T}_1$ and $depth(G(x)) \leq 1\}$;
        $\mathtt{H}_i(x)$ := $\neg(\bigwedge \mathtt{T}_2)$;
        if $\mathtt{H}_i(x)$ is equivalent to $\mathbf{true}$ then
            return $eresolvent$(C, $\mathbf{true}$);
        else if $\mathtt{H}_i(x)$ is equivalent to $\mathbf{false}$ then
            return $\emptyset$
        else if $\mathtt{H}_i(x)$ is equivalent to $\mathtt{H}_{i-1}(x)$ then
            return $eresolvent$(C, $\mathtt{H}_i(x)$);
        end if
        i := i+1;
    while ($\mathtt{T}_2 \neq \emptyset$)
end
```

Figure 4.9: *eres*: A loop search implementation using first-order resolution

$\lceil\Gamma_2\rceil$: 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 selects negative literals.

- Let $\lceil\Gamma_1\rceil = \neg Q_q(\mathsf{app}(s_{ind},x)) \vee \mathsf{C}$ represent an **E**-step clause Γ_1 and let $\lceil\Gamma_2\rceil = Q_q(x) \vee (\neg)Q_p(\mathsf{app}(s_{ind},x)) \vee \mathsf{D}$ represent another **E**-step clause Γ_2. $Q_q(x)$ is not eligible in $\lceil\Gamma_2\rceil$: Condition (i) in the definition of $\succ_{FOL}$ ensures that $(\neg)Q_p(\mathsf{app}(s_{ind},x))\sigma \succ_{FOL} Q_q(x)\sigma$ for any substitution σ and S_{FOL} only selects negative literals.

- Let $\lceil\Gamma_1\rceil = Q_q(\mathsf{app}(s_{ind},x)) \vee \mathsf{C}$ represent an **E**-step clause Γ_1 and let $\lceil\Gamma_2\rceil = \neg Q_q(x) \vee (\neg)Q_p(\mathsf{app}(s_{ind},x)) \vee \mathsf{D}$ represent another **E**-step clause Γ_2. $\neg Q_q(x)$ is not eligible in $\lceil\Gamma_2\rceil$: Condition (i) in the definition of $\succ_{FOL}$ ensures that $(\neg)Q_p(\mathsf{app}(s_{ind},x))\sigma \succ_{FOL} \neg Q_q(x)\sigma$ for any substitution σ and S_{FOL} is defined in such a way that $\neg Q_q(x)$ is not selected.

Proofs for inferences of type C^gC^A, C^gC^E, and C^AC^E 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 $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\mathrm{FOL}}$. □

We now provide an example to show how to use first-order order resolution to implement step resolution rules.

Example 4.1
Previously, in Example 3.3 we considered the set T_3 of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ and derived an empty clause **start** $\Rightarrow$ **false** from it by step resolution. The first-order representation T of the determinate clause set T_3 is as follows.

(1)	$Q_r(\mathsf{0})$	$\longleftarrow$	$\mathbf{start} \Rightarrow r$
(2)	$Q_q(x) \vee Q_p(\mathsf{app}(s_1, x)))$	$\longleftarrow$	$\neg q \Rightarrow \mathbf{E}_{\langle 1 \rangle}\bigcirc p$
(3)	$\neg Q_r(x) \vee \neg Q_q(\mathsf{app}(s_2, x)) \vee Q_p(\mathsf{app}(s_2, x))$	$\longleftarrow$	$r \Rightarrow \mathbf{E}_{\langle 2 \rangle}\bigcirc(\neg q \vee p)$
(4)	$\neg Q_u(x) \vee \neg Q_p(\mathsf{app}(s, x)))$	$\longleftarrow$	$u \Rightarrow \mathbf{A}\bigcirc\neg p$
(5)	$Q_u(x)$	$\longleftarrow$	$\mathbf{true} \Rightarrow u$

We use the ordering based on the precedence $u \succ p \succ q \succ r$ and a selection function which always returns the empty set of negative literals for every clause. Then we can obtain the following derivation from the set T.

[2, R, 4]	(6)	$Q_q(x) \vee \neg Q_u(x)$
[5, R, 6]	(7)	$Q_q(x)$
[3, R, 4]	(8)	$\neg Q_r(x) \vee \neg Q_u(x) \vee \neg Q_q(\mathsf{app}(s_2, x))$
[7, R, 8]	(9)	$\neg Q_r(x) \vee \neg Q_u(x)$
[5, R, 9]	(10)	$\neg Q_r(x)$
[1, R, 10]	(11)	$\bot$

where $[n$, R, $m]$ indicates that we resolve the clauses labelled (n) and (m).

The inferences deriving the determinate clauses $6, 8, 9, 10, 12, 13$ in Example 3.3 corresponds to the inferences in this example deriving the first-order clauses $6, 7, 8, 9, 10, 11$, respectively. On the other hand, the rewrite steps deriving clauses 7 and 11 in Example 3.3 have no correspondence on the first-order level as the rewrite rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ are not required on the first-order level.

4.2.4 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 3.4. Figure 4.9 shows the pseudocode for the implementation of this algorithm in our prover CTL-RP.

The procedure *eres* takes as input

1. 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

2. an **A**-sometime clause or **E**-sometime clause `C`.

As stated in Section 3.4, if `C` is an **A**-sometime clause $Q \Rightarrow \mathbf{A}\Diamond\neg l$, then the loop search algorithm considers all global clauses, **A**-step clauses, and **E**-step clauses in T, while if `C` is an **E**-sometime clause $Q \Rightarrow \mathbf{E}_{\langle ind \rangle}\Diamond\neg l$, then the loop search algorithm considers all global clauses, **A**-step clauses, and 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 in the set $\mathtt{SOS}$. 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 following 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 $\mathtt{C}$ and the loop formula **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 $\mathtt{C}$ and the loop formula $\mathtt{H_i}(x)$ (line 21 and 22).

Lines 13 to 16 deal with the construction of the formula $\mathtt{H_i}(x)$ for the current index $\mathtt{i}$. Recall from Section 3.4 that to construct $\mathtt{H_i}$, we need to look for merged clauses $A_j \Rightarrow (B_j \wedge l)$, $A_j \Rightarrow \mathbf{A}\bigcirc(B_j \wedge l)$ or $A_j \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(B_j \wedge l)$ such that $B_j \Rightarrow \mathtt{H_{i-1}}$ (or, equivalently, $\mathbf{A}\bigcirc B_j \Rightarrow \mathbf{A}\bigcirc\mathtt{H_{i-1}}$). To do so, we construct a set of goal clauses $\mathtt{Goals}$ with each clause containing the literal $\neg Q_l(\mathsf{app}(s, x))$, 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 $\mathbf{A}\bigcirc\neg\mathtt{H_{i-1}}$. When trying to prove these goal clauses using the clauses in $\mathtt{SOS}$, 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 and $ls \succ p$, for all propositions p occurring in the augmented set $aug(T)$ of T, to each of the goal clauses. As there are no negative occurrences of $ls(x)$ in $\mathtt{SOS}$, $ls(x)$ occurs in all clauses derived from our goal clauses. In Figure 4.9, line 13 constructs the goal clauses, line 14 calls the *resolution_sos* procedure to saturate $\mathtt{SOS} \cup \mathtt{Goals}$ using a set of support strategy which described in Figure 4.13, 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.

Example 4.2
Let the set T consist of the three $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses $a \Rightarrow \mathbf{A}\bigcirc l$, $b \Rightarrow \mathbf{A}\bigcirc l$, and $a \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc a$ 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) $\neg Q_a(x) \vee Q_a(\mathsf{app}(s_{ind}, x))$

The atom ordering we use is based on the precedence $ls \succ l \succ a \succ b$ and a selection function which returns the empty set of negative literals for every clause, that is, 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 resolution inference involving 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:

```
procedure $eresolvent$(C, $\mathtt{H_i}$(x))
// C is a sometime clause $Q \Rightarrow \mathbf{A}\Diamond\neg l$ or $Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$
// $\mathtt{H}_i(x)$ = $\neg \bigwedge_{i=1}^{n} G_i(x)$ is a loop formula
begin
    if $\mathtt{H}_i$(x) = true then
       $G_i$(x) := false;
    end if
    if C is an A-sometime clause then
      resolvents := $\{\lceil\neg Q\rceil(x) \vee \neg Q_l(x) \vee G_i(x) \mid 1 \leq i \leq n\} \cup$
                    $\{\neg Q_{w^{\mathbf{A}}_{\neg l}}(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee G_i(x)\sigma \mid 1 \leq i \leq n$,
                                             $\sigma = \{x \leftarrow \mathsf{app}(s,x)\}\} \cup$
                    $\{\lceil\neg Q\rceil(x) \vee \neg Q_l(x) \vee Q_{w^{\mathbf{A}}_{\neg l}}(x)$,
                     $\neg Q_{w^{\mathbf{A}}_{\neg l}}(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee Q_{w^{\mathbf{A}}_{\neg l}}(\mathsf{app}(s,x))\}$;
    else if C is an 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 4.10: The *eresolvent* procedure

[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)	$ls(x) \vee \neg Q_b(x)$

Clauses (5) and (6) contribute to the construction of $\mathtt{H}_0(x)$ (see lines 15 and 16 of the *eres*) and we obtain $\mathtt{H}_0(x) = Q_a(x) \vee Q_b(x)$. As $\mathtt{H}_0(x)$ 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 (7) and (8) below:

[G]	(7)	$ls(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee \neg Q_a(\mathsf{app}(s,x))$
[G]	(8)	$ls(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee \neg Q_b(\mathsf{app}(s,x))$
[1, R, 7]	(9)	$ls(x) \vee \neg Q_a(x) \vee \neg Q_a(\mathsf{app}(s,x))$
[1, R, 8]	(10)	$ls(x) \vee \neg Q_a(x) \vee \neg Q_b(\mathsf{app}(s,x))$
[2, R, 7]	(11)	$ls(x) \vee \neg Q_b(x) \vee \neg Q_a(\mathsf{app}(s,x))$
[2, R, 8]	(12)	$ls(x) \vee \neg Q_b(x) \vee \neg Q_b(\mathsf{app}(s,x))$
[3, R, 9]	(13)	$ls(x) \vee \neg Q_a(x) \vee \neg Q_a(x)$
[13, C]	(14)	$ls(x) \vee \neg Q_a(x)$
[3, R, 11]	(15)	$ls(x) \vee \neg Q_b(x) \vee \neg Q_a(x)$

As the condensed clause (14) makes clause (13) redundant and clause (14) also subsumes clause (15), of all the clauses in the saturated set, only clause (14) contributes to the construction of $\mathtt{H}_1(x)$ and we obtain $\mathtt{H}_1(x) = Q_a(x)$. Again, $\mathtt{H}_1(x)$ does not satisfy any of the three termination

```
procedure resolution_prover(N)
begin
    Wo := $\emptyset$; Us := taut(sub(N));
    while (Us $\neq \emptyset$ and $\bot \notin$ Us)
        Given := choose(Us);
        Us    := Us $\setminus$ {Given};
        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 4.11: A simple resolution prover [74]

conditions, and a third iteration of the main loop of *eres* is required. There is only one goal clause, clause (16).

[G]	(16)	$ls(x) \vee \neg Q_l(\mathsf{app}(s,x)) \vee \neg Q_a(\mathsf{app}(s,x))$
[1, R, 16]	(17)	$ls(x) \vee \neg Q_a(x) \vee \neg Q_a(\mathsf{app}(s,x))$
[2, R, 16]	(18)	$ls(x) \vee \neg Q_b(x) \vee \neg Q_a(\mathsf{app}(s,x))$
[3, R, 17]	(19)	$ls(x) \vee \neg Q_a(x) \vee \neg Q_a(x)$
[19, C]	(20)	$ls(x) \vee \neg Q_a(x)$
[3, R, 18]	(21)	$ls(x) \vee \neg Q_a(x) \vee \neg Q_b(x)$

Again, the condensed clause (20) makes clause (19) redundant and clause (20) also subsumes clause (21). Only clause (20) remains to contribute to the construction of $\mathtt{H}_2(x)$. We obtain $\mathtt{H}_2(x) = Q_a(x)$ which is equivalent to $\mathtt{H}_1(x)$. Thus, the third termination condition of *eres* is satisfied (line 21) and the *eresolvent* procedure, shown in Figure 4.10, 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, $i \geq 0$,

1. $N_{i+1} = N_i \cup \{C\}$ 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 a selection function S_{FOL} corresponding to $\succ$ and S, respectively, to premises in N_i; or

2. $N_{i+1} = N_i \cup R$ where R is $eres(N_i, \Gamma)$ for some eventuality clauses Γ in T^{ev}.

A $\mathsf{R}^{\succ,S}_{\text{CTL}}$-*emulating refutation* of T by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\text{FOL}}$ is a $\mathsf{R}^{\succ,S}_{\text{CTL}}$-emulating derivation $N_0, N_1, \ldots$ from T by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\text{FOL}}$ such that for some $i \geq 0$, N_i contains the empty clause.

Theorem 4.3 *Let T be a set of SNF$^{\text{g}}_{\text{CTL}}$ clauses. Then T has a refutation by $\mathsf{R}^{\succ,S}_{\text{CTL}}$ iff there is a $\mathsf{R}^{\succ,S}_{\text{CTL}}$-emulating refutation of T by $\mathsf{R}^{\succ_{FOL},S_{FOL}}_{\text{FOL}}$.*

Proof. Let $T_0, T_1, \ldots$ be a refutation of $T = T_0$ by $\mathsf{R}^{\succ,S}_{\text{CTL}}$ 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 *eres* in Figure 4.9 is almost the same as the loop search algorithm in Section 3.4.3. The only difference is that we provide implementations for the following two tasks in the loop search algorithm in Section 3.4.3.

(1) Search in T for merged clauses of the form $X_j \Rightarrow \mathbf{A}\bigcirc Y_j$, $X_j \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc Y_j$, and $X_j \Rightarrow Y_j$ such that $Y_j \Rightarrow l$ is provable in propositional logic.

(2) Search in T for merged clauses of the form $X_j \Rightarrow Y_j$, $X_j \Rightarrow \mathbf{A}\bigcirc Y_j$ or $X_j \Rightarrow \mathbf{E}_{\langle ind \rangle}\bigcirc Y_j$ such that $Y_j \Rightarrow l$ and $Y_j \Rightarrow H_i$ are provable in propositional logic.

Thus, we only need to prove the correctness of our implementation for those two tasks. We now give a number of useful definitions.

Using the algorithm *eres*, we essentially search for clauses in `SOS` to form such merged clauses. Let T_l be the set such that a clause Γ is in T_l iff $\lceil \Gamma \rceil$ is in `SOS`.

If there exists a set Δ of SNF$^{\text{g}}_{\text{CTL}}$ clauses such that all clauses in Δ can be merged into one merged clause, then we use $m(\Delta)$ to denote this merged clause.

Let Q be a conjunction of literals. Then we use $cond(Q)$ to denote a conjunction of literals $\bigwedge_{i=1}^{n} p_i$ such that (i) for every i, p_i is in $cond(Q)$ iff p_i occurs in Q and (ii) for every $i, j, 1 \leq i < j \leq n, p_i \neq p_j$. For example, if $Q = a \wedge b \wedge a$, then $cond(Q) = a \wedge b$.

We consider the case for ERES2, i.e. all **E**-step clauses in T have the same index *ind*. Therefore, any subset Δ of T_l has a merged clause $m(\Delta)$. As to the proof for ERES1, it can be achieved in an analogous way.

For task (1), we need to prove that

(3) for every clause $G(x)$ in $\mathtt{T}_2$, there exists a set $\Delta \subseteq T_l$ such that, for its merged clause $m(\Delta) = X \Rightarrow *Y$, $Y \Rightarrow l$ is provable and $cond(X) = P$, where $\lceil \neg P \rceil(x) = G(x)$ and $*$ is either empty or a temporal operator in the set $\{\mathbf{A}\bigcirc, \mathbf{E}_{\langle ind \rangle}\bigcirc\}$; and

(4) if there exists a set $\Delta \subseteq T_l$ such that, for its merged clause $m(\Delta) = X \Rightarrow *Y$, $Y \Rightarrow l$ is provable, then there exists a clause $G(x) = \lceil \neg P \rceil(x)$ in $\mathtt{T}_2$ such that $X \Rightarrow P$, i.e. $\lceil \neg X \rceil(x)$ is equivalent to $G(x)$ or $\lceil \neg X \rceil(x)$ can be subsumed by $G(x)$.

Firstly, we consider case (3). We know that, if a clause $G(x)$ is in $\mathtt{T}_2$, then from the algorithm *eres*, $\lceil \Gamma_1 \rceil$ is in $\mathtt{T}_1$, where $\Gamma_1 = P \wedge \neg ls \Rightarrow *\mathbf{false}$ and $\lceil \neg P \rceil(x) = G(x)$. We also know that ls does not occur in `SOS`, $\lceil \Gamma_2 \rceil$ (where $\Gamma_2 = \neg ls \Rightarrow \mathbf{A}\bigcirc \neg l$) is in `Goals` and `SOS` is saturated. Therefore, $\lceil \Gamma_1 \rceil$ must be derived using $\lceil \Gamma_2 \rceil$ and clauses in `SOS` in *resolution_sos*. By Theorem 4.2, Γ_1 also can be derived using Γ_2 and T_l. We know according to our step resolution rules, **(5)** the left-hand side of the resolvent of an application of step resolution rules to two premises $P_1 \Rightarrow *D_1$ and $P_2 \Rightarrow *D_2$ is $cond(P_1 \wedge P_2)$. Let $\Delta \subseteq T$ be the set consisting of the clauses involved in the derivation to derive $\Gamma_1 = P \wedge \neg ls \Rightarrow *\mathbf{false}$ and let $m(\Delta) = X \Rightarrow *Y$. Then $cond(X) = P$. Since we know that **false** is derived, then by the soundness of our step resolution rule the set $\{Y\} \cup \{\neg l\}$ is inconsistent, i.e. $Y \wedge \neg l \Rightarrow \mathbf{false}$. Consequently $Y \Rightarrow l$.

Secondly, we prove case (4). Assume $m(\Delta)$ is $X \Rightarrow *Y$.

- Assume Y is consistent. As $Y \Rightarrow l$, the set W_r of propositional clauses is unsatisfiable, where $W_r = \{B \mid A \Rightarrow \mathbf{A}\bigcirc B \in \Delta\} \cup \{B \mid A \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc B \in \Delta\} \cup \{B \mid \mathbf{true} \Rightarrow B \in \Delta\} \cup \{\neg l \mid \neg ls \Rightarrow \mathbf{A}\bigcirc\neg l\}$. In Theorem 3.7, we have established that for an unsatisfiable set of propositional clauses like W_r, it has a refutation by propositional ordered resolution with selection using the same ordering and selection function as for $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. Furthermore, there is a derivation by the step resolution rules SRES1 to SRES3 and SRES6 to SRES8 such that $\Gamma_3 = Q \Rightarrow *\mathbf{false}$ can be derived and Q is the conjunction of literals. As Y is not unsatisfiable and $Y \Rightarrow l$, to derive Γ_3, $\neg ls \Rightarrow \mathbf{A}\bigcirc\neg l$ must be involved. Since ls does not occur in T_l, $\neg ls$ is in Q. Let $Q = \neg ls \wedge P$. Then $\Gamma_3 = P \wedge \neg ls \Rightarrow *\mathbf{false}$. By Theorem 4.2 and the algorithm *eres*, $\lceil\Gamma_3\rceil$ is in $\mathtt{T}_1$. Therefore, there exists a clause $G(x)$ in $\mathtt{T}_2$ such that $G(x) = \lceil\neg P\rceil(x)$. Let Δ' be the set consisting of the clauses involved in the derivation to derive $\Gamma_3 = P \wedge \neg ls \Rightarrow *\mathbf{false}$ and let $m(\Delta') = X' \Rightarrow *Y'$. As it is not necessary that all the clauses in Δ occurring in that derivation, we obtain that $\Delta' \subseteq \Delta$ and, consequently, $X \Rightarrow X'$. From (5), we know that $P = cond(X')$, so $X' \Rightarrow P$. Thus, $X \Rightarrow P$.

- Assume Y is inconsistent. In this case, even if the clause $G(x)$ can not be found in $\mathtt{T}_2$, the algorithm *eres* is still correct, as the possible resolvents generated for Δ can always be subsumed by some clauses in $\mathtt{SOS}$. Assume $\Delta = \{A_1 \Rightarrow *C_1, \ldots, A_n \Rightarrow *C_n\}$ and $\bigwedge_{i=1}^n C_i$ is inconsistent. We also assume that (i) loop search is applied for the $\mathbf{E}$-sometime clause $Q \Rightarrow \mathbf{E}_{\langle ind\rangle}\Diamond\neg l$; (ii) a loop can be found; and (iii) the formula H_i built in the last iteration of the loop search is $H_i = D \vee (\bigwedge_{i=1}^n A_i)$. Then the resolvents generated for Δ are the following clauses:

$$\Gamma_4 = \mathbf{true} \Rightarrow \neg Q \vee \neg l \vee \bigvee_{i=1}^n \neg A_i \qquad \Gamma_5 = w^{ind}_{\neg l} \Rightarrow \mathbf{E}_{\langle ind\rangle}\bigcirc(\neg l \vee \bigvee_{i=1}^n \neg A_i)$$

  As $\bigwedge_{i=1}^n C_i$ is inconsistent and $\mathtt{SOS}$ is saturated, there exists a clause $\lceil\Gamma_6\rceil$, where

$$\Gamma_6 = \mathbf{true} \Rightarrow \bigvee_{j=1}^m \neg A'_j,$$

  in $\mathtt{SOS}$ such that $\Delta' = \{A'_1 \Rightarrow *C'_1, \ldots, A'_m \Rightarrow *C'_m\} \subseteq \Delta$. Therefore, $\lceil\Gamma_6\rceil = \bigvee_{j=1}^m \lceil\neg A'_j\rceil(x)$ subsumes $\lceil\Gamma_4\rceil = \lceil\neg Q\rceil(x) \vee \neg Q_l(x) \vee \bigvee_{i=1}^n \lceil\neg A_i\rceil(x)$ and $\lceil\Gamma_5\rceil = \neg Q_{w^{ind}_{\neg l}}(x) \vee \neg Q_l(\mathsf{app}(s_{ind}, x)) \vee \bigvee_{i=1}^n \lceil\neg A_i\rceil(\mathsf{app}(s_{ind}, x))$.

