Model Checking and Theorem Proving

Kailiang Ji

UNIVERSITE PARIS DIDEROT-PARIS 7

SORBONNE PARIS CITE

ECOLE DOCTORALE DE
SCIENCES MATHEMATIQUES DE PARIS CENTRE

DOCTORAT

Discipline: INFORMATIQUE

Kailiang JI

MODEL CHECKING AND THEOREM PROVING
(Le model checking et la démonstration de théorèmes)

Thèse dirigée par Gilles Dowek

JURY
25/09/2015

Sylvain CONCHON	Rapporteur
Gilles DOWEK	Directour de thèse
Ying JIANG	Examinateur
François LAROUSSINIE	Président du jury
Philippe SCHNOEBELEN	Rapporteur

Declaration of Authorship

I, Kailiang Ji, declare that this thesis titled, 'Model Checking and Theorem Proving' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Abstract

Model checking is a technique for automatically verifying correctness properties of finite systems. Normally, model checking tools enjoy two remarkable features: they are fully automatic and a counterexample will be produced if the system fails to satisfy the property. Deduction Modulo is a reformulation of Predicate Logic where some axioms—possibly all—are replaced by rewrite rules. The focus of this dissertation is to give an encoding of temporal properties expressed in CTL as first-order formulas, by translating the logical equivalence between temporal operators into rewrite rules. This way, proof-search algorithms designed for Deduction Modulo, such as Resolution Modulo or Tableaux Modulo, can be used to verify temporal properties of finite transition systems.

To achieve the aim of solving model checking problems with an off-the-shelf automated theorem prover, three works are included in this dissertation. First, we address the graph traversal problems in model checking with automated theorem provers. As a preparation work, we propose a way of encoding a graph as a formula such that the traversal of the graph corresponds to resolution steps. Then we present the way of translating model checking problems as proving first-order formulas in Deduction Modulo. The soundness and completeness of our method shows that solving CTL model checking problems with automated theorem provers is feasible. At last, based on the theoretical basis in the second work, we propose a symbolic model checking method. This method is implemented in iProver Modulo, which is a first-order theorem prover uses Polarized Resolution Modulo.

Abstrait

Le model checking est une technique de vérification automatique de propriétés de correction de systèmes finis. Normalement, les outils de model checking ont deux caractéristiques remarquables: ils sont automatisés et ils produisent un contre-exemple si le systéme ne satisfait pas la propriété. La Déduction Modulo est une reformulation de la logique des prédicats où certains axiomes—possiblement tous—sont remplacés par des régles de réécriture. Le but de cette dissertation est de donner un encodage de propriétés temporelles exprimées en CTL en des formules du premier ordre, en exprimant l'équivalence logique entre les opérateurs temporels avec des règles de réécriture. De cette manière, les algorithmes de recherche de preuve conçus pour la Déduction Modulo, tels que la Résolution Modulo ou les Tableaux Modulo, peuvent être utilisés pour vérifier des propriétés temporelles de systèmes de transition finis.

Afin d'accomplir le but de résoudre des problèmes de model checking avec un prouveur automatique quelconque, trois travaux sont inclus dans cette dissertation. Premièrement, nous abordons le problème de parcours de graphes en model checking avec des prouveurs automatiques. Nous proposons une façon d'encoder un graphe en tant que formule de manière à ce que le parcours du graphe correspond aux etapes de résolution. Nous présentons ensuite comment formuler les problèmes de model checking comme des formules du premier ordre en Déduction Modulo. La correction et la complétude de notre méthode montre que résoudre des problèmes de model checking CTL avec des prouveurs automatiques est faisable. Enfin, en nous appuyant sur la base théorique du deuxième travail, nous proposons une méthode de model checking symbolique. Cette méthode est implantée dans iProver Modulo, qui est un prouveur automatique du premier ordre qui utilise la Résolution Modulo Polarisée.

Acknowledgements

At the end of my thesis, I would like to thank all those people, without whom this thesis would not be possible.

Firstly, I would like to express my sincere gratitude to my advisor Prof. Gilles Dowek, for the continuous support of my PhD study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

My sincerely thanks also goes to Prof. Ying Jiang, without whom I could not come to Paris. You helped me so much both in my study and life. Your attitude of doing research is a benchmark of my future career. I still remember those days after my graduation from ISCAS, you kept on supporting my staying in Beijing.

I would like to thank my colledges: Guillaume Burel, Ali Assaf, Raphaël Cauderlier, Simon Cruanes, Frédéric Gilbert, Ronan Sailard, Pierre Halmagrand. Thank you all for the technical assistance to my project and dealing with the documents related to my living in Paris. It is really a nice experience to work with you!

I am thankful to Chunhui & Yanguo for giving me the home feeling during my stay in Paris. You did give me the parent-like support and generous care. Haijing & Hanbing, my beloved roommates, the time to live together with you was happy and you guys do helped me a lot.

Last but not the least, I am grateful to my family: my parents and my brothers for supporting me spiritually throughout my life in general. I dedicate this thesis to my family.

Contents

List of Figures

List of Tables

Abbreviations

CTL	**C**omputation **T**ree **L**ogic
PRM	**P**olarized **R**esolution **M**odulo
OPRM	**O**rdered **P**olarized **R**esolution **M**odulo
OBDD	**O**rdered **B**inary **D**ecision **D**iagram
PSR	**P**ath **S**ubsumption **R**ule
Lsr-paths	Paths with the **L**ast **S**tate **R**epeated
$\mathcal{R}_{CTL}$	**R**ewrite Rules of **CTL** Connectives

Symbols

$\mathsf{len}(l)$	the length of a finite path l
π	an infinite path
$\pi(s)$	an infinite path starting from s
π_i	the i-th state of π
π_i^j	the finite sub-path $\pi_i, \pi_{i+1}, ..., \pi_j$ of π
ρ	a lsr-path
$\rho(s)$	a lsr-path starting from s
ρ_i	the i-th state of ρ $(i < \mathsf{len}(\rho))$
ρ_i^j	the finite sub-path $\rho_i, \rho_{i+1}, ..., \rho_j$ of ρ

1 Introduction

1.1 Motivation

Model checking [CGP99] is a technique for automatically verifying correctness properties of finite-state systems. Normally, the model checking tools enjoy two remarkable properties: fully automatic and counterexample will be produced if the system falsifies the property. Deduction Modulo [DHK03] is a reformulation of Predicate logic where a theory is represented by a set of rewrite rules, to support automatic and interactive proof search. In this dissertation, a strategy of solving model checking problems with automated theorem proving tools is presented. In this strategy, the logical equivalence between temporal formulas are represented by proposition rewrite rules and the model checking problems are translated into proving first-order formulas in Deduction Modulo.

The first motivation of this dissertation is to express complicated verification problems succinctly. The idea of translating temporal logic into another framework, for instance (quantified) boolean formulas [BCCZ99, McM02, Zha09, Zha14], higher-order logic [Amj04, RSS95], etc., is not new. But using rewrite rules permits to avoid the explosion of the size of formulas during translation, because the rewrite rules can be used on demand to unfold defined symbols.

The second motivation is to solve model checking problems with some off-the-shelf automated theorem provers. If the translation of model checking problems is provable in Deduction Modulo, then the proof-search algorithms designed for Deduction Modulo, such as Resolution Modulo [Dow10] or Tableaux Modulo [DDG+13], can be used to build proofs for the temporal properties of the finite-state systems.

The last motivation is to combine the advantages of the two kinds of formal verification methods. Some complex properties which cannot be expressed by temporal formulas may also provable in automated theorem provers.

1.2 Related work

1.2.1 Bounded Model Checking

Bounded Model Checking was first proposed by Biere et al in 1999 [BCCZ99]. Initially, it was designed for linear-time temporal logic (LTL) with existential interpretation [BCCZ99] and ECTL (the existential fragment of Computation Tree Logic) [PWZ02]. Zhang extended the bounded model checking framework to CTL in 2009 [Zha09] and recently, QBF-based bounded model checking method for Extended Computation Tree Logic (eCTL), which extends CTL with possibility to express simple fairness constraints, was presented in [Zha14].

For LTL or ACTL, the basic idea of bounded model checking is to consider only a finite prefix of a path, which may be a counterexample of the transition systems. The length of the prefix is bounded by some integer k. If no counterexample is found then increase k until a counterexample is found out or the upper bound of k is reached. The infinite path is represented by a finite path, the last state of which has a successor in the previous states of the path. For branching-temporal logics, the length of all the paths starting from a state are bounded to the same integer k.

The technique of bounded model checking does not solve the complexity problem of model checking, because complexity of the procedure for testing the satisfiability of the temporal properties is still exponential. But experiments showed that in many cases, bounded model checking performs better than BDD-based techniques [McM93, BCM$^+$90], thus can be considered as a complementary technique to BDD-based model checking [CBRZ01, BCC$^+$03, Zha14].

1.2.2 Axiomazing Finite Models

Monadic second-order logic [Cou90, CE12] is an extension of first-order logic that allows quantification over monadic (unary) predicates (sets). It is preferred by logicians because it is decidable for many sets of (finite or infinite) structures. Furthermore, it is suitable for expressing numerous graph properties. For example, in the graph $G = \langle V, edge \rangle$,

where V is the set of vertices, and $edge \subseteq V \times V$ is the relationship between two vertices, the property *starting from* s_1*, there exists an infinite path* can be expressed as:

$$\exists Y(s_1 \in Y \wedge \forall x(x \in Y \Rightarrow \exists x'(edge(x, x') \wedge x' \in Y))).$$

To prove the properties of a graph in first-order deduction systems, in [DJ13b], the monadic second-order formula is treated as a two-sorted first-order formula, where the sets are denoted by terms of *class* sort, and the membership are represented by the following axioms

$$\forall x \ \neg x \in \varnothing$$

$$\forall x \forall y \forall Z(x \in add(y, Z) \Leftrightarrow (x = y \vee x \in Z))$$

where $\varnothing$ is a constant for the empty clauses and the binary function symbol *add* denotes the operation of adding an element to a class. For the proving of the formula with universal quantifiers, the following axiom schema is needed:

$$(s_1/x)A \wedge \cdots \wedge (s_n/x)A \Rightarrow \forall x A$$

where $s_1, ..., s_n$ are all the vertices of a graph. The logic structure is also represents by a set of axioms:

- for each predicate symbol Q of arity k and each k-tuple of constants $c_1, ..., c_k$ the axiom $Q(c_1, ..., c_k)$ if the sequence $\langle c_1, ..., c_k \rangle$ is in the interpretation of Q in the logical structure of the graph and the axiom $\neg Q(c_1, ..., c_k)$ otherwise.
- the axioms $s_i = s_i$ and $\neg s_i = s_j$, where $s_1, ..., s_n$ are all the vertices of the graph, $1 \leq i \leq n$, $1 \leq j \leq n$ and $i \neq j$.

By taking these axioms into account, the graph properties that are expressed by monadic second-order formulas can be proved by first-order theorem provers such as Vampire [KV13], E [KSU13], iProver [Kor08], etc..

1.2.3 Deduction Systems for Model Checking

Gilles Dowek and Ying Jiang [DJ13a] gave a slight extension of CTL, named SCTL, where predicates may have an arbitrary arity. For example, the judgment $M, s \models EGp$ in general CTL model checking verification is represented by the formula $EG_x(p(x))(s)$ in SCTL, where the variable x is bounded by EG. The deduction system they defined for SCTL is a special sequent calculus that is tailored to each finite model. In this

deduction system, the greatest fixpoints are not represented by formulas, but reflected in the inference rules. For instance, the inference rules for the temporal operator EG are as follows:

$$\frac{\vdash (s/x)\phi \qquad \Gamma, EG_x(\phi)(s) \vdash EG_x(\phi)(s')}{\Gamma \vdash EG_x(\phi)(s)} \text{ EG-right } s \to s'$$

$$\frac{}{\Gamma \vdash EG_x(\phi)(s)} \text{ EG-merge } EG_x(\phi)(s) \in \Gamma$$

In the first rule, the co-inductive formula $EG_x(\phi)(s)$, whose subformula can be proved, is recorded in the left-hand side of the sequent. In the case when the co-inductive formula appears in both sides of the sequent, one can use the special rule *merge* to end the proof. This calculus is in fact a one-sided sequent calculus, in which the left-hand sides of the sequents are only used to record the co-inductive formulas.

Bernhard Beckert and Steffen Schlager [BS01] defined a sequent calculus for first-order dynamic logic with trace modalities (DLT), which is an extension of dynamic logic with additional trace modalities $[\![\cdot]\!]$ ("throughout") and $\langle\langle\cdot\rangle\rangle$ (" at least once"). Dynamic Logic is a first-order modal logic with modalities $[\alpha]$ and $\langle\alpha\rangle$ for every program α. In deterministic programs, the formula $\phi \Rightarrow \langle\alpha\rangle\psi$ is valid if, for every state s satisfying pre-condition ϕ, a run of the program α starting from s terminates, and in the terminating state the post-condition ψ holds. The formula $\phi \Rightarrow [\alpha]\psi$ is valid if for every state s satisfying pre-condition ϕ and the program α starting from s does not terminate, or if α terminates, ψ holds on the terminating state. $[\![\alpha]\!]\phi$ means that ϕ holds on each state of the program α, while the semantic of $\langle\langle\alpha\rangle\rangle\phi$ is ϕ holds on at least one state of α. The inference rules of the modalities are in fact a performation of the symbolic program execution. For example, the rules

$$\frac{\Gamma \vdash Inv, \Delta \quad Inv, b \vdash [\alpha]Inv \quad Inv, \neg b \vdash \phi}{\Gamma \vdash [\text{while } b \text{ do } \alpha]\phi, \Delta}$$

$$\frac{\Gamma \vdash Inv, \Delta \quad Inv, b \vdash [\alpha]Inv \quad Inv, b \vdash [\![\alpha]\!]\phi \quad Inv, \neg b \vdash \phi}{\Gamma \vdash [\![\text{while } b \text{ do } \alpha]\!]\phi, \Delta}$$

are for the *while* loops in the modalities $[\cdot]$ and $[\![\cdot]\!]$, where b is a quantifier-free first-order formula and Inv is a loop invariant, i.e., a DLT-formula that must be true before and after each execution of the loop body. For the rule of *while* loop in $[\cdot]$, there are three premises: the first one expresses that the invariant Inv holds in the current state, i.e. before the loop is started; the second one expresses that if Inv holds before executing the loop body α, then it holds still if and when α terminates; the third one expresses

that ϕ—the formula that supposedly holds after the executing the loop—is a logical consequence of the invariant and the negation of the loop condition b. In the rule for $[\![\cdot]\!]$, the first two premises have the same meaning as for $[\cdot]$. The last premise is only needed for the case when b is false in the beginning such that the loop body α is never executed. The third premise is required to show that ϕ remains true throughout the execution of α if the invariant is true at the beginning.