For task (ii), the proof is analogous. By the completeness of the loop search algorithm for CTL [15], our version of the loop search algorithm is complete as well.

Finally, 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, $i \geq 0$, $N_i = \lceil T^{det}_i\rceil$. The base case, where

```
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 4.12: The main procedure of CTL-RP

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 4.2 establishes the required correspondence. In the second case, since we use essentially the same loop search algorithm, the *eresolvent* procedure in Figure 4.10 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 $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$-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. □

4.2.5 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, 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 prover [74] as shown in Figure 4.11. 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 8 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; else 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 `Us` $\cup$ `Wo`, that is, 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 `Us` $\cup$ `Wo` is not saturated as inferences between clauses in `Us` have not been computed yet. So, we would need to saturate `Us` $\cup$ `Wo` within *eres* itself, which obviously leads to repeated inferences as *resolution_prover* will continue to saturate `Us` $\cup$ `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 re-iterate the overall saturation process with these new clauses. This gives rise to the algorithm for the main procedure of CTL-RP shown in Figure 4.12. 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, respectively). As we will repeatedly saturate a set of clauses, a set of support strategy described in Figure 4.13 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 [52] (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, that is, have produced a non-empty set of resolvents. Therefore, we eliminate clauses from `New` which are subsumed by clauses in `SOS` (line 19).

The procedure *resolution_sos* is shown in Figure 4.13. 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

```
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 := choose(N);
        N := N $\setminus \{Given\}$;
        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 4.13: The *resolution_sos* procedure

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 4.11, 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 use 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.

4.2.6 CTL-RP

Currently, there are many high performance first-order theorem provers, which have been developed for years, for example, SPASS [54], Vampire [72], E [65] and Prover9 [55]. Among all of them, SPASS satisfies all our requirements, as it

- provides the implementation of *resolution_sos* and all the inference and redundancy elimination rules for first-order ordered resolution with selection;
- is well-documented and well-supported;
- is mature software, having been developed for more than a decade and over several versions; and
- can be modified for our purposes, since the source code is publicly available under the GNU General Public License.

Therefore, our resolution theorem prover for CTL, CTL-RP, is based on the first-order resolution prover SPASS 3.0 [54, 75]. To realise the calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ we have added our own implementations of

1. 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;

2. the procedures *eres* and *eresolvent* that implement the eventuality resolution rules; and

3. the procedure *main* that integrates *resolution_sos* with the implementation of eventuality resolution to finally form our prover CTL-RP.

Our prover CTL-RP is not only able to tell us whether a given CTL formula is satisfiable or not but also able to provide the corresponding proof if the given formula is unsatisfiable. For step resolution, the proof can be extracted straightforwardly, while for eventuality resolution, the premises can be traced from the formula returned by the loop search algorithm.

4.3 Related theorem provers

4.3.1 OTRES and TRP++

In 1991, a clausal resolution calculus $\mathsf{R}_{\mathrm{PLTL}}$ for PLTL was proposed by Fisher [37] (see also [26, 38]). The calculus requires PLTL formulae to be transformed into a clausal normal form, called *Separated Normal Form* (SNF). Just as our resolution calculus for CTL, Fisher's resolution calculus for PLTL consists of several step resolution rules and a single eventuality resolution rule.

In [28], Dixon proposed an approach of implementing $\mathsf{R}_{\mathrm{PLTL}}$ using a propositional/first-order resolution prover to perform both step and eventuality resolution rules. Moreover, Dixon has constructed a prototype prover OTRES based on this approach. A few years later, Hustadt and Konev have developed an efficient temporal resolution theorem prover TRP++ [48], again based on the calculus $\mathsf{R}_{\mathrm{PLTL}}$.

In [48], they also proposed an interesting idea that SNF clauses can be represented by first-order clauses and then the step resolution for PLTL can be performed by first-order resolution. However, in the implementation, TRP++ does not adopt first-order techniques, but instead opts for a "near propositional" approach. Generally speaking, they represent SNF clauses as propositional clauses and supply each literal with an attribute, which indicates whether the literal is originally from an initial clause, the left-hand side of a step clause or the right-hand side of a step clause. In this thesis, we successfully extend their ideas about using first-order techniques for PLTL resolution to CTL resolution and, consequently, we are able to reuse some existing high performance first-order prover for our CTL prover CTL-RP.

4.3.2 Tableau Workbench

Besides CTL-RP, there appears to exist only one other CTL theorem prover, namely a CTL module for the Tableau Workbench (TWB) [2][3].

[3]During the final stage of writing this thesis, we become aware of MLSolver (http://www2.tcs.ifi.lmu.de/mlsolver/), which can be used to check CTL satisfiability. However, there was insufficient time to include it in our empirical study.

	CTL equivalences	CTL-RP 00.14	TWB
1.	$\mathbf{A}\Box p \equiv \neg\mathbf{E}\Diamond\neg p$	0.007s	0.005s
2.	$\mathbf{E}\Box p \equiv \neg\mathbf{A}\Diamond\neg p$	0.007s	0.004s
3.	$\mathbf{E}\bigcirc(p \vee q) \equiv \mathbf{E}\bigcirc p \vee \mathbf{E}\bigcirc q$	0.005s	0.005s
4.	$\mathbf{A}\bigcirc p \equiv \neg\mathbf{E}\bigcirc\neg p$	0.003s	0.006s
5.	$\mathbf{E}(p\,\mathcal{U}\,q) \equiv q \vee (p \wedge \mathbf{E}\bigcirc\mathbf{E}(p\,\mathcal{U}\,q))$	0.019s	0.005s
6.	$\mathbf{A}(p\,\mathcal{U}\,q) \equiv q \vee (p \wedge \mathbf{A}\bigcirc\mathbf{A}(p\,\mathcal{U}\,q))$	0.028s	0.005s
7.	$\mathbf{E}\Diamond p \equiv \mathbf{E}(\mathbf{true}\ \mathcal{U}\,p)$	0.007s	0.008s
8.	$\mathbf{A}\Diamond p \equiv \mathbf{A}(\mathbf{true}\ \mathcal{U}\,p)$	0.007s	0.008s

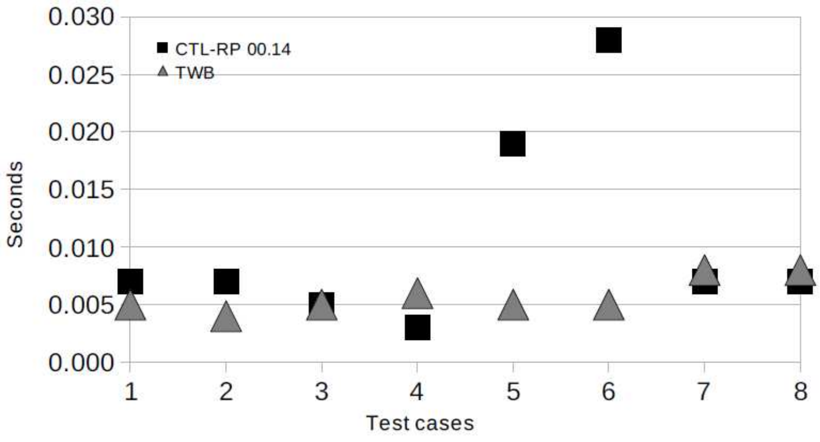
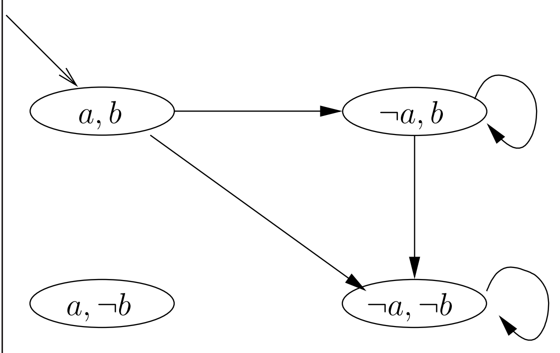


Figure 4.14: Performance of CTL-RP 00.14 and TWB 3.4 on eight ‘textbook’ CTL formulae (CTL-BF1)

The Tableau Workbench is a generic framework for building automated theorem provers for arbitrary 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 a double-EXPTIME decision procedure [3]. Therefore the complexity of this CTL decision procedure is higher than the complexity of CTL-RP, which is EXPTIME. Generally speaking, the one-pass tableau searches for a satisfying CTL model structure by exploring alternative structures and different branches in each structure using depth-first search, instead of constructing a behaviour graph like structure and reducing it using elimination rules. It should be noted that the prime aim of TWB is not efficiency.

4.4 Performance of CTL-RP

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



Representation of this state transition system in CTL

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

Figure 4.15: A state transition system

have therefore created four sets of benchmark formulae ourselves that we have used to compare CTL-RP version 00.14 with TWB version 3.4 and CTL-RP version 00.14 with version 00.09. The difference between versions 00.14 and 00.09 is that version 00.09 implements the transformation procedure used in [15] (see also Appendix A) while version 00.14 implements our new transformation procedure introduced in Section 3.3.2. The comparison was performed on a Linux PC with an Intel Core 2 Duo E6400 CPU@2.13 GHz and 3GB main memory, using the Fedora 9 operating system.

4.4.1 CTL-RP vs. TWB

The first set of benchmark formulae, CTL-BF1, consists of eight well-known equivalences between temporal formulae taken from [31]. The CTL equivalences themselves and the CPU time required by TWB and CTL-RP to prove each of them is shown in Figure 4.14. Both systems easily prove each of the equivalences in less then 0.03 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 4.15. 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 4.16 shows the five properties, their validity status with respect to the finite state transition system, and the CPU time in seconds 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 generated 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

$$\mathbf{A}\Box(s \Rightarrow \mathbf{A}\bigcirc(\textstyle\bigvee_{i=1}^{n} s_i)) \quad \text{or} \quad \mathbf{A}\Box(s \Rightarrow \mathbf{E}\bigcirc(\textstyle\bigvee_{i=1}^{n} s_i)),$$

	Property	Status	CTL-RP 00.14	TWB
1.	$\mathbf{A}\bigcirc(\neg\mathbf{E}\Diamond(a \wedge b))$	Valid	0.04s	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.	$\mathbf{A}\Box(\mathbf{E}\bigcirc\neg b)$	Valid	0.03s	46.95s
5.	$\mathbf{E}(b\,\mathcal{U}\,\neg b)$	Valid	0.01s	2.95s

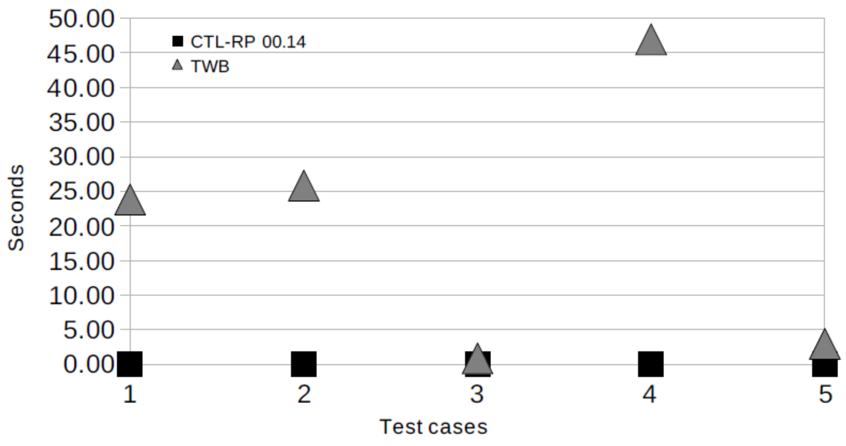


Figure 4.16: Performance of CTL-RP 00.14 and TWB 3.4 on finite state transition system (CTL-BF2)

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 *property specification* be a CTL formula of the form

- $*(\bigvee_{i=1}^{n} s_i)$, where $*$ is a randomly chosen element of $\{\mathbf{A}\bigcirc, \mathbf{E}\bigcirc, \mathbf{A}\Box, \mathbf{E}\Box, \mathbf{A}\Diamond, \mathbf{E}\Diamond\}$; or
- $(\bigvee_{i=1}^{n} s_i) * (\bigvee_{j=1}^{m} s_j)$, where $*$ is a randomly chosen element of $\{\mathbf{A}\mathcal{U}, \mathbf{E}\mathcal{U}\}$, n and m are two randomly generated numbers between 1 and 2, and each s_i, $1 \leq i \leq n$, and $s_j, 1 \leq j \leq m$, 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. All one hundred formulae are unsatisfiable. Figure 4.17 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 the 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 0.25 seconds. In contrast, the performance of TWB is much more variable, with a high percentage of formulae not being solved.

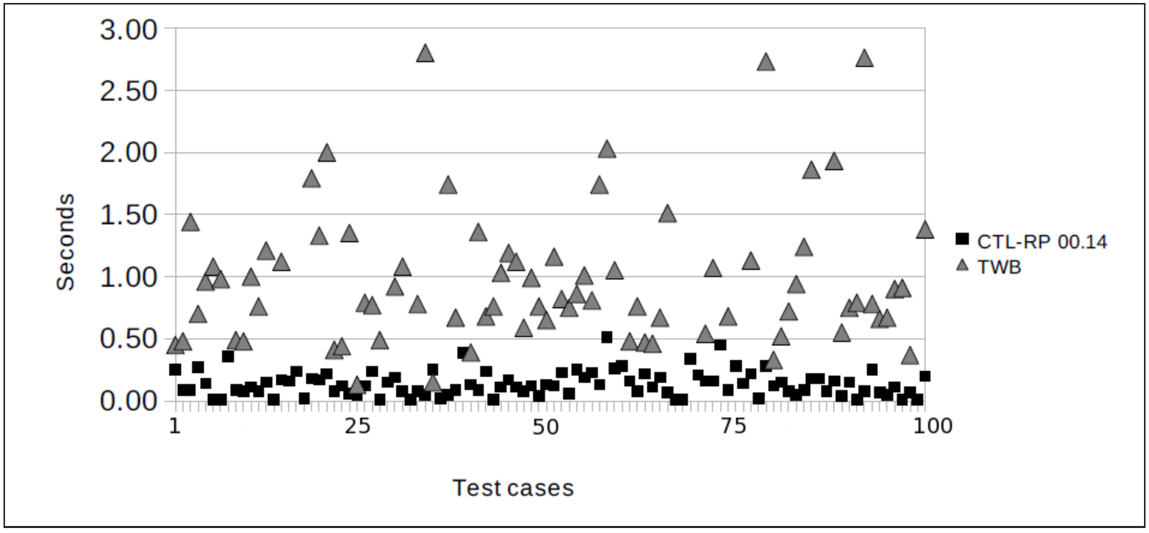


Figure 4.17: Performance of CTL-RP 00.14 and TWB 3.4 on the third set of benchmark formulae (CTL-BF3)

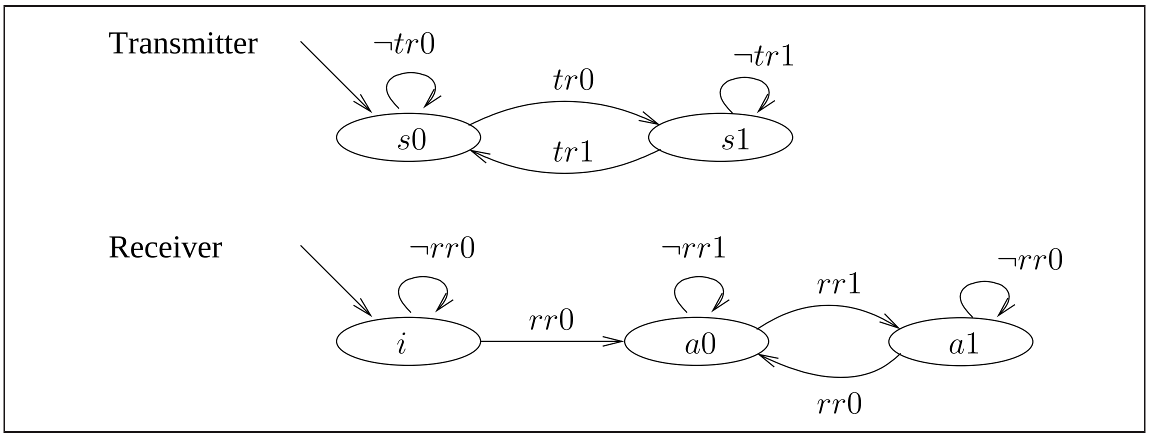


Figure 4.18: Transmitter and Receiver in the Alternating Bit Protocol

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) [49] 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, i.e. the channel may lose messages but not infinitely many messages. 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 transition systems as the ones shown in Figure 4.18. 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 $s0$, that is, follow the transition labelled $\neg$*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, that is, 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 $a0$, that is, 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

$$s0 \land \neg tr0 \land \neg tr1$$

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

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

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

$$\begin{aligned}
\mathbf{A}\Box(s0 \lor s1)\\
\mathbf{A}\Box(s0 \Rightarrow \neg s1)\\
\mathbf{A}\Box(s1 \Rightarrow \neg s0)
\end{aligned}$$

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

$$i \land \neg rr0 \land \neg rr1$$

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

$$\begin{aligned}
\mathbf{A}\Box(i \land \neg rr0 \Rightarrow \mathbf{A}\bigcirc i)\\
\mathbf{A}\Box(i \land rr0 \Rightarrow \mathbf{A}\bigcirc a0)\\
\mathbf{A}\Box(a0 \land \neg rr1 \Rightarrow \mathbf{A}\bigcirc a0)\\
\mathbf{A}\Box(a0 \land rr1 \Rightarrow \mathbf{A}\bigcirc a1)\\
\mathbf{A}\Box(a1 \land \neg rr0 \Rightarrow \mathbf{A}\bigcirc a1)\\
\mathbf{A}\Box(a1 \land rr0 \Rightarrow \mathbf{A}\bigcirc a0)
\end{aligned}$$

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

$$\begin{aligned}&\mathbf{A}\Box(i \vee a0 \vee a1)\\&\mathbf{A}\Box(i \Rightarrow \neg a0 \wedge \neg a1)\\&\mathbf{A}\Box(a0 \Rightarrow \neg i \wedge \neg a1)\\&\mathbf{A}\Box(a1 \Rightarrow \neg i \wedge \neg a0)\end{aligned}$$

In addition, we specify that the Transmitter and the Receiver will always eventually be successful in transmitting their messages, as the channel does not lose infinitely many messages.

$$\begin{aligned}&\mathbf{A}\Box(s0 \Rightarrow \mathbf{A}\Diamond rr0)\\&\mathbf{A}\Box(s1 \Rightarrow \mathbf{A}\Diamond rr1)\\&\mathbf{A}\Box(a0 \Rightarrow \mathbf{A}\Diamond tr0)\\&\mathbf{A}\Box(a1 \Rightarrow \mathbf{A}\Diamond tr1)\end{aligned}$$

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 the CPU time.

Property	CTL-RP 00.14	TWB
1	0.78s	-
2	53.00s	-
3	94.88s	-

4.4.2 CTL-RP 00.14 vs. 00.09

The CPU time required by CTL-RP versions 00.14 and 00.09 to prove each formula in benchmarks CTL-BF1 and CTL-BF2 are shown in Figure 4.19 and 4.20, respectively. Although version 00.14 outperformed 00.09 for every formula, the CPU time they consumed is very close. However for the formulae in the set CTL-BF3 and CTL-BF4, from Figure 4.21 and the table shown below, we can see that the performance of version 00.14 is noticeably better than the performance of version 00.09. More precisely, version 00.14 on average only required one third of the CPU time that version 00.09 needed.