1.3 Contributions

Our work in this dissertation are around the way of solving model checking problems with automated theorem proving method. From the beginning to the end, there are three contributions:

1. A propositional encoding of two graph traversal problems is presented. The first problem is to find a cycle in the graph, starting from a given vertex. The second one is to traverse all the vertices that are reachable from a given vertex, until a vertex, which has no successor, is reached. This work is inspired from the classical graph traversal algorithms, and it is the first time to solve graph traversal problems by simulating the running of graph traversal algorithms with automated theorem provers.

2. A theoritical basis of sloving CTL model checking problems with automated theorem provers is presented. To achieve this goal, an alternative semantics of CTL is defined, where all the temporal o formulas are expressed with finite paths. Then the model checking problems are represented by first-order formulas of a two-sorted language. Finally, the transition system to be checked and the logical equivalences between the two-sorted first-order formulas encoded as proposition rewrite rules. Thus, the specification of the model checking problems can be proved by first-order deduction systems modulo these rewrite rules.

3. A symbolic model checking method, based on Polarized Resolution Modulo, is illustrated in this dissertation. This method is implemented on an off-the-shelf automated theorem prover—iProver Modulo, which is a first-order theorem prover with the implementation of Ordered Polarized Resolution Modulo. The experimental results shows that, Resolution Modulo can be considered as a new way to quickly determine whether a temporal property is violated or not in transition system models.

All in all, from the theoretical basis to the implementation techniques, a sound and complete automated theorem proving strategy for finite transition system models is presented in this dissertation.

1.4 Outline

This document is organized as follows:

Chapter 2. The background of this dissertation, theorem proving systems and the procedure of solving model checking problems, is presented.

Chapter 3. We propose a way of solving some graph traversal problems by resolution, which is an automated theorem proving method.

Chapter 4. We express CTL for a given finite transition system in Deduction Modulo. This way, the theoretical base of solving model checking model checking problems with proof-search algorithms for Deduction Modulo is built.

Chapter 5. We present the procedure to encode model checking problems as input of iProver Modulo, and the experimental comparison among iProver Modulo, VERDS and NuSMV.

Chapter 6. We concludes the thesis and presents some future work.

Publication

Section 3 is an extension of [Ji15b]. Section 4 and Section 5 is an extension of [Ji15a].

2 State of the Art

The work in this dissertation is to solve model checking problems with theorem proving systems. Thus, the background of this dissertation contains two aspects: theorem proving systems and the procedure of solving model checking problems. In order to make our work easier to understand, we describe in this chapter the core of theorem proving systems, especially theorem proving modulo, and model checking procedures. The interested reader can refer to [DHK03, Dow10, CGP99] for more detailed definitions of the various concepts presented hereafter.

2.1 Deduction Modulo

Deduction Modulo is a reformulation of Predicate Logic where some axioms—possibly all—are replaced by rewrite rules. For example, the axiom $P \Leftrightarrow (Q \vee R)$ can be replaced by the rewrite rule $P \hookrightarrow (Q \vee R)$, meaning that during the proof, P can be replaced by $Q \vee R$ at any time. This way, the size of a proof may be much smaller. A deduction can be formulated using inference rules such as Sequent Calculus, Natural Deduction, Hilbert Systems. In this thesis, the deductions are modeled by Sequent Calculus, which is one of the most studied formalism of structural proof theory.

2.1.1 Basic Definitions

First-order Symbols We consider first-order formulas built from quantifies, variables, function symbols, predicate symbols and logical connectives. We will mainly deal with logical symbols $\forall$, $\exists$, $\top$, $\bot$, $\neg$, $\vee$, $\wedge$. Sometimes the connectives $\Rightarrow$ and $\Leftrightarrow$, which

can identically defined by the main symbols, are used for abbreviations. In this thesis, we will use *many-sorted languages* [Dow11]. A many-sorted language is a tuple $\mathcal{L} = (S, \mathcal{F}, \mathcal{P})$ where

1. S is a nonempty set of sorts.
2. $\mathcal{F}$ is a countable set of function symbols whose arities are constructed using sorts that belong to S.
3. $\mathcal{P}$ is a countable set of predicate symbols whose arities are constructed using sorts that belong to S.

A *term* of sort σ is either a variable of sort σ or an expression $f(t_1, ..., t_n)$, where f is a function symbol of arity $\sigma_1 \times \cdots \times \sigma_n \to \sigma$ and t_i is a term of sort σ_i, for $i = 1, ..., n$. A function symbol of arity 0 is called a constant. A term with no free variables is called a *ground term*. An *atomic formula* (also know simply as an *atom*) is an expression of the form $p(t_1, ..., t_n)$, where p is a predicate symbol of arity $\sigma_1 \times \cdots \times \sigma_n$ and t_i is a term of sort σ_i, for $i = 1, ..., n$. A predicate symbol of arity 0 is called a *propositional constant*. A formula with no free variables is called a *sentence* or a *ground* formula.

Model A *model* of the language $\mathcal{L} = (S, \mathcal{F}, \mathcal{P})$ is a structure of the form $\mathcal{M} = ((\hat{\sigma})_{\sigma \in S}, B, (\hat{f})_{f \in \mathcal{F}}, (\hat{p})_{p \in \mathcal{P}}, \hat{\top}, \hat{\bot}, \hat{\neg}, \hat{\wedge}, \hat{\vee}, \hat{\forall}, \hat{\exists})$ where

- $\hat{\sigma}$ is a non-empty set of elements for each sort σ in S,
- B is a non-empty set in which the two distinguished elements $\hat{\top}$ and $\hat{\bot}$ are included.
- $\hat{f}$ is a function from $\hat{\sigma}_1 \times \cdots \times \hat{\sigma}_n$ to $\hat{\sigma}$ if $f \in \mathcal{F}$ is a function symbol of arity $\sigma_1 \times \cdots \times \sigma_n \to \sigma$.
- $\hat{p}$ is a function from $\hat{\sigma}_1 \times \cdots \times \hat{\sigma}_n$ to B if $p \in \mathcal{P}$ is a predicate symbol of arity $\sigma_1 \times \cdots \times \sigma_n$.
- $\hat{\neg}$ is a function from B to B. $\hat{\wedge}$ and $\hat{\vee}$ are functions from $B \times B$ to B. $\hat{\forall}$ and $\hat{\exists}$ are functions from $\mathcal{P}^+(B)$ (non-empty powerset of B) to B.

A formula A is said to be *true* in a model $\mathcal{M}$ its interpretation is $\hat{\top}$, $false$ otherwise. The logical connectives are interpreted in the standard way. A formula or a set of formulas is called *satisfiable*, or *consistent*, if it has a model; otherwise, this formula or this set is said to be em unsatisfiable or *inconsistent*. A formula is said to be *valid* if it is true in all models. A formula A is a *logical consequence* of the set of formulas Γ (written

$\Gamma \models A$), if A is true in all models of Γ. Two formulas A and B are said to be *logically equivalent* (written $A \equiv B$), if and only if they have the same truth value in all models.

Substitution A substitution is a mapping from variables to expressions, with a finite domain, such that each variable is associated to an expression of the same sort. The replacement of variables $x_1, \ldots, x_n$ by $t_1, \ldots, t_n$ in a term or a proposition A can be denoted by $(t_1/x_1, \ldots, t_n/x_n)A$. The application of a substitution σ in a term or a proposition A is denoted as σA.

2.1.2 Polarized Sequent Calculus Modulo

(Polarized) Sequent Calculus Modulo is an extension of Sequent Calculus, by taking (polarized) rewrite rules into account. In this part, first we will give an short overview of Sequent Calculus, then present the definition of (polarized) rewrite system. After that, the combination of these two systems, (Polarized) Sequent Calculus Modulo, is given.

Sequent Calculus A *sequent* is a pair $\Gamma \vdash \Delta$, where Γ and Δ are sets of propositions. For a sequent $A_1, ..., A_m \vdash B_1, ..., B_n$, the left-hand-side or the right-hand-side may be empty. The semantics of a sequent is an assertion that whenever every A_i is true, at least one B_i will also be true. Hence the empty sequent, whose both sides are empty, is false. The comma in the left-hand-side can be expressed as "and", while in the right-hand-side can be thought of as "or". The sequent calculus is a set of inference rules, in which all the premises and conclusions are represented by sequents. For example, the right rule of the conjunction can be expressed as

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta} \wedge\text{-r}$$

Rewrite Rules A *term rewrite rule* is a pair of terms $l \hookrightarrow r$, to indicate that the left hand side can be replaced by the right hand side. A *proposition rewrite rule* is a pair of formulas $l \hookrightarrow r$, in which l is an atomic formula, and r is an arbitrary formula. Note that in this dissertation, we only consider the proposition rewrite rules. In case a term rewrite rule is needed, we can use a special proposition rewrite rule, in which the left hand side is an atomic formula whose main symbol is an equality, and the right hand side is $\top$. For example, the term rewrite rule $x \times 1 \hookrightarrow x$ can be replaced by the proposition rewrite rule $eq(x \times 1, x) \hookrightarrow \top$.

Polarized Rewrite System A *rewrite system* is a set $\mathcal{R}$ of rewrite rules. Formally, the relation $l \hookrightarrow_{\mathcal{R}} r$ denotes that l rewrites, in one step, to r by the system $\mathcal{R}$. $\overset{*}{\hookrightarrow}_{\mathcal{R}}$ is the reflexive-transitive closure of $\hookrightarrow_{\mathcal{R}}$. A *polarized rewrite system* is a pair $\mathcal{R} = \langle \mathcal{R}_-, \mathcal{R}_+ \rangle$, where $\mathcal{R}_-$ and $\mathcal{R}_+$ are sets of rewrite rules. The rules in $\mathcal{R}_-$ are called *negative rules* and those in $\mathcal{R}_+$ are called *positive rules*. The formula A is positively rewritten into formula B ($A \hookrightarrow_+ B$) if it is rewritten by a positive rule at a positive position or by a negative rule at a negative position. It is rewritten negatively ($A \hookrightarrow_- B$) if it is rewritten by a positive rule at a negative position or by a negative rule at a positive position.

Polarized Sequent Calculus Modulo In Sequent Calculus Modulo [DHK03], the equivalence between a pair of propositions are taken into account, so the inference rules in Sequent Calculus Modulo cannot be expressed as usual, but including the rewrite rules. For instance, the right rule of the conjunction above is stated as

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash C, \Delta} \wedge\text{-r } C \overset{*}{\hookrightarrow} A \wedge B$$

Polarized Sequent Calculus Modulo [Dow02, Dow10] is an extension of Sequent Calculus Modulo, where the rewrite system are replaced by polarized rewrite system—some rules can only be used at the positive occurrences, while others can only be used at negative ones. For example, the axiom $P \Rightarrow Q$ can be transformed into the negative rule $P \hookrightarrow_- Q$ and the positive rule $Q \hookrightarrow_+ P$, but the negative rule can only be used when P occurs at a negative position, while the positive rule can only be used when Q occurs at a positive position. The inference rules of Polarized Sequent Calculus Modulo are in Figure 2.1.

$$\frac{}{A \vdash_{\mathcal{R}} B}\ \text{axiom if } A \overset{*}{\hookrightarrow}_{-} P, B \overset{*}{\hookrightarrow}_{+} P$$

$$\frac{\Gamma, B \vdash_{\mathcal{R}} \Delta \qquad \Gamma \vdash_{\mathcal{R}} C, \Delta}{\Gamma \vdash_{\mathcal{R}} \Delta}\ \text{cut if } A \overset{*}{\hookrightarrow}_{-} B, A \overset{*}{\hookrightarrow}_{+} C$$

$$\frac{\Gamma \vdash_{\mathcal{R}} \Delta}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \text{weak-l}$$

$$\frac{\Gamma \vdash_{\mathcal{R}} \Delta}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \text{weak-r}$$

$$\frac{\Gamma, B, C \vdash_{\mathcal{R}} \Delta}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \text{contr-l if } A \overset{*}{\hookrightarrow}_{-} B, A \overset{*}{\hookrightarrow}_{-} C$$

$$\frac{\Gamma \vdash_{\mathcal{R}} B, C, \Delta}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \text{contr-r if } A \overset{*}{\hookrightarrow}_{+} B, A \overset{*}{\hookrightarrow}_{+} C$$

$$\frac{}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \bot \text{ if } A \overset{*}{\hookrightarrow}_{-} \bot$$

$$\frac{}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \top \text{ if } A \overset{*}{\hookrightarrow}_{+} \top$$

$$\frac{\Gamma \vdash_{\mathcal{R}} B, \Delta}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \neg\text{-l if } A \overset{*}{\hookrightarrow}_{-} \neg B$$

$$\frac{\Gamma, B \vdash_{\mathcal{R}} \Delta}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \neg\text{-r if } A \overset{*}{\hookrightarrow}_{+} \neg B$$

$$\frac{\Gamma, B, C \vdash_{\mathcal{R}} \Delta}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \wedge\text{-l if } A \overset{*}{\hookrightarrow}_{-} B \wedge C$$

$$\frac{\Gamma \vdash_{\mathcal{R}} B, \Delta \qquad \Gamma \vdash_{\mathcal{R}} C, \Delta}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \wedge\text{-r if } A \overset{*}{\hookrightarrow}_{+} B \wedge C$$