	CTL equivalences	CTL-RP 00.14	CTL-RP 00.09
1.	$\mathbf{A}\Box p \equiv \neg\mathbf{E}\Diamond\neg p$	0.007s	0.008s
2.	$\mathbf{E}\Box p \equiv \neg\mathbf{A}\Diamond\neg p$	0.007s	0.008s
3.	$\mathbf{E}\bigcirc(p \vee q) \equiv \mathbf{E}\bigcirc p \vee \mathbf{E}\bigcirc q$	0.005s	0.005s
4.	$\mathbf{A}\bigcirc p \equiv \neg\mathbf{E}\bigcirc\neg p$	0.003s	0.004s
5.	$\mathbf{E}(p\,\mathcal{U}\,q) \equiv q \vee (p \wedge \mathbf{E}\bigcirc\mathbf{E}(p\,\mathcal{U}\,q))$	0.019s	0.049s
6.	$\mathbf{A}(p\,\mathcal{U}\,q) \equiv q \vee (p \wedge \mathbf{A}\bigcirc\mathbf{A}(p\,\mathcal{U}\,q))$	0.028s	0.068s
7.	$\mathbf{E}\Diamond p \equiv \mathbf{E}(\mathbf{true}\ \mathcal{U}\,p)$	0.007s	0.010s
8.	$\mathbf{A}\Diamond p \equiv \mathbf{A}(\mathbf{true}\ \mathcal{U}\,p)$	0.007s	0.010s

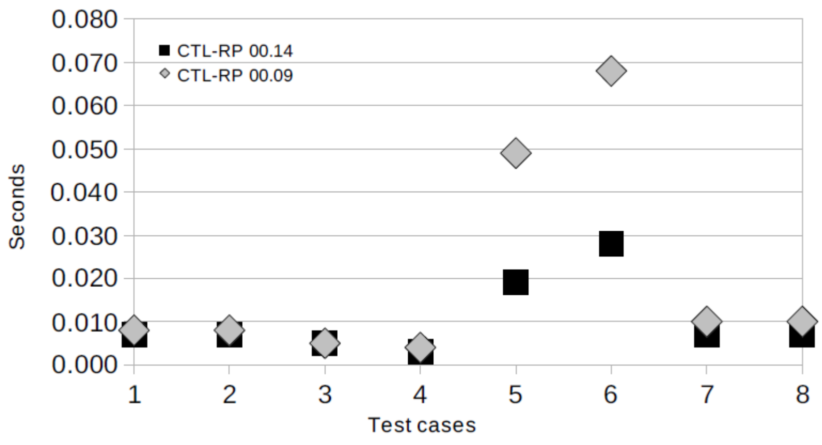


Figure 4.19: Performance of CTL-RP 00.09 and 00.14 on eight ‘textbook’ CTL formulae (CTL-BF1)

Property	CTL-RP 00.14	CTL-RP 00.09
1	0.78s	1.39s
2	53.00s	192.86s
3	94.88s	326.02s

Our experiments show that for the formulae we designed or generated randomly in our benchmarks, in general version 00.14 is better than version 00.09 and as the formulae become more complex, the difference in the performance between these two versions becomes bigger. This indicates

1. the way that the transformation to $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ is performed has a significant impact on the performance of our CTL prover, and

2. our transformation procedure is strictly more efficient than the one used in [15] for benchmark formulae in CTL-BF1 to CTL-BF4.

	Property	Status	CTL-RP 00.14	CTL-RP 00.09
1.	$\mathbf{A}\bigcirc(\neg\mathbf{E}\Diamond(a \wedge b))$	Valid	0.04s	0.06s
2.	$\mathbf{A}\Box(\mathbf{A}(a\,\mathcal{U}\,\neg a))$	Valid	0.02s	0.02s
3.	$\mathbf{A}\Box(a \vee b)$	Not Valid	0.01s	0.01s
4.	$\mathbf{A}\Box(\mathbf{E}\bigcirc\neg b)$	Valid	0.03s	0.03s
5.	$\mathbf{E}(b\,\mathcal{U}\,\neg b)$	Valid	0.01s	0.01s

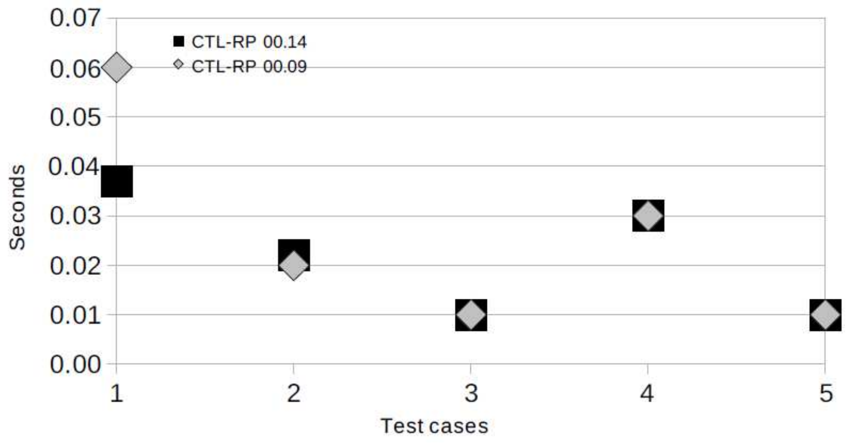


Figure 4.20: Performance of CTL-RP 00.09 and 00.14 on finite state transition system properties (CTL-BF2)

4.5 Conclusions

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 Computation Tree Logic. One explanation is the considerable effort that is required to implement a reasonably efficient theorem prover for these logics. The TWB tries to tackle this problem by providing a generic framework for building tableau-based theorem provers. We are taking a different approach by first developing a resolution calculus for the non-classical logic we are interested in and then building a bridge to first-order resolution which allows us to re-use existing first-order theorem provers. In this thesis, we construct a bridge from CTL to first-order logic, i.e. use a first-order prover to conduct some CTL resolution inferences.

In Chapter 3 we have defined a new normal form $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ for CTL and provided an improved clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL. Moreover, we have presented our refined resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, which consists of eight step resolution rules and two eventuality resolution rules. Some of the choices we made when designing our calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ were already motivated by our intention to use this particular approach, namely bridging first-order resolution to CTL resolution, to realise $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$. Therefore, in this chapter, we have provided a new technique to implement step resolution rules via first-order ordered resolution with selection and have described an algorithm for the eventuality resolution rules of our calculus. In addition, we employ an efficient automated resolution

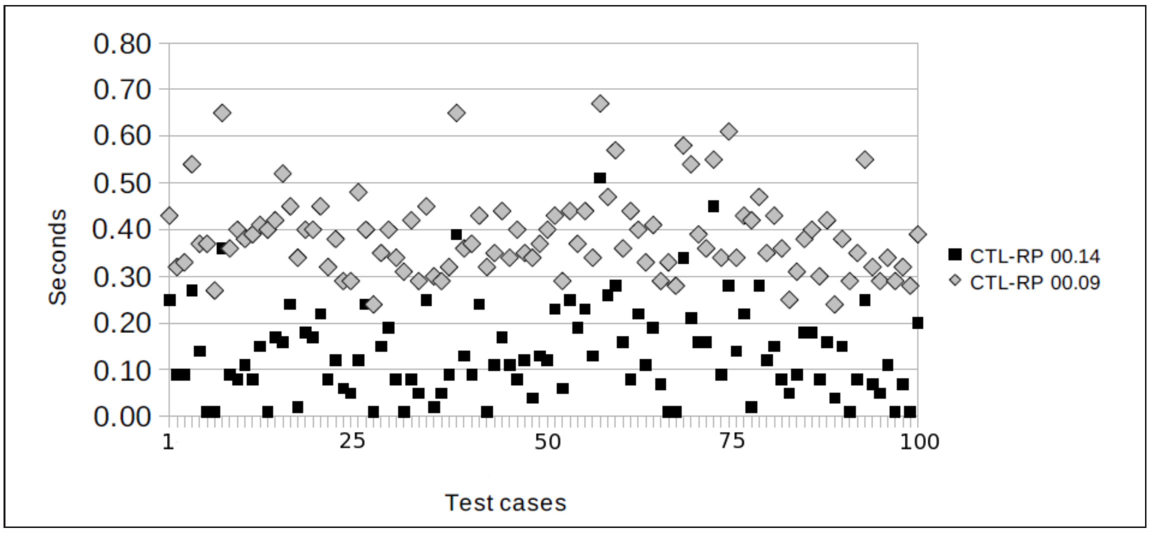


Figure 4.21: Performance of CTL-RP 00.09 and 00.14 on the third set of benchmark formulae (CTL-BF3)

theorem prover for first-order logic, SPASS, to implement our CTL theorem prover CTL-RP. In our comparison between CTL-RP and TWB, we observe that CTL-RP is able to efficiently verify a wide range of problems formalisable in CTL. From the comparison between CTL-RP versions 00.14 and 00.09, we observe that the transformation procedure plays an important role for clausal resolution for CTL and our new transformation procedure is more efficient on the formulae we used in our experiments.

Chapter 5

Resolution for the Next-time fragment of ATL

5.1 Introduction

Alternating-time Temporal Logic (ATL) was first introduced in [5] in 1997 by Alur, Henzinger and Kupferman and then further developed in [6] and [7]. ATL can be thought of as a generalisation and extension of CTL, in which the temporal operators are parameterised by sets of agents. The selective quantification over paths in ATL enables ATL to explicitly express coalition abilities. In this chapter, we focus on the Next-time fragment of ATL (XATL) [41], which excludes the temporal operators $\Box, \Diamond$ and $\mathcal{U}$ from ATL. Generally speaking, the major difference of the expressive power between XATL and ATL is the following. XATL can only express coalition abilities in the short run, for example, the agent 1 and the agent 2 can cooperate to ensure that p holds at the next moment of time. On the other hand, ATL can express everything XATL is able to express and, furthermore, it also can deal with coalition abilities in the long run, for example, the group of agents $\{1, 2, 3\}$ can cooperate to force that p will eventually hold at sometime in the future. In addition, XATL is closely related to another useful modal logic, namely Coalition Logic[1] (CL) [57, 44].

A multi-agent systems [76] is a system consisting of multiple interacting agents, which are able to control their own behaviour and usually have their own goals and motivations. Therefore, multi-agent systems are often utilised to represent many important systems, for example, (i) concurrent and distributed systems by using agents to represent different autonomous components in the systems, (ii) social processes (e.g. auction, election and shopping) by letting agents to represent participants in the processes and so on. XATL is a logic that is able to formally specify the specification or properties of multi-agent systems. Moreover, reasoning of multi-agent systems can also be achieved by theorem proving techniques for XATL.

Consider the following example. A teacher of five students wants that the majority opinion among the students will determine the place where they go for a trip. Assume there are only two options, namely the museum and the zoo. We use 1 to 5 to represent the five students, respectively.

[1] In [39] Goranko shows that Coalition Logic can be embedded into ATL by translating formula $[C]\varphi$ of Coalition Logic into an ATL formula $\langle\langle C\rangle\rangle\bigcirc\varphi$ and the resulting fragment of ATL is XATL.

The proposition m denotes that they go to the museum whereas the proposition z denotes that they go to the zoo. Then XATL can express the collective decision making abilities. For instance,

1. $\langle\langle 1, 2, 3\rangle\rangle \bigcirc m$ states that the students $\{1, 2, 3\}$ can cooperate together to ensure that they will go to the museum;

2. $\neg\langle\langle 1, 2\rangle\rangle \bigcirc m \wedge \neg\langle\langle 1, 2\rangle\rangle \bigcirc z$ states that the students $\{1, 2\}$ have no ability to decide where they will go; and

3. $\langle\langle \emptyset\rangle\rangle \bigcirc(\neg(m \wedge z))$ states that in any situation they will not go to both the museum and the zoo.

(The formal syntax and semantics of XATL will be defined in the next section.) From the example above, we can see that XATL is able to describe the problem precisely. Furthermore, there are many applications of multi-agent systems, for example verifying properties of voting procedures [57], reasoning about various strategic games [58], and designing social procedures [73], that can be achieved by using XATL or ATL. With the increasing significance of multi-agent systems, the significance of ATL and XATL will also increase.

In this chapter, we introduce the first resolution-based calculus $\mathsf{R}_{\mathrm{XATL}}$ for tackling the XATL satisfiability problem. In Section 5.2 we provide the syntax and semantics of XATL. Since the calculus $\mathsf{R}_{\mathrm{XATL}}$ is clausal, in Section 5.3 we define a clausal normal form for XATL, called Separated Normal Form for XATL (denoted by $\mathrm{SNF}_{\mathrm{XATL}}$), and then provide a transformation procedure which applies to any XATL formula and returns a satisfiability equivalent set of XATL formulae in normal form. In Section 5.4 we present our calculus $\mathsf{R}_{\mathrm{XATL}}$, which contains seven step resolution rules to deal with constraints on the next moment, according to coalition abilities of agents. Section 5.5 gives (i) a proof that our transformation rules preserve satisfiability, (ii) a proof that our transformation procedure terminates and can computed in polynomial time in the size of the input formula, (iii) a proof of the soundness of $\mathsf{R}_{\mathrm{XATL}}$, and (iv) an outline of the completeness proof for $\mathsf{R}_{\mathrm{XATL}}$. Finally, we draw our conclusions in Section 5.6.

5.2 Syntax and semantics of XATL

XATL is a multimodal logic with CTL-style modalities indexed by subsets, commonly called *coalitions*, of a finite, non-empty set of names of agents, or players, belonging to the language. The syntax and semantics of XATL presented in this chapter is a fragment of that introduced by Goranko et al in [41] and Alur et al in [7].

5.2.1 Syntax of XATL

The formulae of XATL are defined with respect to a finite, non-empty set Σ of names of agents. The elements of Σ are commonly denoted by the natural numbers from 1 to $|\Sigma|$ (the cardinality of Σ). Subsets of Σ are called *coalitions*.

The language of XATL is based on

- a set of atomic propositions P_{PL};

- propositional constants **true** and **false**;
- boolean operators $\wedge, \vee, \Rightarrow$, and $\neg$ ($\wedge$ and $\vee$ are associative and commutative);
- a temporal operator $\bigcirc$ (at the next moment in time); and
- a coalition quantifier $\langle\langle A\rangle\rangle$ with $A \subseteq \Sigma$ and a coalition quantifier $[\![A]\!]$ with $A \subseteq \Sigma$.

The set of *(well-formed) formulae of XATL* is inductively defined as follows:

1. **true** and **false** are XATL formulae;
2. all atomic propositions in P_{PL} are XATL formulae; and
3. if φ and ψ are XATL formulae, then so are $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$, $\langle\langle A\rangle\rangle\bigcirc\varphi$, $[\![A]\!]\bigcirc\varphi$.

5.2.2 Semantics of XATL

Formulae of XATL over P_{PL} are interpreted in *concurrent game models (CGM).*

Definition 5.1 *Concurrent game model [41]*
A *concurrent game model* is a tuple $(\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$, where

- Σ is a finite, non-empty set of agents;
- S is a finite, non-empty set of states;
- d is a function assigning to every agent $a \in \Sigma$ and every state $s \in S$ a natural number $d_a(s) \geq 1$ of *moves* available to agent a at state s; these moves are identified with the numbers from 1 to $d_a(s)$. For every state $s \in S$, a *move vector* is a k-tuple $\langle\sigma_1, \sigma_2, \ldots, \sigma_k\rangle$, where $k = |\Sigma|$ and $1 \leq \sigma_a \leq d_a(s)$ for every agent $a, 1 \leq a \leq k$. Therefore, σ_a denotes an arbitrary move of agent $a \in \Sigma$. Given a state $s \in S$, $D_a(s)$ denotes the set $\{1, \ldots, d_a(s)\}$ of all moves available to agent a at state s; $D(s)$ denotes the set $\Pi_{a\in\Sigma} D_a(s)$ of all move vectors at state s; and σ denotes an arbitrary member of $D(s)$.
- δ is a *transition function* assigning to every state $s \in S$ and move vector $\sigma \in D(s)$ a state $\delta(s, \sigma) \in S$ that results from the state s if every agent $a \in \Sigma$ plays move σ_a.
- P_{PL} is a set of atomic propositions;
- $L : S \rightarrow 2^{\mathsf{P}_{\mathsf{PL}}}$ is an *interpretation function* mapping each state $s \in S$ to the set of atomic propositions which are true at state s.

Next, we give a number of auxiliary definitions which are used in either semantics of XATL or many proofs in the remainder of this chapter.

Definition 5.2 *Successor and predecessor*
For two states $s, s' \in S$, the state s' is a *successor* of the state s and the state s is a *predecessor* of the state s' iff for some $\sigma \in D(s), s' = \delta(s, \sigma)$.

Definition 5.3 *Reachable*
For two states $s, s' \in S$, the state s' is *reachable* from the state s iff

1. s and s' are the same state;
2. s' is a successor of s; or
3. s' is a successor of a state which is reachable from s.

The reachable relation is, in short, the reflexive and transitive closure of the successor relation.

Definition 5.4 *Edge in a CGM*
Let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM, s and s' be states in S, and σ be a move vector in $D(s)$. Then the tuple (s, σ, s') is an *edge in M* iff $\delta(s, \sigma) = s'$.

Definition 5.5 *Run*
A *run* in a CGM is an infinite sequence $\lambda = s_0, s_1, s_2, \ldots$ of elements of S such that, for all $i \geq 0$, the state s_{i+1} is a successor of the state s_i. Elements of the domain of λ are called *positions*. For a run λ and positions $0 \leq i$ and $0 \leq j \leq i$, we use $\lambda[i]$ and $\lambda[j, i]$ to denote the ith state of λ and the finite subsequence $s_j, s_{j+1}, \ldots, s_i$ of λ, respectively. We use $\lambda[i, \infty]$ to denote the infinite sequence $s_i, s_{i+1}, s_{i+2}, \ldots$. A run with $\lambda[0] = s$ is referred to as an *s-run*.

Given a tuple τ, we interchangeably use τ_i and $\tau(i)$ to refer to the ith element of τ. We use the symbol $*$ as a placeholder for an arbitrary move of a given agent and the symbol $-n$, where $n \in \mathbb{N}$, as a placeholder for some move of a given agent. Intuitively, $*$ acts like an universally quantified variable while $-n$ acts like an existentially quantified variable.

We now define the notions of A-moves and negative-A-moves as well as an operator $\oplus$ on A-moves and negative-A-moves.

Definition 5.6 *A-move [41]*
Let s be a state in S and let $A, A \subseteq \Sigma$, be a coalition of agents, where $|\Sigma| = k$. An *A-move* σ_A (or for brevity, a move) at state s is a k-tuple σ_A such that $\sigma_A(a) \in D_a(s)$ for every $a \in A$ and $\sigma_A(a') = *$ for every $a' \notin A$. $D_A(s)$ denotes the set of all A-moves at state s.

Note that by Definition 5.6, for a state $s \in S$, $D_\emptyset(s) = \{\sigma_\emptyset\}$, where $\sigma_\emptyset$ is the unique k-tuple $\langle *, *, \ldots, * \rangle$. Every Σ-move σ_Σ is also a move vector. For brevity, we usually write σ instead of σ_Σ.

Example 5.1
Let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM. Assume that $\Sigma = \{1, 2, 3\}$, s is a state in S and for every agent $a \in \Sigma$, $D_a(s) = 3$. Then

- $\langle 1, *, * \rangle$, $\langle 2, *, * \rangle$ and $\langle 3, *, * \rangle$ are valid $\{1\}$-moves at state s; and
- $\langle 1, *, 2 \rangle$ and $\langle 2, *, 3 \rangle$ are valid $\{1, 3\}$-moves at state s.

Note that if a k-tuple τ with $k = |\Sigma|$ and $\tau(i) \in \{*\} \cup \mathbb{N}$ for every $i, 1 \leq i \leq k$, is an A-move for some coalition of agents $A \subseteq \Sigma$, then there is no coalition of agents $A' \subseteq \Sigma$ with $A' \neq A$ such that τ is also an A'-move. That is, τ uniquely determines the coalition A of which τ is an A-move. Consequently, we will often not explicitly state the coalition A for a given k-tuple τ.

Definition 5.7 *Negative-A-move*
Let i be a natural number, s be a state in S, and $A, A \subset \Sigma$, be a coalition of agents, where $|\Sigma| = k$. Then a negative-A-move σ_A^N (or for brevity, a negative-move) at state s is a k-tuple such that

- for every $a \in A$, $\sigma_A^N(a) = *$,
- for every $a' \notin A, \sigma_A^N(a') \in D_{a'}(s)$ or $\sigma_A^N(a') = -i$, and
- there exists at least one agent $a'' \notin A$ such that $\sigma_A^N(a'') = -i$.

Example 5.2
Let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM. Assume that $\Sigma = \{1, 2, 3\}$, s is a state in S and for every agent $a \in \Sigma$, $D_a(s) = 3$. Then

- $\langle *, 1, -10 \rangle$ and $\langle *, -5, -5 \rangle$ are valid negative-$\{1\}$-moves at state s; and
- $\langle *, -1, * \rangle$ and $\langle *, -5, * \rangle$ are valid negative-$\{1, 3\}$-moves at state s.

As for A-moves, if a k-tuple τ with $k = |\Sigma|$ and $\tau(i) \in \{*\} \cup \mathbb{Z}$ for every $i, 1 \leq i \leq k$, is a negative-A-move for some coalition of agents $A \subset \Sigma$, then there is no coalition of agents $A' \subset \Sigma$ with $A' \neq A$ such that τ is also a negative-A-move. There is also no coalition of agents $A' \subseteq \Sigma$ such that τ is an A-move. Finally, no A-move σ_A is a negative-A'-move for some $A' \subset \Sigma$.

Definition 5.8 *The operation $\oplus$*
The operation $\oplus$ on A-moves and negative-A-moves is defined as follows. Let $A, A' \subseteq \Sigma$ be coalitions of agents.

1. If σ_A is an A-move and $\sigma_{A'}$ is an A'-move such that $A \cap A' = \emptyset$, then $\sigma_A \oplus \sigma_{A'}$ is an A''-move $\sigma_{A''}$ such that $A'' = A \cup A'$; for every $a \in A, \sigma_{A''}(a) = \sigma_A(a)$; for every $a' \in A'$, $\sigma_{A''}(a') = \sigma_{A'}(a')$; and for every $a'' \notin A'', \sigma_{A''}(a'') = *$.
2. If σ_A is an A-move and $\sigma_{A'}^N$ is a negative-A'-move such that $A \subseteq A'$, then $\sigma_A \oplus \sigma_{A'}^N$ is a negative-A''-move $\sigma_{A''}^N$ such that $A'' = A' \setminus A$ and $\sigma_{A''}^N$ is identical to $\sigma_{A'}^N$ except that for every $a \in A, \sigma_{A''}^N(a) = \sigma_A(a)$. In addition, we define $\sigma_{A'}^N \oplus \sigma_A$ as $\sigma_A \oplus \sigma_{A'}^N$.