$$\frac{\Gamma, B \vdash_{\mathcal{R}} B, \Delta \qquad \Gamma, C \vdash_{\mathcal{R}} \Delta}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \vee\text{-l if } A \overset{*}{\hookrightarrow}_{-} B \vee C$$

$$\frac{\Gamma \vdash_{\mathcal{R}} B, C, \Delta}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \vee\text{-r if } A \overset{*}{\hookrightarrow}_{+} B \vee C$$

$$\frac{\Gamma \vdash_{\mathcal{R}} B, \Delta \qquad \Gamma, C \vdash_{\mathcal{R}} \Delta}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \Rightarrow\text{-l if } A \overset{*}{\hookrightarrow}_{-} B \Rightarrow C$$

$$\frac{\Gamma, B \vdash_{\mathcal{R}} C, \Delta}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \Rightarrow\text{-r if } A \overset{*}{\hookrightarrow}_{+} B \Rightarrow C$$

$$\frac{\Gamma, C \vdash_{\mathcal{R}} \Delta}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \forall\text{-l if } A \overset{*}{\hookrightarrow}_{-} \forall x B, (t/x)B \overset{*}{\hookrightarrow}_{-} C$$

$$\frac{\Gamma \vdash_{\mathcal{R}} B, \Delta}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \forall\text{-r if } A \overset{*}{\hookrightarrow}_{+} \forall x B, x \notin FV(\Gamma, \Delta)$$

$$\frac{\Gamma, B \vdash_{\mathcal{R}} \Delta}{\Gamma, A \vdash_{\mathcal{R}} \Delta}\ \exists\text{-l if } A \overset{*}{\hookrightarrow}_{-} \exists x B, x \notin FV(\Gamma, \Delta)$$

$$\frac{\Gamma \vdash_{\mathcal{R}} C, \Delta}{\Gamma \vdash_{\mathcal{R}} A, \Delta}\ \exists\text{-r if } A \overset{*}{\hookrightarrow}_{+} \exists x B, (t/x)B \overset{*}{\hookrightarrow}_{+} C$$

FIGURE 2.1: Polarized Sequent Calculus Modulo

Example 2.1. *To decide whether the mulplication to any two natural numbers is an even number or not, the following three axioms are given.*

$$Even(zero)$$

$$\forall x(Even(s(s(x))) \Leftrightarrow Even(x))$$

$$\forall x \forall y(Even(mul(x,y)) \Leftrightarrow (Even(x) \vee Even(y)))$$

These axioms can be translated into the following polarized rewrite rules:

$$Even(zero) \hookrightarrow_{\pm} \top$$

$$Even(s(s(x))) \hookrightarrow_{\pm} Even(x)$$

$$Even(mul(x,y)) \hookrightarrow_{\pm} Even(x) \vee Even(y)$$

Then the sequent $\vdash_{\mathcal{R}} Even(mul(s(s(zero)), s(s(s(zero)))))$ *can be proved by the inference rules in Figure 2.1, that is*

$$\dfrac{\dfrac{}{\vdash_{\mathcal{R}} Even(s(s(zero))), Even(s(s(s(zero))))}\,\top\text{-}r}{\vdash_{\mathcal{R}} Even(mul(s(s(zero)), s(s(s(zero)))))}\,\vee\text{-}r$$

2.1.3 Polarized Resolution Modulo

In this part, we present an overview of Polarized Resolution Modulo, which is proof-search algorithm for Polarized Deduction Modulo. (Polarized) Resolution Modulo is an extension of Resolution, by considering (polarized) rewrite rules as part of the proof-search procedure.

Resolution

Literal A *literal* an atomic formula (positive literal) or the negation of an atomic formula (negative literal). A literal which contains no variables is called a *ground literal*. Two literals A and $\neg A$ are said to be *complementary*.

Clause A *clause* is a set of literals. The empty clause is denoted as $\Box$. A clause with only ground literals is called a *ground clause*. A formula is in *clausal normal form* if it is $\bot$ or $\forall x_1, \ldots, x_k(L_1 \vee \cdots \vee L_n)$, where $L_1, \ldots, L_n$ are literals and $x_1, \ldots, x_k$ are all the free variables of $L_1, \ldots, L_n$. In this dissertation, when we write a formula in clausal

normal form, we will omit writing the quantifications, and a clause will be represented by a formula of clausal normal form, that is, the disjunction of all the literals in the clause. Besides, for a set of formulas Γ, the set of clauses for Γ is denoted by $cl(\Gamma)$.

Unifier An *unification problem* is a finite set of equations of the form $t = u$. A solution of a unification problem is a substitution σ such that for each equation $t = u$ in the set, σt and σu are identical. Such a substitution is also called a *unifier* of the unification problem. A solution σ of a unification problem is said to be most general if for each solution ρ there exists a substitution η such that $\rho = \eta \circ \sigma$. Such a solution is also called a *most general unifier (mgu)* of the unification problem.

Theorem 2.1 (Herbrand's theorem)**.** *A set of clauses $\mathcal{C}$ is unsatisfiable if and only if there exists a finite set of instances of $\mathcal{C}$ clauses which is unsatisfiable.*

Resolution Resolution is a refutationally complete theorem proving method. For a set of clauses, if it is unsatisfiable, then the empty clause can be derived by repeatedly applying the two inference rules

$$\frac{C \vee A \qquad D \vee \neg B}{\sigma(C \vee D)} \text{ Resolution, } \sigma = mgu(A, B)$$

$$\frac{C \vee A \vee B}{C \vee A} \text{ Factoring, } \sigma = mgu(A, B)$$

until the given clause set is saturated. An machine-oriented resolution method [Rob65a] was invented by Robinson in 1965. In his work, the inference rules are applied on ground clauses. In fact, the resolution rule is a version of cut rule in sequent calculus that is restricted to atomic formulas, whereas factoring is an instance of contraction. Thus the completeness of resolution can be derived by the completeness of propositional sequent calculus. Then from Theorem 2.1, a link between ground clauses and general clauses is established.

Example 2.2. *Prove that the set of clauses*

$$Even(mul(x, y)) \vee \neg Even(x)$$

$$Even(mul(x, y)) \vee \neg Even(y)$$

$$Even(s(s(z))) \vee \neg Even(z)$$

$$Even(zero)$$

$$\neg Even(mul(s(s(zero)), s(s(s(zero)))))$$

are unsatisfiable. The Resolution steps of the proof is in Figure 2.2

(1)	$Even(mul(x, y)) \vee \neg Even(x)$	[input]
(2)	$Even(mul(x, y)) \vee \neg Even(y)$	[input]
(3)	$Even(s(s(z))) \vee \neg Even(z)$	[input]
(4)	$Even(zero)$	[input]
(5)	$\neg Even(mul(s(s(zero)), s(s(s(zero)))))$	[input]
(6)	$\neg Even(s(s(zero)))$	[resolution between (1) and (5)]
(7)	$\neg Even(zero)$	[resolution between (3) and (6)]
(8)	$\square$	[resolution between (4) and (7)]

FIGURE 2.2: Resolution Steps of Example 2.2

Polarized Resolution Modulo

Clausal Rewrite System A rewrite system is *clausal* if negative rules rewrite atomic propositions to clausal propositions and positive rules atomic propositions to negations of clausal propositions.

One-Way Clause For each rewrite rule in the clausal rewrite system R , we associate a clause called the *one-way clause* of R. Each one-way clause has a underlined literal, which is called the *selected literal* and is privileged to apply resolution rules.

Polarized Resolution Modulo A proof-search method for Sequent Calculus Modulo can be built by extending the existing proof-search method, which takes the rewrite rules into account. For instance, *Extended Narrowing and Resolution(ENAR)* [DHK03] is a proof-search method for sequent calculus modulo. In this dissertation, *Polarized Resolution Modulo(PRM)* [Dow10], which is the proof-search method of Polarized Sequent Calculus Modulo, will be used in proving formulas automatically. Rules of PRM is presented in Figure 2.3. Notice that the **Extended Narrowing** rule based on a one-way clause can be seen as an instance of applying **Resolution** rule between an ordinary clause and a one-way clause.

$$\textbf{Resolution}\ \frac{P \vee C \qquad \neg Q \vee D}{\sigma(C \vee D)}\ \sigma = mgu(P, Q)$$

$$\textbf{Factoring}\ \frac{L \vee K \vee C}{\sigma(L, C)}\ \sigma = mgu(L, K)$$

$$\textbf{Ext.Narr.}\ \frac{P \vee C}{\sigma(D \vee C)}\ \text{if } \underline{\neg Q} \vee D \text{ is a one-way clause of } \mathcal{R},\ \sigma = mgu(P, Q)$$

$$\textbf{Ext.Narr.}\ \frac{\neg P \vee C}{\sigma(D \vee C)}\ \text{if } \underline{Q} \vee D \text{ is a one-way clause of } \mathcal{R},\ \sigma = mgu(P, Q)$$

$$\textbf{Ext.Narr.}\ \frac{\neg L \vee C}{\sigma(L' \vee C)}\ L \overset{\mathfrak{p},\sigma}{\rightsquigarrow} L' \text{ by a term rewrite rule, } L_{|\mathfrak{p}} \notin \mathcal{V}$$

FIGURE 2.3: Polarized Resolution Modulo

Example 2.3. *The polarized rewrite rules in Example 2.1 can be represented by the following one-way clauses*

$$\underline{Even(zero)}$$

$$\underline{Even(s(s(x)))} \vee \neg Even(x)$$

$$\underline{Even(mul(x, y))} \vee \neg Even(x)$$

$$\underline{Even(mul(x, y))} \vee \neg Even(y)$$

Starting from $\neg Even(mul(s(s(zero)), s(s(s(zero)))))$*, The resolution steps are the same as Figure 2.2. However, the difference is that, in Example 2.2, resolution can also be applied between other input clauses, the result of which are considered to be redundant. For example, we can apply resolution between (2) and (3), with* $z = mul(x, y)$*, the new clause generated is* $Even(s(s(mul(x, y)))) \vee \neg Even(y)$*, which is a redundant resolution step. In Polarized Resolution Modulo, these kind of redundant steps are partially avoided.*

Theorem 2.2 (Soundness and Completeness of $PRM_{\mathcal{R}}$)**.** *Let* Γ *be a set of sentences.* $cl(\Gamma) \mapsto_{\mathcal{R}} \square$ *iff the sequent* $\Gamma \vdash$ *has a cut free proof.*

The proof of this theorem refer to [Dow10].

2.1.4 Ordered Polarized Resolution Modulo

Resolution has several refinements, for example hyper-resolution, semantic resolution, ordered resolution and the mixture of these methods [Rob65b, CL97, GlGl02]. In [Bur10], Guillaume Burel showed that using an ordering-based literal selection preserves the completeness of $PRM_{\mathcal{R}}$. The ordering $\succ$ on literals is well-founded and stable by substitution. The inference rules of Ordered Polarized Resolution Modulo ($OPRM^{\succ}_{\mathcal{R}}$) are

the refinements of $PRM_{\mathcal{R}}$, by selecting the literals with the maximal order in the ordinary clauses to apply the inference rules. For example, the Resolution rule in $OPRM^{\succ}_{\mathcal{R}}$ is as follows.

$$\textbf{Resolution}\ \frac{P^* \vee C \qquad \neg Q^* \vee D}{\sigma(C \vee D)}\ \sigma = mgu(P, Q)$$

the literal noted with $*$ means that it has the maximal order in the clause

Selection Function A *selection function* is a mapping on clauses that selects a subset (non-empty) of literals from each clause. In Ordered Polarized Resolution Modulo, if the ordering of literals are decided by a selection function δ, we denote the system as PRM^{δ}.

2.2 Temporal Logic

Given a model of a system, the technique of exhaustively and automatically checking whether this model meets a given specification is called Model Checking. The process of model checking consists three steps:

Modeling A large class of model checking algorithms are developed for hardware and software designs. Given a design, the first step is to convert it into a formalism that can be accepted by the model checking tool.

Specification To verify a property that a reactive system should satisfy, we need to specify it before. Usually, if a transition system is used as the model of a reactive system, the specification is given by Temporal logic [HC68, Eme95], which can be used to assert how the behavior of a system evolves over time.

Verification As is said in the beginning of this section, the verification is completely automatic. Besides, when the result is negative, a counterexample should be given, which can help the system designer track down where the error occurs.

In this section, we focus on the description of Temporal logics on finite state transition systems.

2.2.1 Transition Systems

To build a suitable transition system, two aspects should be considered. On the one hand, the system must contain enough information to prove the correctness of the system. On the other hand, ignore the properties that do not affect the correctness of the checked properties, to make the verification as simple as possible.

In this dissertation, the mainly concerned systems are reactive systems [ZA95], for example, the air traffic control system. Such systems may react to external events and often do not terminate. Normally, the behavior of a reactive system can be described by a transition system. In this dissertation, the Kripke structure [CGP99] is considered. For other models of reactive systems, refer to [SNW96].

Definition 2.3 (Kripke Structure)**.** Let AP be a set of atomic formulas. A Kripke structure M over AP is a three tuple $M = (S, R, L)$ where

- S is a finite set of states.
- $R \subseteq S \times S$ is a total transition relation, that is, for every state $s \in S$ there is a state $s' \in S$ such that $R(s, s')$.
- $L : S \to \mathcal{P}(AP)$ is a function that labels each state with a subset of AP.

Note that for each state s, the set of states $\{s' \in S \mid R(s, s')\}$ is denoted as $R(s)$. An infinite path is an infinite sequence of states $\pi = \pi_0\pi_1...$ such that $\forall i \geq 0$, $R(\pi_i, \pi_{i+1})$. Note that the sequence $\pi_i\pi_{i+1}...\pi_j$ is denoted as π_i^j and the path π with $\pi_0 = s$ is denoted as $\pi(s)$.