Note that the operation $\oplus$ is associative, commutative and has $\sigma_\emptyset$ as the identity element, i.e. $\sigma_A \oplus \sigma_\emptyset = \sigma_A$ and $\sigma_{A'}^N \oplus \sigma_\emptyset = \sigma_{A'}^N$ for any A-move σ_A and any negative A'-move $\sigma_{A'}^N$.

Before we proceed, we give some examples demonstrating how the operation $\oplus$ works on moves and negative-moves.

Example 5.3
Let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM and assume $\Sigma = \{1, 2, 3\}$. Then
$\langle 1, *, * \rangle \oplus \langle *, 1, * \rangle = \langle 1, 1, * \rangle$

$\langle 1, *, * \rangle \oplus \langle *, 2, * \rangle \oplus \langle *, *, 3 \rangle = \langle 1, 2, * \rangle \oplus \langle *, *, 3 \rangle = \langle 1, 2, 3 \rangle$

$\langle 1, *, * \rangle \oplus \langle *, *, -3 \rangle = \langle 1, *, -3 \rangle$

$\langle -5, -5, * \rangle \oplus \langle *, *, 2 \rangle = \langle -5, -5, 2 \rangle$ and

$\langle *, *, * \rangle \oplus \langle *, *, -1 \rangle \oplus \langle 1, 2, * \rangle = \langle *, *, -1 \rangle \oplus \langle 1, 2, * \rangle = \langle 1, 2, -1 \rangle$.

Definition 5.9 *Submove*
Let $A \subseteq \Sigma$ be coalitions of agents. Let σ_A be an A-move, σ^N_A be a negative-A-move, and $\sigma^N_{A'}$ be a negative-A'-move. Then (i) σ_A is a submove of $\sigma_{A'}$ ($\sigma_A \sqsubseteq \sigma_{A'}$) iff $A \subseteq A'$ and for every $a \in A$, $\sigma_A(a) = \sigma_{A'}(a)$, (ii) σ^N_A is a submove of $\sigma^N_{A'}(\sigma^N_A \sqsubseteq \sigma^N_{A'})$ iff $A \supseteq A'$, for every $a \in \Sigma \setminus A, \sigma^N_A(a) = \sigma^N_{A'}(a)$ and for every $a' \in A \setminus A', \sigma^N_{A'}(a) \in D_a(s)$; and (iii) σ_A is a submove of $\sigma^N_{A'}(\sigma_A \sqsubseteq \sigma^N_{A'})$ iff $A \cap A' = \emptyset$ and for every $a \in A, \sigma_A(a) = \sigma^N_{A'}(a)$.

We give a few examples illustrating the notion of a submove.

Example 5.4
Let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM. Assume that $\Sigma = \{1, 2, 3\}$, s is a state in S and for every agent $a \in \Sigma$, $D_a(s) = 3$. Then

- $\sigma_A = \langle *, *, 1\rangle$ is a submove of $\sigma_{A'} = \langle *, 2, 1\rangle$;
- $\sigma^N_A = \langle *, *, -5\rangle$ is a submove of $\sigma^N_{A'} = \langle *, 1, -5\rangle$; and
- $\sigma_A = \langle 1, *, *\rangle$ is a submove of $\sigma^N_{A'} = \langle 1, *, -5\rangle$.

Moreover, the following relations are also true: $\langle *, *, *\rangle \sqsubseteq \langle 1, *, *\rangle \sqsubseteq \langle 1, 1, *\rangle \sqsubseteq \langle 1, 1, 1\rangle$.

Note that by Definition 5.9, $\sigma_\emptyset$ is a submove of any A-move σ_A or negative-A-move σ^N_A; any A-move is a submove of itself, $\sigma_A \sqsubseteq \sigma_A$; and any negative-$A$-move is also a submove of itself, $\sigma^N_A \sqsubseteq \sigma^N_A$.

Definition 5.10 *Outcome of an A-Move [41]*
Let $\sigma_A \in D_A(s)$. The *outcome* of σ_A at state s, denoted by $out(s, \sigma_A)$, is the set of all states s' for which there exists a move vector $\sigma \in D(s)$ such that $\sigma_A \sqsubseteq \sigma$ and $\delta(s, \sigma) = s'$.

Note that $\sigma_\emptyset \sqsubseteq \sigma$ for every move vector $\sigma \in D(s)$ and $out(s, \sigma_\emptyset)$ is the set consisting of all successors of s, i.e. $out(s, \sigma_\emptyset) = \{s' \mid s' = \delta(s, \sigma_\emptyset), \sigma \in D(s)\}$.

The satisfaction relation $\models$ between a pair consisting of a CGM M and a state $s \in S$, and an XATL formula is inductively defined as follows.

$$
\begin{array}{ll}
\langle M, s\rangle \models \mathbf{true} & \\
\langle M, s\rangle \not\models \mathbf{false} & \\
\langle M, s\rangle \models p & \text{iff } p \in L(s) \text{ for an atomic proposition } p \in \mathsf{P}_{\mathsf{PL}} \\
\langle M, s\rangle \models \neg\varphi & \text{iff } \langle M, s_i\rangle \not\models \varphi \\
\langle M, s\rangle \models (\varphi \vee \psi) & \text{iff } \langle M, s_i\rangle \models \varphi \text{ or } \langle M, s_i\rangle \models \psi \\
\langle M, s\rangle \models (\varphi \Rightarrow \psi) & \text{iff } \langle M, s_i\rangle \models \neg\varphi \text{ or } \langle M, s_i\rangle \models \psi \\
\langle M, s\rangle \models \langle\!\langle A\rangle\!\rangle\bigcirc\psi & \text{iff there exists an } A\text{-move } \sigma_A \in D_A(s) \text{ such that } \langle M, s'\rangle \models \psi \\
& \quad \text{for all } s' \in out(s, \sigma_A)
\end{array}
$$

Instead of $\langle M, s\rangle \models \varphi$, we also write $M, s \models \varphi$ and if $M, s \models \varphi$, we say that φ holds at state s in M.

We use the following equivalence to define the dual operator $[\![A]\!]$ of $\langle\!\langle A\rangle\!\rangle$.

$$[\![A]\!]\bigcirc\varphi \equiv \neg\langle\!\langle A\rangle\!\rangle\bigcirc\neg\varphi$$

In [73], various types of satisfiability for full ATL have been defined. Here we adopt them to XATL. Based on the satisfaction relation $\models$, we can define the notion of tight satisfiability as follows.

Definition 5.11 *Tight satisfiability*
An XATL formula φ is *tightly satisfiable* iff for some CGM $M = (\Sigma_\varphi, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ and some state $s \in S$, $\langle M, s\rangle \models \varphi$, and *tightly unsatisfiable* otherwise. φ is *valid*, written $\models \varphi$, iff for every CGM $M = (\Sigma_\varphi, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ and every state $s \in S$, $\langle M, s\rangle \models \varphi$. A CGM $M = (\Sigma_\varphi, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ such that φ is true at some state $s \in S$ is called a *model* for φ.

Besides tight satisfiability, two further variants of satisfiability can be defined for XATL.

Definition 5.12 Σ-*satisfiability*
Given an XATL formula φ and a set $\Sigma \supseteq \Sigma_\varphi$, φ is Σ-*satisfiable* if φ is satisfiable in a CGM $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$; φ is Σ-*valid* if φ is true in every such CGM.

Definition 5.13 *General satisfiability*
Given an XATL formula φ, φ is *generally satisfiable*, if there exists a set $\Sigma' \supseteq \Sigma_\varphi$ such that φ is satisfiable in a CGM $M = (\Sigma', S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$; φ is *generally valid* if φ is true in every such CGM.

In this thesis, we focus on the tight satisfiability problem. An XATL formula φ is Σ-satisfiable or generally satisfiable iff there exists a CGM $M = (\Sigma', S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$, where $\Sigma' = \Sigma_\varphi \cup \{a\}$ and $a \notin \Sigma_\varphi$, and a state $s \in S$ such that $M, s \models \varphi$ [73]. Therefore, φ is Σ-satisfiable or generally satisfiable iff $\varphi \wedge \langle\!\langle\{a\}\rangle\!\rangle\bigcirc\mathbf{true}$, where $a \notin \Sigma_\varphi$, is tightly satisfiable. Thus, a decision procedure for tight satisfiability also provides us with a decision procedure for the other two variants of satisfiability in XATL. The (tight) satisfiability problem of XATL is known to be PSPACE-complete [57, 58].

In the remainder of the chapter, we usually use the term "satisfiability" instead of "tight satisfiability" if it is clear in its context and, further, use Σ to denote the set of agents mentioned in the input formulae φ, instead of Σ_φ, unless otherwise specified. For every XATL-formula φ, we denote the set of agents mentioned in φ by Σ_φ. Finally, when we say a set of XATL formulae T is satisfiable in a CGM $M = (\Sigma_\varphi, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$, we mean that there exists a state $s \in S$ such that every formula in T holds at s.

5.3 Normal form

5.3.1 Normal form for XATL $\mathrm{SNF}_{\mathrm{XATL}}$

Our calculus $\mathsf{R}_{\mathrm{XATL}}$ operates on formulae in a clausal normal form called Separated Normal Form for XATL, denoted by $\mathrm{SNF}_{\mathrm{XATL}}$. $\mathrm{SNF}_{\mathrm{XATL}}$ clauses are formulae of the following forms:

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

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

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

$$\langle\langle\emptyset\rangle\rangle\Box(\textstyle\bigwedge_{i=1}^{n} l_i \Rightarrow [\![A]\!]\bigcirc \bigvee_{j=1}^{k} m_j) \qquad \text{(negative step clause)}$$

where

1. $k \geq 0$, $n > 0$;
2. **start** is a propositional constant and only holds at exactly one state in a CGM;
3. l_i $(1 \leq i \leq n)$ and m_j $(1 \leq j \leq k)$ are literals, that is atomic propositions or their negation;
4. A in a step clause is a (possible empty) set of agents, and A in a negative step clause is a (possible empty) set of agents but not Σ; and
5. $\langle\langle\emptyset\rangle\rangle\Box$ is a temporal operator in ATL, which is defined in the next section.

As all clauses are of the form $\langle\langle\emptyset\rangle\rangle\Box(P \Rightarrow D)$ we often simply write $P \Rightarrow D$ instead. In the remainder of this chapter, the two terms, a set of $\mathrm{SNF_{XATL}}$ clauses and a conjunction of $\mathrm{SNF_{XATL}}$ clauses, are used interchangeably like we did for $\mathrm{SNF^{g}_{CTL}}$ clauses.

Note that our intention of developing the calculus $\mathsf{R_{XATL}}$ is to utilise $\mathsf{R_{XATL}}$ as a stepping stone to eventually develop a resolution calculus for the full ATL. Therefore, we design our normal form for XATL in SNF-style, i.e. each clause in normal form is an implication with the operator $\Box$ outside. Thus, we introduce the operator $\Box$ into our normal form although the language XATL only contains the next step operator $\bigcirc$. Some simpler normal form (e.g. without the operator $\Box$) for XATL might exist and possibly can be operated by some resolution-style inference rules. However they are usually very hard to be reused once the operator $\Diamond, \Box, \mathcal{U}, \mathcal{W}$ are included in the language.

5.3.2 Semantics of $\mathrm{SNF_{XATL}}$

Before we define the semantics of $\mathrm{SNF_{XATL}}$, we need to introduce the notion of agents' strategies and the outcomes of such strategies.

Given a natural number $n \in \mathbb{N}$, we denote the set of finite sequences of elements of S of the length n by S^n. The length of a sequence χ is denoted by $|\chi|$, for every $i, 0 \leq i < |\chi|$, the ith element of χ is denoted by $\chi[i]$ and for all $i, j, 0 \leq i \leq j < |\chi|, \chi[i,j]$ denotes the sequence $\chi[i], \chi[i+1], \ldots, \chi[j]$. The last element of a non-empty sequence χ, i.e. $\chi[|\chi|-1]$, is also denoted by $last(\chi)$. Given two sequences χ_1 and χ_2, the concatenation χ of χ_1 and χ_2 is denoted by $\chi_1 \circ \chi_2$.

Definition 5.14 *A-strategy*
Let $A \subseteq \Sigma$ be a coalition. A *strategy for the coalition A* (or for brevity, an *A-strategy*) is a map $F_A : \bigcup_{1<n} S^n \to \bigcup\{D_A(s) \mid s \in S\}$ such that for every $\chi \in \bigcup_{1<n} S^n$, $F_A(\chi) \in D_A(last(\chi))$.

Definition 5.15 *Outcome of an A-strategy*
Let F_A be an A-strategy. The *outcome* of F_A at state s, denoted by $out(s, F_A)$, is a set of all s-runs λ such that $\lambda[i+1] \in out(\lambda[i], F_A(\lambda[0,i]))$, for all $i \geq 0$.

Note that according to Definition 5.15, the outcome of $F_\emptyset$ at state s is the set of all s-runs λ such that $\lambda[i+1] \in out(\lambda[i], \sigma_\emptyset)$, for all $i \geq 0$. Thus, the outcome of $F_\emptyset$ is the set of all s-runs.

The semantics of SNF_{XATL} is defined as shown below as an extension of the semantics of XATL defined in 5.2.2.

$$\begin{aligned}\langle M, s\rangle \models \langle\langle A\rangle\rangle\Box\psi \text{ iff there exists an } A\text{-strategy } F_A \text{ such that, for all } \lambda \in out(s, F_A)\\ \text{and all positions } i \geq 0, \langle M, \lambda[i]\rangle \models \psi\end{aligned}$$

A SNF_{XATL} formula φ is *tight satisfiable* iff for some CGM $M = (\Sigma_\varphi, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ and some state $s \in S$, $\langle M, s\rangle \models \varphi$, and *tight unsatisfiable* otherwise. φ is *valid*, written $\models \varphi$, iff for every CGM $M = (\Sigma_\varphi, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ and some state $s \in S$, $\langle M, s\rangle \models \varphi$. A CGM $M = (\Sigma_\varphi, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ such that φ is true at some state $s \in S$ is called a *model* for φ.

5.3.3 Transformation

Before going into the details of the transformation of an arbitrary XATL formula into its normal form, we first define the notion of an XATL clause as an intermediate form between XATL formulae and SNF_{XATL} clauses.

Definition 5.16 *XATL clauses*
An XATL formula in the form of $\langle\langle\emptyset\rangle\rangle\Box(P \Rightarrow \varphi)$, where P is a conjunction of literals (possibly consisting of a single literal) or a propositional constant and φ is an arbitrary XATL formula, is an *XATL clause* or a *clause*.

It should be noted that a formula of XATL in SNF_{XATL} must be an XATL clause, but the converse is not necessarily true.

In the following we define a set of transformation rules which allows us to transform an arbitrary XATL formula into an equi-satisfiable set of SNF_{XATL} clauses.

Let *nnf* denote a function which transforms an arbitrary XATL formula into its negation normal form by pushing negations "inwards". Let *simp* be a function which simplifies an arbitrary XATL formula by exhaustive application of the following simplification rules:

$$\begin{array}{rcl}(\varphi \wedge \mathbf{true}) \longrightarrow \varphi & \qquad & (\varphi \wedge \mathbf{false}) \longrightarrow \mathbf{false}\\ (\varphi \vee \mathbf{true}) \longrightarrow \mathbf{true} & & (\varphi \vee \mathbf{false}) \longrightarrow \varphi\\ \neg\mathbf{true} \longrightarrow \mathbf{false} & & \neg\mathbf{false} \longrightarrow \mathbf{true}\end{array}$$

where φ is an XATL formula and $\vee$ and $\wedge$ are commutative and associative, plus the following rules.

$$\langle\!\langle A\rangle\!\rangle\bigcirc\mathbf{false} \longrightarrow \mathbf{false} \qquad \langle\!\langle A\rangle\!\rangle\bigcirc\mathbf{true} \longrightarrow \mathbf{true}$$

$$[\![A]\!]\bigcirc\mathbf{false} \longrightarrow \mathbf{false} \qquad [\![A]\!]\bigcirc\mathbf{true} \longrightarrow \mathbf{true}$$

The correctness proof for the simplification rules above straightforwardly follows from the semantics of XATL.

Let $\mathit{init}(\varphi)$ be the set of XATL formulae $\{\langle\!\langle\emptyset\rangle\!\rangle\Box(\mathbf{start} \Rightarrow p), \langle\!\langle\emptyset\rangle\!\rangle\Box(p \Rightarrow \mathit{simp}(\mathit{nnf}(\varphi)))\}$, where p is a new proposition that does not occur in φ. Then the transformation of an arbitrary XATL formula φ into $\mathrm{SNF}_{\mathrm{XATL}}$ consists of a sequence $T_0, T_1, \ldots, T_n$ of sets of XATL clauses such that (i) $T_0 = \mathit{init}(\varphi)$ and (ii) for every $t, 0 \leq t < n, T_{t+1} = (T_t \setminus \{\psi\}) \cup R_t$, where ψ is a clause in T_t not in $\mathrm{SNF}_{\mathrm{XATL}}$ and R_t is the result of applying a matching transformation rule to ψ. Moreover, for every $i, 0 \leq t < n$, T_t contains at least one formula not in $\mathrm{SNF}_{\mathrm{XATL}}$ while all formulae in T_n are in $\mathrm{SNF}_{\mathrm{XATL}}$.

Note that for each rule of *Trans* below containing a proposition p, p represents a new atomic proposition which does not occur in T_t, when we apply the rule to a clause in T_t. Furthermore, in the presentation of the rules let

- q be an atomic proposition,
- l be a literal,
- D be a disjunction of literals (possibly consisting of a single literal),
- φ, φ_1 and φ_2, be XATL formulae,
- Σ be the set containing all the agents occurring in T_0 and $A \subseteq \Sigma$ be a coalition.

The rule set *Trans* consists of the following rules:

$$\mathit{Trans}(1) \qquad q \Rightarrow \varphi_1 \wedge \varphi_2 \longrightarrow \begin{cases} q \Rightarrow \varphi_1 \\ q \Rightarrow \varphi_2 \end{cases}$$

$$\mathit{Trans}(2) \qquad q \Rightarrow \varphi_1 \vee \varphi_2 \longrightarrow \begin{cases} q \Rightarrow \varphi_1 \vee p \\ p \Rightarrow \varphi_2 \end{cases} \quad \text{if } \varphi_2 \text{ is not a disjunction of literals.}$$

$$\mathit{Trans}(3) \qquad q \Rightarrow D \longrightarrow \mathbf{true} \Rightarrow \neg q \vee D$$

$$\mathit{Trans}(4) \qquad q \Rightarrow \langle\!\langle A\rangle\!\rangle\bigcirc\varphi \longrightarrow \begin{cases} q \Rightarrow \langle\!\langle A\rangle\!\rangle\bigcirc p \\ p \Rightarrow \varphi \end{cases} \quad \text{if } \varphi \text{ is not a disjunction of literals.}$$

$$\mathit{Trans}(5) \quad q \Rightarrow [\![A]\!]\bigcirc\varphi \longrightarrow \begin{cases} q \Rightarrow [\![A]\!]\bigcirc p \\ p \Rightarrow \varphi \end{cases} \quad \text{if } \varphi \text{ is not a disjunction of literals and } A \neq \Sigma.$$

$$\mathit{Trans}(6) \quad q \Rightarrow [\![\Sigma]\!]\bigcirc\varphi \longrightarrow \quad q \Rightarrow \langle\!\langle\emptyset\rangle\!\rangle\bigcirc\varphi$$

We now give an example illustrating the transformation of an XATL formula into a satisfiability equivalent set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses using our transformation rules.

Example 5.5
Let $\varphi = ([\![1]\!]\bigcirc[\![1]\!]\bigcirc q \vee \langle\!\langle 1,2\rangle\!\rangle\bigcirc r) \wedge [\![2]\!]\bigcirc\neg q$. First, we apply the function *init* to φ.

$$T_0 = \mathit{init}(\varphi) = \{\langle\!\langle\emptyset\rangle\!\rangle\Box(\mathbf{start} \Rightarrow p_0),\ \langle\!\langle\emptyset\rangle\!\rangle\Box(p_0 \Rightarrow ([\![1]\!]\bigcirc[\![1]\!]\bigcirc q \vee \langle\!\langle 1,2\rangle\!\rangle\bigcirc r) \wedge [\![2]\!]\bigcirc\neg q)\}.$$

Then we transform the set T_0 of XATL clauses into a set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses.

1.	$\mathbf{start} \Rightarrow p_0$	
2.	$p_0 \Rightarrow ([\![1]\!]\bigcirc[\![1]\!]\bigcirc q \vee \langle\!\langle 1,2\rangle\!\rangle\bigcirc r) \wedge [\![2]\!]\bigcirc\neg q$	
3.	$p_0 \Rightarrow [\![1]\!]\bigcirc[\![1]\!]\bigcirc q \vee \langle\!\langle 1,2\rangle\!\rangle\bigcirc r$	$\mathit{Trans}(1) \rightarrow 2$
4.	$p_0 \Rightarrow [\![2]\!]\bigcirc\neg q$	$\mathit{Trans}(1) \rightarrow 2$
5.	$p_0 \Rightarrow p_1 \vee \langle\!\langle 1,2\rangle\!\rangle\bigcirc r$	$\mathit{Trans}(2) \rightarrow 3$
6.	$p_1 \Rightarrow [\![1]\!]\bigcirc[\![1]\!]\bigcirc q$	$\mathit{Trans}(2) \rightarrow 3$
7.	$p_0 \Rightarrow p_1 \vee p_2$	$\mathit{Trans}(2) \rightarrow 5$
8.	$p_2 \Rightarrow \langle\!\langle 1,2\rangle\!\rangle\bigcirc r$	$\mathit{Trans}(2) \rightarrow 5$
9.	$p_1 \Rightarrow [\![1]\!]\bigcirc p_3$	$\mathit{Trans}(4) \rightarrow 6$
10.	$p_3 \Rightarrow [\![1]\!]\bigcirc q$	$\mathit{Trans}(4) \rightarrow 6$
11.	$\mathbf{true} \Rightarrow \neg p_0 \vee p_1 \vee p_2$	$\mathit{Trans}(3) \rightarrow 7$

where $\mathit{Trans}(x) \rightarrow y$ indicates that we apply the transformation rule $\mathit{Trans}(x)$ to the clause labelled by y. In total, the transformation requires five applications of one of our transformation rules.