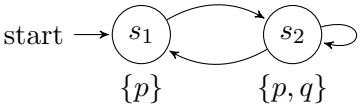


FIGURE 2.4: Kripke Structure Example

Example 2.4. *The transition system in Figure 2.4 is a simple Kripke structure example.*

2.2.2 Temporal Logic

Temporal logic is a kind of modal logic, which is used to describe sequences of transitions between states in a reactive system. According to the structure of time, temporal logics are often divided into linear-time temporal logic (LTL) [Pnu77] and branching-time temporal logic (CTL) [BAPM83, EC80]. Besides, CTL* [EH86] combines both

branching-time and linear-time operators. All of these logics can be transformed into μ-calculus [Koz82]. In this dissertation, the temporal properties are expressed by CTL formulas.

Syntax

Let AP be a set of atomic formulas and p ranges over AP. The set of CTL formulas Φ over AP is defined as follows:

$$\begin{aligned}\Phi ::= \quad & p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid AX\Phi \mid EX\Phi \mid AF\Phi \mid EF\Phi \mid AG\Phi \mid EG\Phi \mid \\ & AU(\Phi,\Phi) \mid EU(\Phi,\Phi) \mid AR(\Phi,\Phi) \mid ER(\Phi,\Phi)\end{aligned}$$

Each CTL operator name is a pair of symbols. The first one is either A ("for All paths"), or E ("there Exists a path"). The second one is one of X ("neXt state"), F("in a Future state"), G ("Globally in the future"), U ("Until") or R ("Release").

Semantics

In this thesis, the semantics of CTL formulas are interpreted over Kripke structures.

Definition 2.4 (Semantics of CTL)**.** Let p be an atomic formula. Let φ, φ_1, φ_2 be CTL formulas. $M, s \models \varphi$ is defined inductively on the structure of φ as follows:

$M, s \models p$ iff $p \in L(s)$

$M, s \models \neg\varphi_1$ iff $M, s \not\models \varphi_1$

$M, s \models \varphi_1 \wedge \varphi_2$ iff $M, s \models \varphi_1$ and $M, s \models \varphi_2$

$M, s \models \varphi_1 \vee \varphi_2$ iff $M, s \models \varphi_1$ or $M, s \models \varphi_2$

$M, s \models AX\varphi_1$ iff $\forall s' \in R(s)$, $M, s' \models \varphi_1$

$M, s \models EX\varphi_1$ iff $\exists s' \in R(s)$, $M, s' \models \varphi_1$

$M, s \models AG\varphi_1$ iff $\forall \pi(s)$, $\forall i \geq 0$, $M, \pi_i \models \varphi_1$

$M, s \models EG\varphi_1$ iff $\exists \pi(s)$ s.t. $\forall i \geq 0$, $M, \pi_i \models \varphi_1$

$M, s \models AF\varphi_1$ iff $\forall \pi(s)$, $\exists i \geq 0$ s.t. $M, \pi_i \models \varphi_1$

$M, s \models EF\varphi_1$ iff $\exists \pi(s)$, $\exists i \geq 0$ s.t. $M, \pi_i \models \varphi_1$

$M, s \models AU(\varphi_1, \varphi_2)$ iff $\forall \pi(s)$, $\exists j \geq 0$ s.t. $M, \pi_j \models \varphi_2$ and $\forall 0 \leq i < j$, $M, \pi_i \models \varphi_1$

$M, s \models EU(\varphi_1, \varphi_2)$ iff $\exists \pi(s)$, $\exists j \geq 0$ s.t. $M, \pi_j \models \varphi_2$ and $\forall 0 \leq i < j$, $M, \pi_i \models \varphi_1$

$M, s \models AR(\varphi_1, \varphi_2)$ iff $\forall \pi(s)$, $\forall j \geq 0$, either $M, \pi_j \models \varphi_2$ or $\exists 0 \leq i < j$ s.t. $M, \pi_i \models \varphi_1$

$M, s \models ER(\varphi_1, \varphi_2)$ iff $\exists \pi(s)$, $\forall j \geq 0$, either $M, \pi_j \models \varphi_2$ or $\exists 0 \leq i < j$ s.t. $M, \pi_i \models \varphi_1$

Example 2.5. *Let M be the Kripke structure in Figure 2.4. We have $M, s_1 \vDash EGp$ because there exists an infinite path, for instance $s_1, s_2, s_1, s_2 \ldots$ such that p holds on each state.*

Semantic Equivalence The CTL formulas φ and ψ are said to be semantic equivalent ($\varphi \equiv \psi$), if for any state in any model which satisfies one also satisfies the other. Following the semantics defined above, the equivalences below (**expansion laws**) holds.

$$AF\varphi \equiv \varphi \vee AXAF\varphi \qquad EF\varphi \equiv \varphi \vee EXEF\varphi$$
$$AG\varphi \equiv \varphi \wedge AXAG\varphi \qquad EG\varphi \equiv \varphi \wedge EXEG\varphi$$
$$AU(\varphi, \psi) \equiv \psi \vee (\varphi \wedge AXAU(\varphi, \psi)) \qquad EU(\varphi, \psi) \equiv \psi \vee (\varphi \wedge EXEU(\varphi, \psi))$$
$$AR(\varphi, \psi) \equiv \psi \wedge (\varphi \vee AXAR(\varphi, \psi)) \qquad ER(\varphi, \psi) \equiv \psi \wedge (\varphi \vee EXER(\varphi, \psi))$$

Minimal Set of Operators In CTL there is a minimal set of operators, which means that all CTL formulas can be expressed in terms of these operators. One of the minimal set is $\{\top, \vee, \neg, EG, EU, EX\}$. The transformation rules for this minimal set are as follows.

$AX\varphi \equiv \neg EX(\neg\varphi)$

$EF\varphi \equiv EU(\top, \varphi)$

$AF\varphi \equiv AU(\top, \varphi) \equiv \neg EG(\neg\varphi)$

$AG\varphi \equiv \neg EF(\neg\varphi) \equiv \neg EU(\top, \neg\varphi)$

$AU(\varphi, \psi) \equiv \neg(EU(\neg\psi, \neg(\varphi \vee \psi)) \vee EG(\neg\psi))$

$ER(\varphi, \psi) \equiv \neg AU(\neg\varphi, \neg\psi) \equiv EU(\psi, \varphi) \vee EG\psi$

$AR(\varphi, \psi) \equiv AU(\psi, \varphi) \vee AG\psi \equiv \neg EU(\neg\varphi, \neg\psi)$

Negation Normal Form A CTL formula is in negation normal form (NNF), if the negation $\neg$ is applied only to propositional symbols. Every CTL formula can be transformed into an equivalent formula in NNF by using the following equivalences (**De Morgan's laws**).

$$\neg\neg\varphi \equiv \varphi$$

$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi \qquad \neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$$

$$\neg AX\varphi \equiv EX\neg\varphi \qquad \neg EX\varphi \equiv AX\neg\varphi$$

$$\neg AF\varphi \equiv EG\neg\varphi \qquad \neg EG\varphi \equiv AF\neg\varphi$$

$$\neg AG\varphi \equiv EF\neg\varphi \qquad \neg EF\varphi \equiv AG\neg\varphi$$

$$\neg AU(\varphi,\psi) \equiv ER(\neg\varphi,\neg\psi) \qquad \neg ER(\varphi,\psi) \equiv AU(\neg\varphi,\neg\psi)$$

$$\neg AR(\varphi,\psi) \equiv EU(\neg\varphi,\neg\psi) \qquad \neg EU(\varphi,\psi) \equiv AR(\neg\varphi,\neg\psi)$$

2.3 Symbolic Model Checking

Initially, the algorithms for solving model checking problems used an *explicit* representation of the Kripke structures. However, in realistic designs, the number of states in the transition system can be very large and the explicit traversal of the state space becomes infeasible. This inspires the idea of symbolic model checking, in which the Kripke structure is encoded by boolean formulas.