The set T_5 of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses we obtain as the final result is

1.	$\mathbf{start} \Rightarrow p_0$
4.	$p_0 \Rightarrow [\![2]\!]\bigcirc\neg q$
8.	$p_2 \Rightarrow \langle\!\langle 1,2\rangle\!\rangle\bigcirc r$
9.	$p_1 \Rightarrow [\![1]\!]\bigcirc p_3$
10.	$p_3 \Rightarrow [\![1]\!]\bigcirc q$
11.	$\mathbf{true} \Rightarrow \neg p_0 \vee p_1 \vee p_2$

and $\varphi = ([\![1]\!]\bigcirc[\![1]\!]\bigcirc q \vee \langle\!\langle 1,2\rangle\!\rangle\bigcirc r) \wedge [\![2]\!]\bigcirc\neg q$ is satisfiable iff T_5 is satisfiable (as will be shown in Section 5.5.1).

5.4 The clausal resolution calculus R_{XATL}

The clausal resolution calculus R_{XATL} is based on the calculus $\mathsf{R}^{\succ,S}_{\text{CTL}}$ we presented in Section 3, but it is not a trivial extension of this calculus. The calculus R_{XATL} consists of (i) seven *step resolution* rules, SRES1 to SRES7, and (ii) two *rewrite* rules, RW1 and RW2. All the rules of R_{XATL} operate on formulae in SNF$_{\text{XATL}}$. Below we present the definition of the step resolution rules of R_{XATL} and present some examples illustrating the application of those rules to SNF$_{\text{XATL}}$ clauses.

5.4.1 Step resolution

In the following, let P, Q be conjunctions of literals; C, D be disjunctions of literals; l be a literal; $\emptyset$ be an empty set of agents; $A, A' \subseteq \Sigma$ be a set of agents.

SRES1

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

SRES2

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

SRES3

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

SRES4

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

where $A \cap A' = \emptyset$

SRES5

$$\frac{\begin{array}{c}P \Rightarrow \langle\langle A \rangle\rangle \bigcirc (C \vee l) \\ Q \Rightarrow [\![A']\!] \bigcirc (D \vee \neg l)\end{array}}{P \wedge Q \Rightarrow [\![A' \setminus A]\!] \bigcirc (C \vee D)}$$

where $A \subseteq A'$

SRES6

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

SRES7

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

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

RW2 $\quad \bigwedge_{i=1}^{n} m_i \Rightarrow [\![A]\!] \bigcirc \mathbf{false} \longrightarrow \mathbf{true} \Rightarrow \bigvee_{i=1}^{n} \neg m_i$

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

It should be noted that we assume $\mathrm{SNF}_{\mathrm{XATL}}$ clauses are always in their condensed forms, i.e. there are no duplicate literals in the left-hand side or right-hand side of clauses. For example, for $\mathrm{SNF}_{\mathrm{XATL}}$ clauses $p \wedge p \wedge q \Rightarrow \langle\langle A\rangle\rangle\bigcirc(r \vee q)$ and $p \Rightarrow [\![A]\!]\bigcirc(q \vee \neg l \vee \neg l)$, their condensed forms are $p \wedge q \Rightarrow \langle\langle A\rangle\rangle\bigcirc(r \vee q)$ and $p \Rightarrow [\![A]\!](q \vee \neg l)$, respectively.

We now introduce some useful definitions which allows us to discuss the calculus $\mathsf{R}_{\mathrm{XATL}}$ precisely.

Definition 5.17 *Derivation*
A *derivation* from a set T of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses by $\mathsf{R}_{\mathrm{XATL}}$ is a sequence $T_0, T_1, T_2, \ldots$ of sets of clauses such that $T = T_0$ and $T_{t+1} = T_t \cup R_t$ where R_t is a set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses obtained as the conclusion of an application of a rule of $\mathsf{R}_{\mathrm{XATL}}$ to premises in T_t.

Definition 5.18 *Refutation*
A *refutation* of a set T of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses (by $\mathsf{R}_{\mathrm{XATL}}$) is a derivation from T such that for some $t \geq 0$, T_t contains a contradiction, where a contradiction is either the formula $\mathbf{true} \Rightarrow \mathbf{false}$ or $\mathbf{start} \Rightarrow \mathbf{false}$.

Definition 5.19 *Termination*
A derivation *terminates* iff either a contradiction is derived or if no new clauses can be derived by further applications of resolution rules of $\mathsf{R}_{\mathrm{XATL}}$.

Definition 5.20 *Saturation with respect to* $\mathsf{R}_{\mathrm{XATL}}$
A set T of $\mathrm{SNF}_{\mathrm{XATL}}$ clause is *saturated* with respect to $\mathsf{R}_{\mathrm{XATL}}$ if all clauses that can be derived by an application of a rule of $\mathsf{R}_{\mathrm{XATL}}$ to premises in T are contained in T.

In the following, we give two examples of refutations of unsatisfiable sets of $\mathrm{SNF}_{\mathrm{XATL}}$ by $\mathsf{R}_{\mathrm{XATL}}$.

Example 5.6
Consider the set T of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses consisting of the clauses 1 to 4 shown below.

$$\begin{array}{ll}
1. & \mathbf{start} \Rightarrow p_0 \\
2. & \quad p_0 \Rightarrow \langle\langle 1\rangle\rangle\bigcirc(p_1 \vee p_2) \\
3. & \quad p_0 \Rightarrow \langle\langle 2\rangle\rangle\bigcirc\neg p_1 \\
4. & \quad p_0 \Rightarrow \langle\langle 3\rangle\rangle\bigcirc\neg p_2
\end{array}$$

We are able to derive a contradiction $\mathbf{start} \Rightarrow \mathbf{false}$ using step resolution and rewrite rules as follows:

$$\begin{array}{lll}
5. & \quad p_0 \Rightarrow \langle\langle 1,2\rangle\rangle\bigcirc p_2 & [2, 3, \mathrm{SRES4}] \\
6. & \quad p_0 \Rightarrow \langle\langle 1,2,3\rangle\rangle\bigcirc\mathbf{false} & [4, 5, \mathrm{SRES4}] \\
7. & \mathbf{true} \Rightarrow \neg p_0 & [6, \mathrm{RW1}] \\
8. & \mathbf{start} \Rightarrow \mathbf{false} & [1, 7, \mathrm{SRES1}]
\end{array}$$

Example 5.7

Consider the set T of SNF_{XATL} clauses consisting of the clauses 1 to 6 shown below.

$$\begin{array}{rl}
1. & \textbf{start} \Rightarrow p_0 \\
2. & p_0 \Rightarrow \langle\langle 1\rangle\rangle\bigcirc p_1 \\
3. & p_0 \Rightarrow \langle\langle 2\rangle\rangle\bigcirc p_3 \\
4. & p_0 \Rightarrow \langle\langle 1,2,3\rangle\rangle\bigcirc p_4 \\
5. & p_0 \Rightarrow [\![1,2]\!]\bigcirc(\neg p_1 \vee \neg p_2 \vee \neg p_3) \\
6. & \textbf{true} \Rightarrow \neg p_1 \vee p2
\end{array}$$

We are able to derive a contradiction $\textbf{start} \Rightarrow \textbf{false}$ using step resolution and rewrite rules as follows:

$$\begin{array}{rll}
7. & p_0 \Rightarrow [\![1,2]\!]\bigcirc(\neg p_1 \vee \neg p_3) & [5,6,\text{SRES7}] \\
8. & p_0 \Rightarrow [\![2]\!]\bigcirc\neg p_3 & [2,7,\text{SRES5}] \\
9. & p_0 \Rightarrow [\![\emptyset]\!]\bigcirc\textbf{false} & [3,8,\text{SRES5}] \\
10. & \textbf{true} \Rightarrow \neg p_0 & [9,\text{RW2}] \\
11. & \textbf{start} \Rightarrow \textbf{false} & [1,10,\text{SRES1}]
\end{array}$$

5.5 Correctness of the calculus R_{XATL}

5.5.1 Correctness of the transformation

In Section 5.3.3 we have presented rules for the transformation of an arbitrary formula of XATL into a set of SNF_{XATL} clauses. In the following, we show that our transformation (i) preserves satisfiability, (ii) is terminating, and (iii) allows only a polynomial bounded number of applications of transformation rules.

Lemma 5.1 *Let T be a set of XATL formulae, and let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM such that T is satisfiable in M. Let $p \in \mathsf{P}_{\mathsf{PL}}$ be an atomic proposition not occurring in T, and let $M' = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L')$ be a CGM identical to M except for the truth value assigned by L' to p in each state. Then T is also satisfiable in M'.*

Proof. By the inductive definition of the semantics of XATL, 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'. □

Lemma 5.2 *An XATL formula φ is satisfiable iff the set of formulae $T_0 = \{\langle\langle\emptyset\rangle\rangle\Box(\textbf{start} \Rightarrow p), \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)\}$, where p is a new atomic proposition that does not occur in φ, is satisfiable.*

Proof. Assume $\{\langle\langle\emptyset\rangle\rangle\Box(\textbf{start} \Rightarrow p), \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)\}$ is satisfiable in a CGM $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$, i.e. $M, s_0 \models \langle\langle\emptyset\rangle\rangle\Box(\textbf{start} \Rightarrow p) \wedge \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)$, for some state $s_0 \in S$ which also satisfies $\textbf{start}$. From the semantics of $\Rightarrow, \langle\langle\emptyset\rangle\rangle\Box, \wedge$, $M, s_0 \models (\textbf{start} \Rightarrow p) \wedge (p \Rightarrow \varphi)$. From the semantics of $\Rightarrow, \wedge$, $M, s_0 \models \textbf{start} \Rightarrow \varphi$. Because $\textbf{start}$ holds at s_0, $M, s_0 \models \varphi$. Thus, if $\{\langle\langle\emptyset\rangle\rangle\Box(\textbf{start} \Rightarrow p), \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)\}$ is satisfiable, so is φ.

Assume φ is satisfiable in a CGM $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$, i.e. $M, s_0 \models \varphi$ for some state $s_0 \in S$. Let $M' = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L')$ be identical to M except that $\textbf{start}$ and a new proposition p not occurring in φ hold only at s_0. From the semantics of $\Rightarrow, \langle\langle\emptyset\rangle\rangle\Box$, $M', s_0 \models \langle\langle\emptyset\rangle\rangle\Box(\mathbf{start} \Rightarrow p)$. From Lemma 5.1, $M', s_0 \models \varphi$. From the semantics of $\Rightarrow, \langle\langle\emptyset\rangle\rangle\Box$, $M', s_0 \models \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)$. From the semantics of $\wedge$, $M', s_0 \models \langle\langle\emptyset\rangle\rangle\Box(\mathbf{start} \Rightarrow p) \wedge \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)$. Thus, if φ is satisfiable, so is $\{\langle\langle\emptyset\rangle\rangle\Box(\mathbf{start} \Rightarrow p), \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)\}$. □

Now we show every transformation rule preserves satisfiability, starting with rule *Trans*(1).

Lemma 5.3 *Let $T_t = \Delta \cup \{\psi\}$, where $\psi = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_1 \wedge \varphi_2)$, and $T_{t+1} = \Delta \cup R_t$, where $R_t = \{\langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_1), \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_2)\}$, be two sets of XATL clauses such that T_{t+1} is obtained by an application of Trans(1) to the formula ψ in T_t. Then T_t is satisfiable iff T_{t+1} is satisfiable.*

Proof. Assume $T_t = \Delta \wedge \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_1 \wedge \varphi_2)$ holds in a CGM $M = (\Sigma, S, d, \delta, \mathsf{P_{PL}}, L)$ at a state s_0 in S. Based on the semantics of the logical connectives involved, we have that

$\langle M, s_0\rangle \models \Delta \wedge \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_1 \wedge \varphi_2)$

iff $\langle M, s_0\rangle \models \Delta \wedge \langle\langle\emptyset\rangle\rangle\Box((q \Rightarrow \varphi_1) \wedge (q \Rightarrow \varphi_2))$

iff $\langle M, s_0\rangle \models \Delta$ and there is a $\emptyset$-strategy $F_\emptyset$ such that for all $\lambda \in out(s, F_\emptyset)$ and all positions $i \geq 0, M, \lambda[i] \models q \Rightarrow \varphi_1$ and $M, \lambda[i] \models q \Rightarrow \varphi_2$

iff (i) $\langle M, s_0\rangle \models \Delta$, (ii) there is a $\emptyset$-strategy $F_\emptyset$ such that for all $\lambda \in out(s, F_\emptyset)$ and all positions $i \geq 0, M, \lambda[i] \models q \Rightarrow \varphi_1$, and (iii) there is a $\emptyset$-strategy $F_\emptyset$ such that for all $\lambda \in out(s, F_\emptyset)$ and all positions $i \geq 0$, $M, \lambda[i] \models q \Rightarrow \varphi_2$

iff $\langle M, s_0\rangle \models \Delta \wedge \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_1) \wedge \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_2)$

Therefore, T_t is satisfiable iff T_{t+1} is satisfiable. □

Next, we show that the transformation rule *Trans*(3) preserves satisfiability.

Lemma 5.4 *Let $T_t = \Delta \cup \{\psi\}$, where $\psi = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow D)$, and $T_{t+1} = \Delta \cup R_t$, where $R_t = \{\langle\langle\emptyset\rangle\rangle\Box(\mathbf{true} \Rightarrow \neg q \vee D)\}$, be two sets of XATL clauses such that T_{t+1} is obtained by an application of Trans(3) to the formula ψ in T_t. Then T_t is satisfiable iff T_{t+1} is satisfiable.*

Proof. $\langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow D)$ is obviously equivalent to $\langle\langle\emptyset\rangle\rangle\Box(\mathbf{true} \Rightarrow \neg q \vee D)$ as $q \Rightarrow D$ is propositionally equivalent to $\mathbf{true} \Rightarrow \neg q \vee D$. Therefore, T_t is actually equivalent to T_{t+1}. □

Next, we consider the transformation rule *Trans*(4).

Lemma 5.5 *Let $T_t = \Delta \cup \{\psi\}$, where $\psi = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi)$, and $T_{t+1} = \Delta \cup R_t$, where $R_t = \{\langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc p), \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)\}$, be two sets of XATL clauses such that T_{t+1} is obtained by an application of Trans(4) to the formula ψ in T_t. Then T_t is satisfiable iff T_{t+1} is satisfiable.*

Proof. We first show the 'if' part. Assume T_{t+1} holds in a CGM $M = (\Sigma, S, d, \delta, \mathsf{P_{PL}}, L)$ at a state s_0, i.e. $M, s_0 \models \Delta \wedge \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc p) \wedge \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)$. From the semantics of $\wedge$, **(1)** $M, s_0 \models \Delta$; from the semantics of $\langle\langle\emptyset\rangle\rangle\Box$, for all states $s_i \in S$ reachable from s_0, **(2)** $M, s_i \models q \Rightarrow \langle\langle A\rangle\rangle\bigcirc p$ and **(3)** $M, s_i \models p \Rightarrow \varphi$.

- If q does not hold at the state s_i, then $M, s_i \models q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi$.
- If, on the other hand, q holds at the state s_i, it follows from (2) and the semantics of $\langle\langle A\rangle\rangle\bigcirc$ that there exists an A-move $\sigma_A \in D_A(s_i)$ such that $M, s_{i+1} \models p$ for each state $s_{i+1} \in out(s_i, \sigma_A)$. From (3) and the semantics of $\Rightarrow$, we obtain that $M, s_{i+1} \models \varphi$ for each state $s_{i+1} \in out(s_i, \sigma_A)$. Therefore, from the semantics of $\langle\langle A\rangle\rangle\bigcirc$, $M, s_i \models \langle\langle A\rangle\rangle\bigcirc\varphi$. From the semantics of $\Rightarrow$, we obtain that $M, s_i \models q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi$.

From the semantics of $\langle\langle\emptyset\rangle\rangle\Box$, we obtain that $M, s_0 \models \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi)$. From (1) and semantics of $\wedge$, $M, s_0 \models \Delta \wedge \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi)$. Thus, if T_{t+1} is satisfiable, so is T_t.

Next, we prove the 'only if' part. Assume T_t is satisfiable in a CGM $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ at a state s_0, i.e. $M, s_0 \models \Delta \wedge \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi)$. Let $M' = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L')$ be a CGM identical to M except that p is true at a state $s_i \in S$ iff φ is true at the state s_i. By the definition of M' and the semantics of $\langle\langle\emptyset\rangle\rangle\Box$ and $\Rightarrow$, we obtain that **(4)** $M', s_0 \models \langle\langle\emptyset\rangle\rangle\Box(p \Leftrightarrow \varphi)$. By Lemma 5.1, as $M, s_0 \models \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi)$, $M', s_0 \models \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi)$. From the semantics of $\langle\langle\emptyset\rangle\rangle\Box$, for every state $s \in S$ reachable from s_0, $M', s \models q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi$.

- If $M', s \not\models q$, then $M', s \models q \Rightarrow \langle\langle A\rangle\rangle\bigcirc p$.
- If, on the other hand, $M', s \models q$, then $M', s \models \langle\langle A\rangle\rangle\bigcirc\varphi$. From the semantics of $\langle\langle A\rangle\rangle\bigcirc$, there exists an A-move $\sigma_A \in D_A(s)$ such that $M', s' \models \varphi$ for all states $s' \in out(s, \sigma_A)$. From (4), $M', s' \models p$. Therefore, from the semantics of $\langle\langle A\rangle\rangle\bigcirc$, we obtain $M', s \models \langle\langle A\rangle\rangle\bigcirc p$. From the semantics of $\Rightarrow$, $M', s \models q \Rightarrow \langle\langle A\rangle\rangle\bigcirc p$.

As s is an arbitrary state, from the semantics of $\langle\langle\emptyset\rangle\rangle\Box$, $M', s_0 \models \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle A\rangle\rangle\bigcirc p)$. From (4), $M', s_0 \models \langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \varphi)$. By Lemma 5.1, as p does not occur in Δ, $M', s_0 \models \Delta$. Thus, if T_t is satisfiable, so is T_{t+1}. □

Finally, we show that *Trans*(6) preserves satisfiability.

Lemma 5.6 *Let $T_t = \Delta \cup \{\psi\}$, where $\psi = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow [\![\Sigma]\!]\bigcirc\varphi)$, and $T_{t+1} = \Delta \cup R_t$, where $R_t = \{\langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle\emptyset\rangle\rangle\bigcirc\varphi)\}$, be two sets of XATL clauses such that T_{t+1} is obtained by an application of Trans(6) to the formula ψ in T_t. Then T_t is satisfiable iff T_{t+1} is satisfiable.*

Proof. In [7] (see also Appendix B), the equivalence $[\![\Sigma]\!]\bigcirc\varphi \equiv \langle\langle\emptyset\rangle\rangle\bigcirc\varphi$ is shown. By propositional reasoning, the formula $q \Rightarrow [\![\Sigma]\!]\bigcirc\varphi$ is equivalent to the formula $q \Rightarrow \langle\langle\emptyset\rangle\rangle\bigcirc\varphi$. □

Theorem 5.1 *Let $T_t = \Delta \cup \{\psi\}$ and $T_{t+1} = \Delta \cup R_t$ be two sets of XATL clauses such that T_{t+1} is obtained by an application of a transformation rule of the form $\psi \to R_t$ in the set Trans to the formula ψ in T_t. Then T_t is satisfiable iff T_{t+1} is satisfiable.*

Proof. To prove this theorem, we need to show that every transformation rule in the set *Trans* preserves satisfiability.

Lemmas 5.3, 5.4, 5.5, and 5.6 show that the transformation rules *Trans*(1), *Trans*(3), *Trans*(4), and *Trans*(6) preserve satisfiability, respectively.

The proof that *Trans*(2) preserves satisfiability is analogous to the proof for *Trans*(1) in Lemma 5.3. The proof that *Trans*(5) preserves satisfiability is analogous to the proof for *Trans*(4) in Lemma 5.5.

□

Weight functions for XATL formulae

To show that the transformation terminates, we assign weights to XATL clauses and sets of XATL clauses and prove that every application of a transformation rule strictly reduces the weight of a set of XATL clauses.

In particular, we define the following three weight functions

1. $w(\Gamma)$, which assigns a weight to an XATL clause Γ;
2. $w(\mathsf{L}, \varphi)$, which assigns a weight to an XATL formula φ occurring on the left-hand side of an XATL clause; and
3. $w(\mathsf{R}, \varphi)$, which assigns a weight to an XATL formula φ occurring on the right-hand side of an XATL clause.

Except for the case of atomic propositions, $w(\mathsf{L}, \varphi)$ and $w(\mathsf{R}, \varphi)$ have identical definitions. Therefore, to ease the following definition, we use $w(x, \varphi)$ where a case of the definition applies to both $w(\mathsf{L}, \varphi)$ and $w(\mathsf{R}, \varphi)$. Then the inductive definition of the three weight functions is as follows.

For every XATL clause $\Gamma = \langle\langle\emptyset\rangle\rangle\Box(\varphi_1 \Rightarrow \varphi_2)$, the weight $w(\Gamma)$ of Γ is defined as follows.

1. $w(\langle\langle\emptyset\rangle\rangle\Box(\varphi \Rightarrow \psi)) = w(\mathsf{L}, \varphi) + w(\mathsf{R}, \psi) + 1$;
2. $w(x, \mathbf{start}) = 1$;
3. $w(x, \mathbf{true}) = w(x, \mathbf{false}) = 1$;
4. $w(\mathsf{L}, p) = 5$;
5. $w(\mathsf{R}, p) = 1$;
6. $w(x, \neg\varphi) = w(x, \varphi)$;
7. $w(x, \varphi \wedge \psi) = w(x, \varphi) + w(x, \psi)+7$;
8. $w(x, \varphi \vee \psi) = w(x, \varphi) + w(x, \psi)+1$, where both φ and ψ are disjunctions of literals;
9. $w(x, \varphi \vee \psi) = w(x, \varphi) + w(x, \psi) + 9$, where only one of φ and ψ is a disjunction of literals;
10. $w(x, \varphi \vee \psi) = w(x, \varphi) + w(x, \psi) + 17$, where neither of φ and ψ is a disjunctions of literals;
11. $w(x, \langle\langle A\rangle\rangle\bigcirc\varphi) = w(x, \varphi) + 9$, where φ is not a disjunction of literals;
12. $w(x, \langle\langle A\rangle\rangle\bigcirc\varphi) = w(x, \varphi) + 1$, where φ is a disjunction of literals;
13. $w(x, [\![A]\!]\bigcirc\varphi) = w(x, \varphi) + 10$, where φ is not a disjunction of literals;
14. $w(x, [\![A]\!]\bigcirc\varphi) = w(x, \varphi) + 2$, where φ is a disjunction of literals;

It should be noted that a disjunction of literals can consist of a single literal.

For every set Δ of XATL clauses, the weight of Δ is

$$w(\Delta) = \sum_{\Gamma \in \Delta} w(\Gamma).$$

In the following, we prove that each application of a transformation rule to a clause Γ in a set T of XATL clauses results in a set T' of XATL clauses that weights strictly less than T.

Lemma 5.7 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_1 \wedge \varphi_2)$, be a set of XATL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma_1, \Gamma_2\}$, where $\Gamma_1 = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_1)$ and $\Gamma_2 = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \varphi_2)$, be a set of XATL clauses such that T_{t+1} is obtained by an application of Trans(1) to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - (w(\Delta) + w(\Gamma_1) + w(\Gamma_2)) > 0$. According to the definition of the weight function for XATL clauses, we have

$$\begin{aligned}
w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, \varphi_1 \wedge \varphi_2) + 1 \\
&= 5 + w(\mathsf{R}, \varphi_1) + w(\mathsf{R}, \varphi_2) + 7 + 1 \\
&= w(\mathsf{R}, \varphi_1) + w(\mathsf{R}, \varphi_2) + 13 \\
w(\Gamma_1) &= w(\mathsf{L}, q) + w(\mathsf{R}, \varphi_1) + 1 \\
&= 5 + w(\mathsf{R}, \varphi_1) + 1 \\
&= w(\mathsf{R}, \varphi_1) + 6 \\
w(\Gamma_2) &= w(\mathsf{L}, q) + w(\mathsf{R}, \varphi_2) + 1 \\
&= 5 + w(\mathsf{R}, \varphi_2) + 1 \\
&= w(\mathsf{R}, \varphi_2) + 6
\end{aligned}$$

Therefore, $w(\Delta) + w(\Gamma) - (w(\Delta) + w(\Gamma_1) + w(\Gamma_2)) = 1 > 0$. $\square$

Next, we consider the transformation rule *Trans*(3).

Lemma 5.8 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow D)$ and D is a disjunction of literals, be a set of XATL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma'\}$, where $\Gamma' = \langle\langle\emptyset\rangle\rangle\Box(\textbf{true} \Rightarrow \neg q \vee D)$, be a set of XATL clauses such that T_{t+1} is obtained by an application of Trans(3) to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - (w(\Delta) + w(\Gamma')) > 0$. According to the definition of the weight function for XATL clauses, we have

$$\begin{aligned} w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, D) + 1 \\ &= 5 + w(\mathsf{R}, D) + 1 \\ &= w(\mathsf{R}, D) + 6 \end{aligned}$$

$$\begin{aligned} w(\Gamma') &= w(\mathsf{L}, \mathbf{true}) + w(\mathsf{R}, \neg q \vee D) + 1 \\ &= 1 + w(\mathsf{R}, \neg q) + w(\mathsf{R}, D) + 1 + 1 \\ &= w(\mathsf{R}, q) + w(\mathsf{R}, D) + 3 \\ &= w(\mathsf{R}, D) + 4 \end{aligned}$$

Therefore, $w(\Delta) + w(\Gamma) - (w(\Delta) + w(\Gamma')) = 2 > 0$. □

Lemma 5.9 *Let $T_t = \Delta \cup \{\Gamma\}$, where $\Gamma = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow [\![\Sigma]\!]\bigcirc\varphi)$, be a set of XATL clauses. Let $T_{t+1} = \Delta \cup \{\Gamma'\}$, where $\Gamma' = \langle\langle\emptyset\rangle\rangle\Box(q \Rightarrow \langle\langle\emptyset\rangle\rangle\bigcirc\varphi)$, be a set of XATL clauses such that T_{t+1} is obtained by an application of Trans(6) to the formula Γ in T_t. Then the weight of T_t is strictly greater than the weight of T_{t+1}.*

Proof. We need to show that $w(T_t) - w(T_{t+1}) > 0$, i.e. $w(\Delta) + w(\Gamma) - (w(\Delta) + w(\Gamma')) > 0$.

If φ is a disjunction of literals, then we have

$$\begin{aligned} w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, [\![\Sigma]\!]\bigcirc\varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 2 + 1 \\ &= w(\mathsf{R}, \varphi) + 8 \end{aligned}$$

$$\begin{aligned} w(\Gamma') &= w(\mathsf{L}, q) + w(\mathsf{R}, \langle\langle\emptyset\rangle\rangle\bigcirc\varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 1 + 1 \\ &= w(\mathsf{R}, \varphi) + 7 \end{aligned}$$

Otherwise, we have

$$\begin{aligned} w(\Gamma) &= w(\mathsf{L}, q) + w(\mathsf{R}, [\![\Sigma]\!]\bigcirc\varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 10 + 1 \\ &= w(\mathsf{R}, \varphi) + 16 \end{aligned}$$

$$\begin{aligned} w(\Gamma') &= w(\mathsf{L}, q) + w(\mathsf{R}, \langle\langle\emptyset\rangle\rangle\bigcirc\varphi) + 1 \\ &= 5 + w(\mathsf{R}, \varphi) + 9 + 1 \\ &= w(\mathsf{R}, \varphi) + 15 \end{aligned}$$

Therefore, for both cases, $w(\Delta) + w(\Gamma) - (w(\Delta) + w(\Gamma')) = 1 > 0$. □

Theorem 5.2 *Let the set of XATL formulae T_{t+1} be obtained by an application of a transformation rule in Trans to a formula Γ in the set of XATL formulae T_t. Then the weight $w(T_t)$ of T_t is strictly greater than the weight $w(T_{t+1})$ of T_{t+1}.*

Proof. To show this theorem holds, we need to prove that $w(T_t) - w(T_{t+1}) > 0$ for each transformation rule. For the transformation rules *Trans*(1), *Trans*(3) and *Trans*(6) we have already done so in Lemma 5.7, 5.8 and 5.9, respectively. For the remaining transformation rules the result can be computed analogously. Below we only list the result of $w(T_t) - w(T_{t+1})$ for each rule.

Rule	$w(T_t) - w(T_{t+1})$	Rule	$w(T_t) - w(T_{t+1})$	Rule	$w(T_t) - w(T_{t+1})$
(1)	1	(2)	1	(3)	2
(4)	1	(5)	1	(6)	1

□

Lemma 5.10 *Let T be a set of XATL clauses. If T contains a clause Γ which is not in* SNF_{XATL}*, then there exists a transformation rule, which can be applied to Γ in T.*

Proof. According to the definition of the syntax of XATL formulae and SNF_{XATL} formulae, the possible forms of formulae occurring on the right-hand side of an XATL clause are the following: **true**, **false**, p, $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$, $\langle\langle A\rangle\rangle\bigcirc\varphi$, and $[\![A]\!]\bigcirc\varphi$, where p is a proposition and φ and ψ are XATL formulae. As we apply the functions *simp* and *nnf* at the beginning of the transformation, XATL formulae of the form $\mathbf{true}, \mathbf{false}, \neg\varphi$ (for a formula φ which is not a proposition), and $\varphi \Rightarrow \psi$ can not occur on the right-hand side of XATL clauses in T. For the remaining possible forms that Γ might take, the table below shows that if Γ is not a SNF_{XATL} clause, then there exists a transformation rule which can be applied to Γ.

Form	*Trans*
$q \Rightarrow p$	(3)
$q \Rightarrow \neg p$	(3)
$q \Rightarrow \varphi \wedge \psi$	(1)
$q \Rightarrow \varphi \vee \psi$	(2) or (3)
$q \Rightarrow \langle\langle A\rangle\rangle\bigcirc\varphi$	(4)
$q \Rightarrow [\![A]\!]\bigcirc\varphi$	(5) or (6)

□

Theorem 5.3 *Let $T_0, T_1, \ldots$ be a sequence of sets of XATL clauses such that $T_0 = init(\varphi)$ for some XATL formula φ and T_{t+1} is obtained from T_t by applying a transformation rule to a clause in T_t. Then the sequence $T_0, T_1, \ldots$ terminates, i.e. there exists an index $n, n \geq 0$, such that no transformation rule can be applied to any clause in T_n. Furthermore, all clauses in T_n are in* SNF_{XATL}.

Proof. Follows from Lemma 5.10 and Theorem 5.2. □

Using the notion of the size of an XATL formula we are able to characterise the computational complexity of the normal form transformation for XATL.

Definition 5.21 *Size of an XATL formula*
Let φ be an arbitrary XATL formula. The size of φ is the number of occurrences of constants, propositions, boolean operators and temporal operators in φ.

For examples, the size of $\langle\langle A\rangle\rangle\bigcirc p$ is 2, the size of $\mathbf{start} \Rightarrow p \vee q$ is 5, and the size of $q \Rightarrow [\![A]\!]\bigcirc(\langle\langle A\rangle\rangle\bigcirc p \vee (p \wedge q))$ is 9.

Theorem 5.4 *Let φ be an arbitrary XATL formula and T_n be a set of* $\mathrm{SNF}_{\mathrm{XATL}}$ *clauses obtained from $T_0 = init(\varphi)$ by n applications of our transformation rules. Then n is linearly bounded in the size of φ and the set T_n can be computed in polynomial time in the size of φ.*

Proof. Let φ be of size m and we assume φ is already in negation normal form. By the definition of the weight function, we know that the weight of $T_0 = init(\varphi)$ is $w(\langle\langle\emptyset\rangle\rangle\Box(\mathbf{start} \Rightarrow p)) + w(\langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \psi))$, where $\psi = simp(nnf(\varphi))$. Furthermore,

$$\begin{aligned} w(\langle\langle\emptyset\rangle\rangle\Box(\mathbf{start} \Rightarrow p)) &= w(\mathsf{L}, \mathbf{start}) + w(\mathsf{R}, p) + 1 \\ &= 1 + 1 + 1 \\ &= 3 \end{aligned}$$

and

$$\begin{aligned} w(\langle\langle\emptyset\rangle\rangle\Box(p \Rightarrow \psi)) &= w(\mathsf{L}, p) + w(\mathsf{R}, \psi) + 1 \\ &= 5 + w(\mathsf{R}, \psi) + 1 \\ &= w(\mathsf{R}, \psi) + 6 \end{aligned}$$

Therefore, $w(T_0) = w(\mathsf{R}, \psi) + 9$. It is not hard to see that the function *simp* only reduces the size and weight of φ. Thus, the size and weight of ψ is bounded by the size and weight of φ, respectively. As the maximal weight of a constant, proposition, boolean operator, or temporal operator is 17, the weight $w(\mathsf{R}, \psi)$ is bounded by $17m + 9$. Since, by Theorem 5.2, each application of a transformation rule to a set of XATL clauses T_t results a set of XATL clauses T_{t+1} with $w(T_{t+1}) \leq w(T_t) - 1$, T_n can be computed in less than $17m + 9$ applications of the transformation rules.

Regarding the complexity of each application, we assume XATL clauses are stored in a tree data structure. For example, the tree in Figure 5.1 represents the XATL clause $p \Rightarrow \langle\langle A\rangle\rangle\bigcirc(q_1 \vee [\![A]\!]\bigcirc q_2)$. Then according to our transformation rules, by reusing the subtrees representing subformulae as appropriate, generating the XATL clauses resulting from an application of a transformation rule can be accomplished in constant time in the size of φ. The pattern matching procedure determining which rule to apply to an XATL clause, which is not in $\mathrm{SNF}_{\mathrm{XATL}}$, requires linear time in the size of φ in the worst case.

Therefore, the set T_n can be computed in polynomial time in the size of φ. □

$$\begin{tikzcd}
 & \Rightarrow \arrow[dl] \arrow[dr] & & \\
p & & \langle\!\langle A \rangle\!\rangle\bigcirc \arrow[d] & \\
 & & \vee \arrow[dl] \arrow[dr] & \\
 & q_1 & & [\![A]\!]\bigcirc \arrow[d] \\
 & & & q_2
\end{tikzcd}$$

Figure 5.1: $p \Rightarrow \langle\!\langle A \rangle\!\rangle\bigcirc(q_1 \vee [\![A]\!]\bigcirc q_2)$ stored in a tree structure

Theorem 5.5 *Let φ be an arbitrary XATL formula and T_n be a set of* $\mathrm{SNF}_{\mathrm{XATL}}$ *clauses obtained from $T_0 = init(\varphi)$ by a linearly bounded applications of our transformation rules in the size of φ. Then φ is satisfiable iff T_n is satisfiable.*

Proof. Follows from Theorem 5.3, Lemma 5.2, Theorem 5.1 and Theorem 5.4. $\square$

5.5.2 Soundness and completeness

We now establish the soundness of $\mathsf{R}_{\mathrm{XATL}}$ via the semantics of XATL. Before we state and prove the soundness theorem for $\mathsf{R}_{\mathrm{XATL}}$, we first establish a number of properties of the ordering $\sqsubseteq$ on A-moves that will be used in the proof of that theorem.

Lemma 5.11 *Let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM, s be a state in S and A, A' be coalitions. If the operation $\oplus$ is applicable to an A-move $\sigma_A \in D_A(s)$ and an A'-move $\sigma_{A'} \in D_{A'}(s)$, then $\sigma_A \sqsubseteq \sigma_A \oplus \sigma_{A'}$ and $\sigma_{A'} \sqsubseteq \sigma_A \oplus \sigma_{A'}$.*

Proof. Let $\sigma_{A''}$ be $\sigma_A \oplus \sigma_{A'}$. By the definition of the operation $\oplus$, $A'' \supseteq A$ and for every agent $a \in A$, $\sigma_A(a) = \sigma_{A''}(a)$. Therefore, $\sigma_A \sqsubseteq \sigma_{A''}$. Analogously for $\sigma_{A'}$ and $\sigma_{A''}$. $\square$

Lemma 5.12 *Let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM, s be a state in S and A, A' be coalitions. If there exist an A-move $\sigma_A \in D_A(s)$, an A'-move $\sigma_{A'} \in D_{A'}(s)$, and a move vector $\sigma \in D(s)$ such that $\sigma_A \sqsubseteq \sigma_{A'}$ and $\sigma_{A'} \sqsubseteq \sigma$, then $\sigma_A \sqsubseteq \sigma$.*

Proof. From $\sigma_A \sqsubseteq \sigma_{A'}$ and $\sigma_{A'} \sqsubseteq \sigma$, we obtain that (i) for every agent $a \in A, \sigma_A(a) = \sigma_{A'}(a)$; (ii) for every agent $a' \in A', \sigma_{A'}(a') = \sigma(a')$ and (iii) $A \subseteq A'$. Therefore, for every agent $a \in A, \sigma_A(a) = \sigma(a)$. Thus $\sigma_A \sqsubseteq \sigma$. $\square$

Lemma 5.13 *Let $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ be a CGM, s be a state in S and A, A' be coalitions. If there exist an A-move $\sigma_A \in D_A(s)$ and an A'-move $\sigma_{A'} \in D_{A'}(s)$ such that $\sigma_A \sqsubseteq \sigma_{A'}$, then $out(s, \sigma_{A'}) \subseteq out(s, \sigma_A)$.*

Proof. Let Δ and Δ' be the set of move vectors $\{\sigma \mid \sigma_A \sqsubseteq \sigma\}$ and the set of move vectors $\{\sigma' \mid \sigma_{A'} \sqsubseteq \sigma'\}$, respectively. As $\sigma_A \sqsubseteq \sigma_{A'}$, by Lemma 5.12 every $\sigma' \in \Delta'$ must also be an element of Δ. Therefore, $\Delta' \subseteq \Delta$. By the definition of the transition function δ, we obtain $\{\delta(s,\sigma') \mid \sigma' \in \Delta'\} \subseteq \{\delta(s,\sigma) \mid \sigma \in \Delta\}$. By the definition of the outcome of σ_A and $\sigma_{A'}$ at state s, we obtain $out(s,\sigma_{A'}) \subseteq out(s,\sigma_A)$. $\square$

Theorem 5.6 (Soundness of R_{XATL}). *Let T be a set of* SNF_{XATL} *clauses. If there is a refutation of T by R_{XATL}, then T is unsatisfiable.*

Proof. Let $T_0, T_1, \ldots, T_n$ be a derivation from a set of SNF_{XATL} clauses $T = T_0$ by the calculus R_{XATL}. 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, the claim obviously holds. Now, consider the step of the derivation in which we derive T_{t+1} from T_t for some $t \geq 0$. Assume T_t holds in $M = (\Sigma, S, d, \delta, \mathsf{P}_{\mathsf{PL}}, L)$ at a state $s_0 \in S$.

Assume $\langle\langle\emptyset\rangle\rangle\Box(P \Rightarrow \langle\langle A\rangle\rangle\bigcirc(C \vee l))$ and $\langle\langle\emptyset\rangle\rangle\Box(Q \Rightarrow \langle\langle A'\rangle\rangle\bigcirc(D \vee \neg l))$ are in T_t and that the coalitions A and A' are disjoint. Let T_{t+1} be obtained by an application of SRES4 to $\langle\langle\emptyset\rangle\rangle\Box(P \Rightarrow \langle\langle A\rangle\rangle\bigcirc(C \vee l))$ and $\langle\langle\emptyset\rangle\rangle\Box(Q \Rightarrow \langle\langle A'\rangle\rangle\bigcirc(D \vee \neg l))$, that is, $T_{t+1} = T_t \cup \{\langle\langle\emptyset\rangle\rangle\Box(P \wedge Q \Rightarrow \langle\langle A \cup A'\rangle\rangle\bigcirc(C \vee D))\}$. We show that M also satisfies T_{t+1}.

Consider an arbitrary state s reachable from $s_0 \in S$. If either $M, s \not\models P$ or $M, s \not\models Q$, then $M, s \models P \wedge Q \Rightarrow \langle\langle A \cup A'\rangle\rangle\bigcirc(C \vee D)$. If, on the other hand, $M, s \models P$ and $M, s \models Q$, then from $M, s_0 \models \langle\langle\emptyset\rangle\rangle\Box(P \Rightarrow \langle\langle A\rangle\rangle\bigcirc(C \vee l))$, $M, s_0 \models \langle\langle\emptyset\rangle\rangle\Box(Q \Rightarrow \langle\langle A'\rangle\rangle\bigcirc(D \vee \neg l))$, and the semantics of $\langle\langle\emptyset\rangle\rangle\Box$, we obtain $M, s \models \langle\langle A\rangle\rangle\bigcirc(C \vee l)$ and $M, s \models \langle\langle A'\rangle\rangle\bigcirc(D \vee \neg l)$. From the semantics of $M, s \models \langle\langle A\rangle\rangle\bigcirc(C \vee l)$, we know there exists an A-move $\sigma_A \in D_A(s)$ such that for every state $s_i \in out(s,\sigma_A), M, s_i \models C \vee l$. For an analogous reason, we know there exists an A'-move $\sigma_{A'} \in D_{A'}(s)$ such that for every state $s_j \in out(s,\sigma_{A'}), M, s_j \models D \vee \neg l$. As A and A' are disjoint, for $A'' = A \cup A'$, there exists an A''-move $\sigma_{A''} = \sigma_A \oplus \sigma_{A'}$. By Lemma 5.11, $\sigma_A \sqsubseteq \sigma_{A''}$ and $\sigma_{A'} \sqsubseteq \sigma_{A''}$. By Lemma 5.13, $out(s,\sigma_{A''}) \subseteq out(s,\sigma_A)$ and $out(s,\sigma_{A''}) \subseteq out(s,\sigma_{A'})$. We know for every $s_i \in out(s,\sigma_A), M, s_i \models C \vee l$ and for every $s_j \in out(s,\sigma_{A'}), M, s_j \models D \vee \neg l$. Therefore, for every $s_k \in out(s,\sigma_{A''}), M, s_k \models (C \vee l)$ and $M, s_k \models (D \vee \neg l)$. By propositional reasoning, $M, s_k \models C \vee D$. Therefore, $M, s \models \langle\langle A \cup A'\rangle\rangle\bigcirc(C \vee D)$.

As s is an arbitrary state reachable from s_0, we obtain $M, s_0 \models \langle\langle\emptyset\rangle\rangle\Box(P \wedge Q \Rightarrow \langle\langle A \cup A'\rangle\rangle\bigcirc(C \vee D))$ from the semantics of $\langle\langle\emptyset\rangle\rangle\Box$. Thus, T_{t+1} is satisfiable and SRES4 is sound.

Next we show SRES5 is sound. Assume $\langle\langle\emptyset\rangle\rangle\Box(P \Rightarrow \langle\langle A\rangle\rangle\bigcirc(C \vee l))$ and $\langle\langle\emptyset\rangle\rangle\Box(Q \Rightarrow [\![A']\!]\bigcirc(D \vee \neg l))$ are in T_t and the coalition A is a subset of the coalition A'. Let T_{t+1} be obtained by an application of SRES5 to $\langle\langle\emptyset\rangle\rangle\Box(P \Rightarrow \langle\langle A\rangle\rangle\bigcirc(C \vee l))$ and $\langle\langle\emptyset\rangle\rangle\Box(Q \Rightarrow [\![A']\!]\bigcirc(D \vee \neg l))$, that is, $T_{t+1} = T_t \cup \{\langle\langle\emptyset\rangle\rangle\Box(P \wedge Q \Rightarrow [\![A' \setminus A]\!]\bigcirc(C \vee D))\}$. We show that the CGM M also satisfies T_{t+1}.

Consider an arbitrary state s reachable from $s_0 \in S$. If either $M, s \not\models P$ or $M, s \not\models Q$, then $M, s \models P \wedge Q \Rightarrow [\![A' \setminus A]\!]\bigcirc(C \vee D)$. If, on the other hand, $M, s \models P$ and $M, s \models Q$, then from $\langle\langle\emptyset\rangle\rangle\Box(P \Rightarrow \langle\langle A\rangle\rangle\bigcirc(C \vee l))$, $\langle\langle\emptyset\rangle\rangle\Box(Q \Rightarrow [\![A']\!]\bigcirc(D \vee \neg l))$, and the semantics of $\langle\langle\emptyset\rangle\rangle\Box$, we obtain that $M, s \models \langle\langle A\rangle\rangle\bigcirc(C \vee l)$ and $M, s \models [\![A']\!]\bigcirc(D \vee \neg l)$. By the semantics of $\langle\langle A\rangle\rangle\bigcirc$, **(1)** there exists an A-move $\sigma_A \in D_A(s)$ such that $M, s' \models C \vee l$ for all states $s' \in out(s,\sigma_A)$. By the definition of $[\![A']\!]\bigcirc$ and the semantics of $\langle\langle A'\rangle\rangle\bigcirc$, we have, **(2)** for every A'-move $\sigma_{A'} \in D_{A'}(s)$, there exists a move vector σ such that $\sigma_{A'} \sqsubseteq \sigma, s'' = \delta(s,\sigma)$ and $M, s'' \models D \vee \neg l$. Let Δ be the set of A'-moves

$\{\sigma^i_{A'} \mid \sigma^i_{A'} \in D_{A'}(s)$ and $\sigma_A \sqsubseteq \sigma^i_{A'}\}$. As $A \subseteq A'$, $\Delta \subseteq D_{A'}(s)$, i.e. every A'-move in Δ is also in $D_{A'}(s)$. Therefore, for every A'-move $\sigma^i_{A'} \in \Delta$, there exists a move vector σ^i such that $\sigma^i_{A'} \sqsubseteq \sigma^i, s^i = \delta(s, \sigma^i), M, s^i \models D \vee \neg l$. By Lemma 5.12, $\sigma_A \sqsubseteq \sigma^i$. By Lemma 5.13, $out(s, \sigma^i) = \{s^i\} \subseteq out(s, \sigma_A)$. From (1), $M, s^i \models C \vee l$. Thus $M, s^i \models C \vee D$ by propositional reasoning. Therefore, **(3)** for every A'-move $\sigma^i_{A'} \in \Delta$, there exists a move vector σ^i such that $\sigma^i_{A'} \sqsubseteq \sigma^i, s^i = \delta(s, \sigma^i), M, s^i \models C \vee D$. (3) is equivalent to that for every $(A' \setminus A)$-move $\sigma_{(A' \setminus A)} \in D_{(A' \setminus A)}(s)$, there exists a move vector σ^i such that (i) $\sigma_A \sqsubseteq \sigma^i, \sigma_{(A' \setminus A)} \sqsubseteq \sigma^i$ and (ii) $s^i = \delta(s, \sigma^i), M, s^i \models C \vee D$. From the definition of $[\![A' \setminus A]\!]\bigcirc$, we obtain $M, s \models [\![A' \setminus A]\!]\bigcirc(C \vee D)$. As s is an arbitrary state reachable from s_0, from the semantics of $\langle\langle\emptyset\rangle\rangle\Box$, we obtain $M, s_0 \models \langle\langle\emptyset\rangle\rangle\Box(P \wedge Q \Rightarrow [\![A' \setminus A]\!]\bigcirc(C \vee D))$. Thus, T_{t+1} is satisfiable and SRES5 is sound.

The soundness proofs for SRES1 to SRES3, SRES6 and SRES7 are analogous to those for SRES4 and SRES5.

Regarding RW1, by the semantics of $\langle\langle A\rangle\rangle\bigcirc$ and **false** the formula $\langle\langle\emptyset\rangle\rangle\Box(\wedge^n_{i=1} m_i \Rightarrow \langle\langle A\rangle\rangle\bigcirc\mathbf{false})$ is true iff $\langle\langle\emptyset\rangle\rangle\Box(\wedge^n_{i=1} m_i \Rightarrow \mathbf{false})$ is true. This formula is equivalent to $\langle\langle\emptyset\rangle\rangle\Box(\vee^n_{i=1} \neg m_i)$ which in turn, by the semantics of $\Rightarrow$ and **true**, is equivalent to $\langle\langle\emptyset\rangle\rangle\Box(\mathbf{true} \Rightarrow \vee^n_{i=1} \neg m_i)$. The proof for RW2 is similar.

So, we have shown that for any derivation $T_0, T_1, \ldots, T_n$, if T_0 is satisfiable, then T_n is satisfiable. Thus, if T_n contains a contradiction, then T_0 is unsatisfiable. $\Box$

Recall from Section 3.5.2 that in the completeness proof for the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL, we make use of labelled behaviour graphs, terminal nodes, and reduced labelled behaviour graphs. In the following, we adapt these notions to XATL.

Definition 5.22 *Agent behaviour graph*
Let σ_A be an A-move, σ^N_A be a negative-A-move, and $C^\ddagger$ be a conjunction of disjunctions of literals (possibly consisting of only one disjunction of literals or one literal). Then an expression of the form $\sigma_A\bigcirc C^\ddagger$ or $\sigma^N_A\bigcirc C^\ddagger$ is called a *next formula* or *negative next formula*, respectively.

Let T be a set of SNF$_{\mathrm{XATL}}$ clauses. We construct a finite labelled directed graph $H = (N, E)$, called an *agent behaviour graph* for T as follows. The set of nodes N of H consists of all valuations of propositions occurring in T.

To define the set of edges E of H, we need the following auxiliary definitions. Let $n = V$ be a node in N. Let $R_\emptyset(n, T) = \{\sigma_\emptyset\bigcirc D \mid \mathbf{true} \Rightarrow D \in T\} \cup \{\sigma_\emptyset\bigcirc C \mid P \Rightarrow \langle\langle\emptyset\rangle\rangle\bigcirc C \in T, V \models P\} \cup \{\sigma_\emptyset\bigcirc\mathbf{true}\}$. Assume $R_\emptyset(n, T) = \{\sigma_\emptyset\bigcirc C_1, \sigma_\emptyset\bigcirc C_2, \ldots, \sigma_\emptyset\bigcirc C_k\}$, then let $X(n, T) = C_1 \wedge C_2 \wedge \ldots \wedge C_k$.

Let $R_m(n, T) = \{\langle\langle A\rangle\rangle\bigcirc C \mid P \Rightarrow \langle\langle A\rangle\rangle\bigcirc C \in T, A \neq \emptyset, V \models P\} \cup \{[\![B]\!]\bigcirc D \mid Q \Rightarrow [\![B]\!]\bigcirc D \in T, B \neq \Sigma, V \models Q\}$. Then let $seq(R_m(n, T))$ be a sequence consisting of all formulae in $R_m(n, T)$ such that all the formulae of the form $\langle\langle A\rangle\rangle\bigcirc C$ precede all the formulae of the form $[\![B]\!]\bigcirc D$. We use $seq(R_m(n, T))[i]$ to denote the ith element of $seq(R_m(n, T))$.

Let $next(n, T)$ be the smallest set of next formulae and negative next formulae such that for every element $seq(R_m(n, T))[i]$ in $seq(R_m(n, T))$,

- if $seq(R_m(n, T))[i]$ is of the form $\langle\langle A\rangle\rangle\bigcirc C$, then the next formula $\sigma_A\bigcirc(C \wedge X(n, T))$ is in the set $next(n, T)$, where σ_A is an A-move such that for every agent $a \in A, \sigma_A(a) = i$ and for every agent $a' \notin A, \sigma_A(a') = *$; and

- if $seq(R_m(n,T))[i]$ is of the form $[\![B]\!]\bigcirc D$, then the negative next formula $\sigma_B^N\bigcirc(D \wedge X(n,T))$ is in the set $next(n,T)$, where σ_B^N is a negative-B-move such that for every agent $a \in B, \sigma_B^N(a) = *$ and for every agent $a' \notin B, \sigma_B^N(a') = -i$.

In the following, $D_1^\ddagger$ and $D_2^\ddagger$ represent a conjunction of disjunctions of literals and we assume that $\wedge$ is idempotent. We inductively define the closure $cls(n,T)$ of the set $next(n,T)$ to be the set such that

- every element of $next(n,T)$ is in $cls(n,T)$;
- if $\sigma_A\bigcirc D_1^\ddagger, \sigma_{A'}\bigcirc D_2^\ddagger \in cls(n,T)$, $A \cap A' = \emptyset$ and neither A nor A' is an empty set, then $\sigma_A \oplus \sigma_{A'}\bigcirc(D_1^\ddagger \wedge D_2^\ddagger) \in cls(n,T)$; and
- if $\sigma_A\bigcirc D_1^\ddagger, \sigma_{A'}^N\bigcirc D_2^\ddagger \in cls(n,T)$, $A \subseteq A'$, A is not an empty set and $A' \neq \Sigma$, then $\sigma_A \oplus \sigma_{A'}^N\bigcirc(D_1^\ddagger \wedge D_2^\ddagger) \in cls(n,T)$.

Given a node $n = V$, for every next formula $\sigma_A\bigcirc C^\ddagger$ or negative next formula $\sigma_A^N\bigcirc C^\ddagger$ in the set $cls(n,T)$, if there exists a node $n' = V'$ such that $V' \models C^\ddagger$, then there exists an edge in E from n to n' labelled by σ_A or σ_A^N, respectively.

Let $R_0(T) = \{D \mid \textbf{start} \Rightarrow D \in T\}$ and $R_g(T) = \{C \mid \textbf{true} \Rightarrow C \in T\}$. Then any node $n_0 = V$, where V satisfies the set $R_0(T) \cup R_g(T)$, is an initial node of H. The *agent behaviour graph* for an set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses T is the set of nodes and edges reachable from the initial nodes.

Definition 5.23 *Terminal node*
A node $n = V$ in the agent behaviour graph for a set T of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses is a *terminal node* iff there exists a next formula $\sigma_A\bigcirc C^\ddagger$ or a negative next formula $\sigma_A^N\bigcirc C^\ddagger$ in the set $cls(n,T)$ such that no edge labelled by σ_A or σ_A^N departs from the node n.

Definition 5.24 *Reduced agent behaviour graph*
Given an agent behaviour graph $H = (N,E)$ for a set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses T, the *reduced labelled behaviour graph* $red(H)$ for T is the result of exhaustively applying the following deletion rule DEL to H.

DEL If $n \in N$ is a terminal node, then delete n and every edge into or out of n.

Outline of the completeness proof

In the following, we present an outline of the completeness proof of the calculus $\mathsf{R}_{\mathrm{XATL}}$.

Lemma 5.14 *Let T be a set of* $\mathrm{SNF}_{\mathrm{XATL}}$ *clauses and let $red(H)$ be its reduced agent behaviour graph. If the graph $red(H)$ is not empty, then T is satisfiable.*

Proof (Outline). This lemma can be proved by showing that if $red(H)$ is not empty, then **(1)** a CGM M can be constructed from $red(H)$ and **(2)** M satisfies T.

We know that all edges in $red(H)$ are labelled with an A-move or a negative-A-move. According to these edges and their labels, we are able to generate corresponding new edges labelled with move vectors. We use $vec(H)$ to denote the new graph formed by the nodes from $red(H)$ and those edges labelled move vectors.

Let n be a node in $vec(H)$. If there exists an edge from n to n' labelled with the move vector σ in $vec(H)$, then we say n' is a σ-successor of n. We use $\delta(n, \sigma)$ to denote the set of all σ-successors of the node n. Then, by eliminating unnecessary edges in $vec(H)$, i.e. ensure that for every node n in $vec(H)$ and for every move vector σ available at n, $|\delta(n, \sigma)| = 1$, we obtain M. Thus, we have (1).

Regarding (2), it can be shown that, by the way we construct M from T, M satisfies T. □

Lemma 5.15 *Let T be a set of* $\mathrm{SNF}_{\mathrm{XATL}}$ *clauses. If T is unsatisfiable, then its reduced agent behaviour graph $red(H)$ is empty.*

Proof. Follows from Lemma 5.14. □

Now we are able to show that if a finite set T of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses is unsatisfiable, then T has a refutation using the step resolution rules SRES1 to SRES7 and the rewrite rules RW1 and RW2.

Let T be an unsatisfiable set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses. The proof proceeds by induction on the sequence of applications of the deletion rule DEL to the agent behaviour graph of T. If the unreduced agent behaviour graph is empty then we can obtain a refutation by applying step resolution rules SRES1 to SRES3. Now suppose the agent behaviour graph H is non-empty. The reduced agent behaviour graph must be empty by Lemma 5.15, so there must be a node that can be deleted from H.

We show that for every application of the deletion rule DEL to a behaviour graph H of T resulting in a smaller graph H'', there is a derivation from T by $\mathsf{R}_{\mathrm{XATL}}$ resulting in a set T' such that the behaviour graph H' for T' is a strict subgraph of H''. In particular, we can prove that the deletion rule DEL corresponds to a series of step resolution inferences by SRES1 to SRES7 deriving a clause of the form $\mathbf{true} \Rightarrow \mathbf{false}$, $P \Rightarrow \langle\langle A\rangle\rangle\bigcirc\mathbf{false}$ or $P \Rightarrow [\![A']\!]\bigcirc\mathbf{false}$. The rewrite rules RW1 and RW2 will replace $P \Rightarrow \langle\langle A\rangle\rangle\bigcirc\mathbf{false}$ and $P \Rightarrow [\![A']\!]\bigcirc\mathbf{false}$ by the simpler clauses $\mathbf{true} \Rightarrow \neg P$.

Therefore, if T is unsatisfiable, then the reduced agent behaviour graph $red(H)$ for T is empty and the sequence of applications of the deletion rule DEL which reduces the agent behaviour graph H for T to an empty $red(H)$ can be used to construct a refutation of T by $\mathsf{R}_{\mathrm{XATL}}$.

5.6 Conclusions

ATL [7] is a well-known and successful formalism for specification and verification of multi-agent systems. As the significance of multi-agent systems grows, the research on ATL is increasingly important and active. Our research focuses on a fragment of ATL, namely XATL, which is also useful for many applications, for example, specification and verification of a variety of strategic games [57, 58]. Currently there are a number of methods that have been established for tackling the satisfiability problem for ATL and XATL, for instance, two tableau-based methods by Walther et al. [73] and by Goranko et al. [41] for ATL, two automata-based methods by van Drimmelen [70] and by Goranko et al. [42] for ATL, and one tableau-based method[2] by Hansen [44] for XATL. No resolution-based method has been developed for ATL or any fragment of ATL so far.

[2]This tableau method is originally developed for CL. Since the equivalence of the semantics for XATL and CL has been shown in [39], it is easy to show that this tableau method for CL translates into a tableau method for XATL.

In this chapter, we have developed the first clausal resolution calculus $\mathsf{R}_{\mathrm{XATL}}$ for XATL. We define a normal form $\mathrm{SNF}_{\mathrm{XATL}}$ and a transformation procedure, which can be used to transform an XATL formula into a satisfiability equivalent set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses. We have shown that the transformation procedure preserves satisfiability and terminates, and that the computational complexity of this procedure is polynomially bounded with respect to the size of the input formula. The calculus $\mathsf{R}_{\mathrm{XATL}}$ consists of seven step resolution rules and two rewrite rules and we provide a number of examples illustrating deriving a contradiction from an unsatisfiable set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses by those rules. Moreover, we have shown that $\mathsf{R}_{\mathrm{XATL}}$ is sound and outlined a completeness proof for $\mathsf{R}_{\mathrm{XATL}}$ using agent behaviour graphs, terminal nodes and reduced agent behaviour graphs.

This approach has been successfully applied to PLTL and CTL to prove the completeness of the resolution calculus for PLTL in [38] and the resolution calculus for CTL in Section 3, respectively. Here we have extended this approach to XATL. In future work, it is our intention to reuse the techniques we developed for $\mathsf{R}_{\mathrm{XATL}}$ in order to develop a calculus for full ATL.

Chapter 6

Conclusions and future work

In this chapter, we summarise our work on tackling the satisfiability problems for CTL and XATL by using resolution-based approaches and draw conclusions. We also outline the directions of future work including more refinements on the calculi we have developed in this thesis, the possibilities of applying resolution-based methods to other forms of modal logics, and the potential of using first-order resolution to emulate those methods.

6.1 Conclusions

In this thesis, we have considered the propositional branching-time temporal logic CTL and the agent logic XATL and have developed corresponding resolution-based calculi, which can be used as bases of decision procedures for the satisfiability problems for both CTL and XATL.

In the case of CTL, we have provided a refined resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ and a new approach to automating the inferences of CTL resolution of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ using first-order techniques, and, consequently, have obtained a theorem prover for CTL, namely CTL-RP, which can be used to efficiently verify various problems encoded into CTL formulae.

The work we have done on the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is motivated by Fisher's calculus [37] for PLTL and Bolotov's calculus [15] for CTL. In 1991, Fisher first proposed a clausal resolution calculus for PLTL, which provides many important ideas and techniques such as the Separated Normal Form transformation, step resolution, eventuality resolution, etc. Later, these ideas and techniques were adapted to a number of other non-classical logics for their corresponding resolution methods. One of them is Bolotov's calculus [15], where he extended Fisher's calculus from PLTL to CTL. He proposed the Separated Normal Form for CTL incorporating indices to denote existentially quantified paths. He also identified different types of eventualities, $\mathbf{A}\Diamond$ and $\mathbf{E}\Diamond$, and developed the corresponding eventuality resolution rules for them. Based on Bolotov's work [15], we have developed a new clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ for CTL. The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ contains step resolution rules, eventuality resolution rules, and rewrite rules. We have shown that $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is sound and complete, i.e. for a given CTL formula φ, φ is unsatisfiable iff by applying step resolution rules, eventuality resolution rules and rewrite rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ to the set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses representing φ, a contradiction can be derived. Compared with [15], we have the following advantages.

1. We introduce a new type of clause, *global clauses*, into the normal form. By doing so, we can often reduce the number of clauses generated from the transformation as well as those derived from resolution inferences. We also provide a formal semantics for normal form formulae.

2. We develop a new and improved transformation procedure. In experiments, we observe that for checking the satisfiability of a CTL formula, the CPU time consumed by our prover CTL-RP using the improved transformation procedure is usually less than half of the CPU time used by CTL-RP using our implementation of Bolotov's transformation procedure for the same CTL formula.

3. We show the termination of the transformation procedure and determine its computational complexity, whereas in [15] the corresponding proofs are absent.

4. The applications of the step resolution rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ are subject to an ordering and a selection function. Therefore, the applicability of the resolution rules is reduced compared to Bolotov's resolution rules. Thus, the search space is pruned noticeably.

5. We present a new completeness proof for $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, which also shows that two out of the four eventuality resolution rules in [15] are redundant. Consequently, $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ has fewer eventuality rules than Bolotov's calculus for CTL.

6. A complexity analysis for $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is given, which is absent from [15].

Inspired by Dixon's work on using a propositional/first-order resolution prover to implement step resolution rules [28] for the calculus $\mathsf{R}_{\mathrm{PLTL}}$ and Hustadt and Konev's work on the efficient PLTL theorem prover TRP++ [48], we have developed an approach for using first-order clauses to represent determinate clauses and then, establish a mapping, which maps resolution rules SRES1 to SRES8 to a first-order resolution rule. Therefore, all inferences of step resolution can be done by a first-order resolution theorem prover. Finally, we extend the high performance first-order theorem prover SPASS with an implementation of the eventuality resolution rules to realise the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ in the theorem prover CTL-RP and according to our empirical study of CTL-RP, it shows good performance.

In the case of XATL, we have developed a resolution calculus $\mathsf{R}_{\mathrm{XATL}}$. A number of tableau-based [44, 73, 41] and automata-based methods [70, 42] have been developed for the satisfiability problem for ATL or XATL in recent years, however no attempts have been made to develop resolution-based methods. Thus, $\mathsf{R}_{\mathrm{XATL}}$ is the first resolution-based method for XATL satisfiability checking.

We define a normal form $\mathrm{SNF}_{\mathrm{XATL}}$ and a transformation procedure to transform an arbitrary XATL formula into this normal form. We prove the correctness of the procedure and analyse its complexity. $\mathsf{R}_{\mathrm{XATL}}$ consists of seven step resolution rules and two rewrite rules. We show $\mathsf{R}_{\mathrm{XATL}}$ is sound, i.e. if there is a refutation by $\mathsf{R}_{\mathrm{XATL}}$ from a set of $\mathrm{SNF}_{\mathrm{XATL}}$ clauses representing an XATL formula φ, then φ is unsatisfiable. We also provide an outline of the completeness proof of $\mathsf{R}_{\mathrm{XATL}}$.

6.2 Future work

6.2.1 Further research on XATL resolution

In this thesis, we have presented the resolution calculus $\mathsf{R}_{\mathrm{XATL}}$ for XATL and outlined its completeness proof, but there still are a number of objectives regarding $\mathsf{R}_{\mathrm{XATL}}$ that we have not achieved yet. We discuss them individually in the following.

Firstly, since the completeness issues are important for a calculus, we should consider the work on those issues further and also provide a formal completeness proof for $\mathsf{R}_{\mathrm{XATL}}$.

Secondly, we are interested in realising the calculus $\mathsf{R}_{\mathrm{XATL}}$ in a theorem prover for XATL. We may be able to apply our implementation approach for the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ to $\mathsf{R}_{\mathrm{XATL}}$. To do so, several issues need to be investigated.

1. We have to find an approach for representing $\mathrm{SNF}_{\mathrm{XATL}}$ clauses using first-order clauses. For this issue, we have already developed some ideas. For example, for a given set of agents $\Sigma = \{1, 2, 3\}$, the XATL formula $\varphi = r \Rightarrow \langle\!\langle 1, 2\rangle\!\rangle \bigcirc (p \vee q)$ may be represented by the first-order clause

$$\varphi_f = \neg Q_r(x) \vee Q_p(\mathsf{app}(m(1, 1, y), x)) \vee Q_q((\mathsf{app}(m(1, 1, y), x)))$$

and the XATL formula $\psi = q \Rightarrow \langle\!\langle 3\rangle\!\rangle \bigcirc \neg p$ may be represented by

$$\psi_f = \neg Q_q(w) \vee \neg Q_p(\mathsf{app}(m(u, v, 1), w)).$$

Here m is a $|\Sigma|$-ary function symbol, $m(1, 1, y)$ represents the $\{1, 2\}$-move $\langle 1, 1, *\rangle$ indicating that agents 1 and 2 each play move 1 while agent 3 can play any move available to it and $m(u, v, 1)$ represents the $\{3\}$-move $\langle *, *, 1\rangle$ indicating that agent 3 plays move 1 while agents 1 and 2 can play any move available to them. The rest of the notation in φ_f and ψ_f is analogous to the notation we define in Section 4.2.2.

2. We need to utilise first-order resolution techniques to emulate the step resolution rules in $\mathsf{R}_{\mathrm{XATL}}$ to conduct inferences for XATL. For this issue, it seems that it can be naturally solved due to the transformation we proposed above. For example, take the following application of the step resolution rule SRES4 to the $\mathrm{SNF}_{\mathrm{XATL}}$ clauses $\varphi = r \Rightarrow \langle\!\langle 1, 2\rangle\!\rangle \bigcirc (p \vee q)$ and $\psi = q \Rightarrow \langle\!\langle 3\rangle\!\rangle \bigcirc \neg p$, where $\{1, 2\} \cap \{3\} = \emptyset$:

$$\frac{\begin{array}{c} r \Rightarrow \langle\!\langle 1, 2\rangle\!\rangle \bigcirc (p \vee q) \\ q \Rightarrow \langle\!\langle 3\rangle\!\rangle \bigcirc \neg p \end{array}}{r \wedge q \Rightarrow \langle\!\langle 1, 2, 3\rangle\!\rangle \bigcirc q}$$

The first-order clauses φ_f and ψ_f representing the $\mathrm{SNF}_{\mathrm{XATL}}$ clauses φ and ψ, respectively, are

$\varphi_f = \neg Q_r(x) \vee Q_p(\mathsf{app}(m(1, 1, y), x)) \vee Q_q(\mathsf{app}(m(1, 1, y), x))$ and
$\psi_f = \neg Q_q(w) \vee \neg Q_p(\mathsf{app}(m(u, v, 1), w))$

which can be resolved using first-order resolution:

$$\begin{array}{l}\neg Q_r(x) \vee Q_p(\mathsf{app}(m(1,1,y),x)) \vee Q_q(\mathsf{app}(m(1,1,y),x)) \\ \neg Q_q(w) \vee \neg Q_p(\mathsf{app}(m(u,v,1),w)) \\ \hline \neg Q_r(x) \vee \neg Q_q(x) \vee Q_q(\mathsf{app}(m(1,1,1),x))\end{array}$$

As we can see, the resolvent $\neg Q_r(x) \vee \neg Q_q(x) \vee Q_q(\mathsf{app}(m(1,1,1),x))$ represents the SNF_{XATL} clause $r \wedge q \Rightarrow \langle\!\langle 1,2,3 \rangle\!\rangle \bigcirc q$ that we obtain as the result of the application of SRES4. Thus, it may be possible to emulate all step resolution rules in R_{XATL} by using first-order resolution.

3. Last but not least, we need to find an appropriate efficient first-order theorem prover, for example SPASS [54] or E [65], and extend the prover with an implementation of the transformation procedure for XATL to realise the calculus R_{XATL}.

Thirdly, we intend to extend our calculus R_{XATL} to ATL. We believe that all techniques we have developed for R_{XATL} can be adapted to ATL. It would remain to develop rules and procedure for handling eventualities in ATL to achieve a clausal resolution calculus for ATL.

6.2.2 Extension to other modal logics

We believe that it is possible to utilise the calculi $\mathsf{R}^{\succ,S}_{\text{CTL}}$ and R_{XATL} as bases for developing resolution calculi for other modal logics, in particular Alternating-time Temporal Epistemic Logic [69], Propositional Dynamic Logic [45], and CTL* [35].

Extension to ATEL

ATL can be augmented with *knowledge modalities* to deal with agents' cooperation, knowledge and time and the resulting language is Alternating-time Temporal Epistemic Logic (ATEL) [69]. ATEL has been useful for the specification and verification of many game-like multi-agent systems. As the significance of multi-agent systems has grown rapidly in recent years, proof methods for ATEL are desirable. Fisher's resolution method R_{PLTL} [37] has been extended to deal with linear-time and knowledge in [30] and Bolotov's resolution method [15] has been adapted to handle branching-time and knowledge in [29]. We believe that if we can successfully extend our calculus R_{XATL} to ATL, then it is feasible to further develop the calculus to ATEL with some reasonable effort.

It should be noted that in [40] Goranko shows that ATEL can be reduced to ATL. That is, there exists a translation of models and formulae of ATEL into ATL that preserves satisfiability. While the translation preserves satisfiability, it may not preserve other properties of ATEL formulae. Therefore, a genuine ATEL calculus may still be of use. For example, for the model checking problem of ATL and ATEL, there are systems for both, namely MCMAS [61] for ATEL and MOCHA [8] for ATL. A comparison between the performance of MCMAS on ATEL problems and the performance of MOCHA [8] on the translation of these problems shows some encouraging results for MCMAS. Therefore, we believe that it is also worth developing a resolution theorem prover to address the satisfiability problem for ATEL directly.

Extension to PDL

Propositional Dynamic Logic (PDL) [45] is a of modal logic, where the modal operators $\Box$ and $\Diamond$ are indexed with expressions over a regular language of *actions*. For instance, the PDL formula $[\mathit{shoot}]\mathit{empty}$ expresses that shooting necessarily results that the chamber of a shotgun is empty, on the other hand, the formula $\langle \mathit{shoot}\rangle \mathit{dead}$ specifies that shooting possibly results that the prey is dead. Its underlying models involve (i) a set of states, which are labelled with atomic propositions indicating that such propositions hold on such states, (ii) a set of transitions between states, which are labelled with atomic actions.

The model structures for PDL are similar to the model structures for CTL. In addition, the satisfiability problem of PDL, which is known to be EXPTIME [43], has the same complexity as the satisfiability problem of CTL. Due to the similarities between PDL and CTL, we would like to investigate the possibility of developing a clausal resolution calculus for PDL, which we expect to be similar to the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ presented in Chapter 3, i.e. a suitable normal form for PDL, a transformation procedure for transforming a PDL formula into its normal form, and a series of resolution style rules for conducting inferences. Then an arbitrary PDL formula φ can be transformed into a satisfiability equivalent set T of formulae in normal form and a refutation can be derived by applying those resolution-like rules to T iff the original formula φ is unsatisfiable.

Extension to CTL*

CTL* was first proposed by Emerson et al. in 1986 and is often referred to as full branching-time temporal logic, as it is the most expressive language in that category. CTL* extends CTL by removing the syntactic constraint that path quantifiers must always be paired with a temporal operator and vice versa, i.e. CTL* allows the path quantifiers followed by an arbitrary linear-time formula, which can be boolean combination and nesting over temporal operators $\Diamond$, $\Box$, $\bigcirc$, $\mathcal{U}$ and $\mathcal{W}$; and some subformulae without path quantifiers. For example, $\mathbf{A}\Diamond\Box p$ and $\mathbf{E}(\bigcirc p\,\mathcal{U}\,\Diamond q)$ are well-formed formulae of CTL* but not formulae of CTL.

Since CTL* is an extension of CTL, it may be possible to extend our calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ in Chapter 3 to CTL*. However the stronger expressive power of CTL* comes at a cost, as the complexity of the satisfiability problem for CTL* is 2EXPTIME [31] instead of EXPTIME for CTL [22]. Due to the high complexity of CTL* formulae, extending our calculus from CTL to CTL*, we can anticipate that finding a suitable normal form for CTL* and developing a transformation procedure preserving satisfiability are very challenging. For example, if the complexity of the transformation procedure reaches EXPTIME or 2EXPTIME, we have to make sure that it is not the case that for every CTL* formula the size of the formula grows exponentially after the transformation procedure is applied to it, in particular, for CTL formulae, the transformation should still be polynomial. Otherwise, even if we obtain a decision procedure for the satisfiability problem of CTL*, this decision procedure would not be very practical.

In [17], Bolotov et al. made the first attempt to develop a clausal resolution calculus for CTL*, however they also pointed out in [15] that this calculus for CTL* is incorrect. Consequently, a sound and complete resolution system for CTL* is still an open area of research.

Appendix A

The previous transformation rules

In [15], Bolotov provides a set of transformation rules to transform an arbitrary CTL formula into an equi-satisfiable set of SNF_{CTL} clauses. The transformation rules defined below are identical to those in [15] but modified to allow for global clauses. Moreover the transformation here is implemented in the CTL resolution theorem prover CTL-RP version 00.09. In addition, we also provide a set of more efficient transform rules for an arbitrary CTL formula in this thesis and compare these two set of transformation rules in Section 3.7 and 4.4.

The definition of the rule set *Trans*:

- Index introduction rules:

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

  ind is a new index.

- Boolean rules:

$$\mathit{Trans}(4) \quad q \Rightarrow \varphi_1 \wedge \varphi_2 \longrightarrow \left\{ \begin{array}{l} q \Rightarrow \varphi_1 \\ q \Rightarrow \varphi_2 \end{array} \right.$$

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

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

- Temporal operator rules:

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

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

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

$$\mathbf{X}* \in \{\mathbf{A}\Box, \mathbf{A}\Diamond, \mathbf{E}_{\langle ind \rangle}\Box, \mathbf{E}_{\langle ind \rangle}\Diamond\}$$

Trans(9) $$q \Rightarrow \mathbf{X}(\varphi_1 * \varphi_2) \longrightarrow \begin{cases} q \Rightarrow \mathbf{X}(p * \varphi_2) \\ p \Rightarrow \varphi_1 \end{cases}$$ if φ_1 is not a literal.

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

Trans(10) $$q \Rightarrow \mathbf{X}(\varphi_1 * \varphi_2) \longrightarrow \begin{cases} q \Rightarrow \mathbf{X}(\varphi_1 * p) \\ p \Rightarrow \varphi_2 \end{cases}$$ if φ_2 is not a literal.

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

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

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

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

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

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

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

Appendix B

Equivalences in ATL

The following equivalences between ATL formulae are shown either in [7] or in [42].

- $[\![A]\!]\bigcirc\varphi \equiv \neg\langle\!\langle A\rangle\!\rangle\bigcirc\neg\varphi$
- $[\![A]\!]\Box\varphi \equiv \neg\langle\!\langle A\rangle\!\rangle\Diamond\neg\varphi$
- $[\![A]\!]\Diamond\varphi \equiv \neg\langle\!\langle A\rangle\!\rangle\Box\neg\varphi$
- $\langle\!\langle A\rangle\!\rangle\bigcirc\varphi \Rightarrow [\![\Sigma \setminus A]\!]\bigcirc\varphi$ is true, however $[\![A]\!]\bigcirc\varphi \Rightarrow \langle\!\langle\Sigma \setminus A\rangle\!\rangle\bigcirc\varphi$ is not necessarily true.
- $\langle\!\langle A\rangle\!\rangle\Box\varphi \equiv \varphi \wedge \langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\Box\varphi$
- $\neg\langle\!\langle A\rangle\!\rangle\Box\varphi \equiv \neg\varphi \vee \neg\langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\Box\varphi$
- $\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\,\varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\,\varphi_2)$
- $\neg\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\,\varphi_2 \equiv \neg\varphi_2 \wedge (\neg\varphi_1 \vee \neg\langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\,\varphi_2)$
- $\langle\!\langle A\rangle\!\rangle\bigcirc\varphi_1 \wedge \langle\!\langle A'\rangle\!\rangle\bigcirc\varphi_2 \equiv \langle\!\langle A \cup A'\rangle\!\rangle\bigcirc(\varphi_1 \wedge \varphi_2)$ for disjoint A and A'.
- $\langle\!\langle\emptyset\rangle\!\rangle\Box(\theta \Rightarrow (\varphi \wedge \langle\!\langle A\rangle\!\rangle\bigcirc\theta)) \Rightarrow \langle\!\langle\emptyset\rangle\!\rangle\Box(\theta \Rightarrow \langle\!\langle A\rangle\!\rangle\Box\varphi)$
- $\langle\!\langle\emptyset\rangle\!\rangle\Box((\varphi_2 \vee (\varphi_1 \wedge \langle\!\langle A\rangle\!\rangle\bigcirc\theta)) \Rightarrow \theta) \Rightarrow \langle\!\langle\emptyset\rangle\!\rangle\Box(\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\,\varphi_2 \Rightarrow \theta)$

Bibliography

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

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

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

[4] L. Afanasiev, M. Franceschet, M. Marx, and M. de Rijke. CTL Model Checking for Processing Simple XPath Queries. In *Proc. TIME'04*, pages 117–124. IEEE Comp. Soc. Press, 2004.

[5] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. FOCS'97*, page 100. IEEE Computer Society, 1997.

[6] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. COMPOS'97*, pages 23–60. Springer, 1998.

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

[8] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *Proc. CAV'98*, pages 521–525. Springer, 1998.

[9] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification language. In *Proc. TACAS'02*, pages 296–211. Springer, 2002.

[10] P. C. Attie. On the implementation complexity of specifications of concurrent programs. In *Proc. DISC'03*, volume 2848, pages 151–165. Springer, 2003.

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

[12] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Commun. ACM*, 12(5):260–261, 1969.

[13] A. Basso and A. Bolotov. Towards GCM Re-configuration Extending Specification by Norms. In *CoreGRID Workshop 2007*. CoreGrid Technical Report TR-0080, 2007.

[14] I. Beer, S. Ben-David, C. Eisner, D. Geist, L. Gluhovsky, T. Heyman, A. Landver, P. Paanah, Y. Rodeh, G. Ronin, and Y. Wolfsthal. RuleBase: Model Checking at IBM. In *Proc. CAV'97*, pages 480–483. Springer, 1997.

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

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

[17] A. Bolotov, C. Dixon, and M. Fisher. Clausal resolution for CTL*. In *Proc. MFCS'99*, pages 137–148. Springer, 1999.

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

[19] G. Boole. *An Investigation of The Law of Thought.* Macmillan, 1854.

[20] H. K. Buning and T. Letterman. *Propositional Logic: Deduction and Algorithms.* Cambridge University Press, 1999.

[21] 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.

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

[23] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

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

[25] C. Dixon. *Strategies for Temporal Resolution.* PhD thesis, University of Manchester, 1995.

[26] C. Dixon. Search strategies for resolution in temporal logics. In *Proc. CADE-13*, pages 673–687. Springer, 1996.

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

[28] C. Dixon. Using Otter for Temporal Resolution. In *In Advances in Temporal Logic*, pages 149–166. Kluwer, 2000.

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

[30] C. Dixon, M. Fisher, and M. Wooldridge. Resolution for temporal logics of knowledge. *Journal of Logic and Computation*, 8:345–372, 1998.

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

[32] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[33] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proc. STOC '82*, pages 169–180. ACM Press, 1982.

[34] 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.

[35] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

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

[37] M. Fisher. A resolution method for temporal logic. In *Proc. IJCAI'91*, pages 99–104. Morgan Kaufmann, 1991.

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

[39] V. Goranko. Coalition games and alternating temporal logics. In *Proc. TARK '01*, pages 259–272. Morgan Kaufmann, 2001.

[40] V. Goranko and W. Jamroga. Comparing semantics of logics for multi-agent systems. *Synthese*, 139:241–280, 2004.

[41] V. Goranko and D. Shkatov. Tableau-based decision procedures for logics of strategic ability in multiagent systems. *ACM Trans. Comput. Logic*, 11(1):1–51, 2009.

[42] V. Goranko and G. van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theor. Comput. Sci.*, 353(1):93–117, 2006.

[43] R. Goré and F. Widmann. An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In *Proc. CADE-22*, pages 437–452. Springer, 2009.

[44] H. H. Hansen. Tableau game for Coalition Logic and Alternating-time Temporal Logic. Master's thesis, University of Amsterdam, The Netherlands, 2004.

[45] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[46] G. Holzmann. *Spin model checker, the: primer and reference manual.* Addison-Wesley, 2003.

[47] U. Hustadt. *Resolution-Based Decision Procedures for Subclasses of First-Order Logic.* PhD thesis, Universität des Saarlandes, 1999.

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

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

[50] G. Jaeger, P. Balsiger, A. Heuerding, S. Schwendimann, M. Bianchi, K. Guggisberg, G. Janssen, W. Heinle, F. Achermann, A. D. Boroumand, P. Brambilla, I. Bucher, and H. Zimmermann. LWB–The Logics Workbench 1.1. `http://www.lwb.unibe.ch/`.

[51] A. Leitsch. *The Resolution Calculus*. Springer, 1997.

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

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

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

[55] W. McCune. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`.

[56] C. Nalon and C. Dixon. Clausal resolution for normal modal logics. *Journal of Algorithms*, 62(3-4):117–134, 2007.

[57] M. Pauly. *Logic for Social Software*. PhD thesis, University of Amsterdam, The Netherlands, 2001.

[58] M. Pauly. A Modal Logic for Coalitional Power in Games. *Journal of Logic and Computation*, 12(1):149–166, 2002.

[59] A. Pnueli. The temporal logic of programs. In *Proc. SFCS '77*, pages 46–57. IEEE Computer Society, 1977.

[60] V. R. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20(2):231–254, 1980.

[61] F. Raimondi and A. Lomuscio. Model Checking ATL and its epistemic extensions. Technical Report RN/05/01, University College London, 2005.

[62] M. Reynolds. Towards a CTL* Tableau. In *Proc. FSTTCS'05*, volume 3821 of *LNCS*, pages 384–395. Springer, 2005.

[63] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965.

[64] K. Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms*. Springer, 2004.

[65] S. Schulz. Theorem prover E. `http://www4.informatik.tu-muenchen.de/~schulz/`.

[66] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theor. Comput. Sci.*, 49(2-3):217–237, 1987.

[67] R. Socher-Ambrosius and P. Johann. *Deduction Systems*. Springer, 1999.

[68] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass. *Temporal databases: theory, design, and implementation*. Benjamin-Cummings, 1993.

[69] W. van der Hoek and M. J. W. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

[70] G. van Drimmelen. Satisfiability in alternating-time temporal logic. In *Proc. LICS '03*, pages 208–217. IEEE Computer Society, 2003.

[71] M. Y. Vardi and P. Wolper. Automata-Theoretic Techniques for Modal Logics of Programs. *J. Comput. Syst. Sci.*, 32(2):183–221, 1986.

[72] A. Voronkov. Vampire. `http://www.vprover.org/index.cgi`.

[73] D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed exptime-complete. *J. Log. Comput.*, 16(6):765–787, 2006.

[74] 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.

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

[76] M. J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley, 2001.

[77] L. Zhang, U. Hustadt, and C. Dixon. First-order Resolution for CTL. Technical Report ULCS-08-010, Department of Computer Science, University of Liverpool, 2008.

[78] L. Zhang, U. Hustadt, and C. Dixon. A Refined Resolution Calculus for CTL. In *Proc. CADE-22*, pages 245–260. Springer, 2009.

[79] L. Zhang, U. Hustadt, and C. Dixon. CTL-RP: A computation tree logic resolution prover. *AI Commun.*, 23(2-3):111–136, 2010.

List of Figures

Index