In this section, two kinds of symbolic representation for finite states systems are presentated.

2.3.1 Binary Decision Dragrams

In this part we discuss how to represent Kripke structures symbolically using Binary Decision Diagrams (BDDs) [Bry86]. BBDs are a canonical data structure for the representation of boolean formulas. On a more abstract level, BDDs can be considered as a compressed representation of sets or relations.

Binary Decision Diagram

To discuss the form of BDDs, let's consider binary decision trees first, which is a special form of BDDs. A binary decision tree is a rooted, directed tree, which consists of two types of nodes: terminal nodes and decision nodes. Each decision node v is labeled

by a boolean variable $\mathsf{var}(v)$ and has two successors: $\mathsf{low}(v)$ corresponding to the case where v is assigned 0, while $\mathsf{high}(v)$ the assignment of v to 1. Each terminal node has a value, which is either 0 or 1. For example, the binary decision tree for the three input AND gate, represented by the formula $a \wedge b \wedge c$, is shown in Figure 2.5. However,

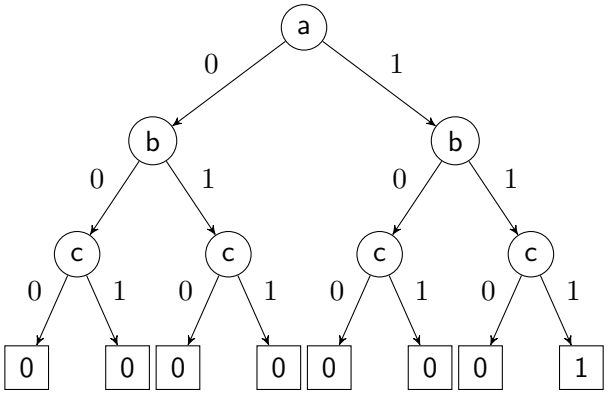


FIGURE 2.5: Binary Decision Tree for Three-input AND Gate

binary decision trees do not provide a very concise representation for boolean formulas. Usually, there is a lot of redundancy in binary decision trees. For example, in the tree of Figure 2.5, there are four subtrees with roots labeled by c, but only one is distinct. Thus, we can obtain a more concise representation for the boolean formulas by merging the ismorphic subtrees. This results in the definition of *binary decision diagram*, which is a directed acyclic graph. More precisely, a BDD is a rooted, directed acyclic graph, which consists of two types of nodes: terminal nodes and decision nodes. As in the case of binary decision trees, each decision node v is labeled by a boolean variable $\mathsf{var}(v)$ and has two successors: $\mathsf{low}(v)$ corresponding to the case where v is assigned 0, while $\mathsf{high}(v)$ the assignment of v to 1. Each terminal node has a value, which is either 0 or 1. A BDD is called 'ordered' if different variables appear in the same order on all paths from the root. In practical applications it is desirable to have a *reduced representation* of OBDD. A reduced OBDD can be achieved by repeadly applying the two rules to the graph:

- Merge any isomorphic subgraphs.
- Eliminate any node whose two children are isomorphic, and redirect all incoming edges of this node to one of its children.

For example, the reduced OBDD of Figure 2.5 is shown in Figure 2.6. Besides, the size of an OBDD may depend critically on the order of the variables. The readers intereasted can refer to Section 5 of [CGP99].

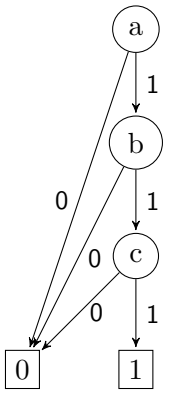


FIGURE 2.6: OBDD of Three-input AND Gate

Representing Kripke Structures

To represent a Kripke structure $M = (S, R, L)$ using OBDD, we must describe the set S, the relation R and the mapping L. For each state, we need to encode it into a list of binary numbers. Assume that the number of all the states is n and $2^{m-1} < n \leq 2^m$, then each state can be represented by a boolean vector of m number of boolean variables. If state s is a member of S, then in the OBDD for the set S, the value of the characteristic function for the boolean vector of s is 1. To represent the transition relations, two sets of boolean variables are needed, one to represent the starting state and the other to represent the final state. Let $\overline{x}$ be the boolean vector represention of a starting state, and $\overline{x}'$ representing the boolean vector of a final state. If $\overline{x}'$ is a successor of $\overline{x}$, then in the OBDD for the transition relations, the value of the characteristic function for the pair of boolean vectors $(\overline{x}, \overline{x}')$ is 1. Finally we consider the OBDD representation of atomic propositions. For the atomic proposition p, the set of states $\{s \mid p \in L(s)\}$ can be encoded into an OBDD, such that if $p \in L(s)$, then in the OBDD for the proposition p, the value of the characteristic function for boolean vector representation of the state s is 1.

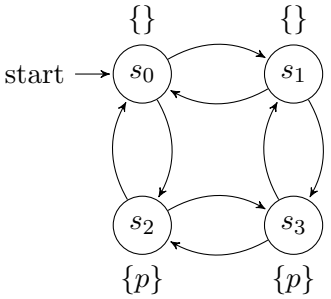


FIGURE 2.7: Kripke Structure Example for OBDD Representation

Example 2.6. *The Kripke structure in Figure 2.7 can be expressed as follows:*

States $s_0 : \neg a \wedge \neg b / \neg a' \wedge \neg b'$, $s_1 : \neg a \wedge b / \neg a' \wedge b'$, $s_2 : a \wedge \neg b / a' \wedge \neg b'$, $s_3 : a \wedge b / a' \wedge b'$, *in which* $\{a, b\}$ *and* $\{a', b'\}$ *are two set of boolean variables for the start states and final states respectively.*

Relations $(\neg a \wedge \neg b \wedge \neg a' \wedge b') \vee (\neg a \wedge \neg b \wedge a' \wedge \neg b') \vee (\neg a \wedge b \wedge \neg a' \wedge \neg b') \vee (\neg a \wedge b \wedge a' \wedge b') \vee (a \wedge \neg b \wedge \neg a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge \neg b') \vee (a \wedge b \wedge \neg a' \wedge b')$.

Atomic propositions $p : (a \wedge \neg b) \vee (a \wedge b)$.

By the methods mentioned above, these formulas can be converted to OBDDs to obtain more concise representations. For example the reduced OBDD for the transition relation is given in Figure 2.8, which is in fact a representation of the formula $(a \Leftrightarrow b) \wedge (\neg a' \Leftrightarrow b')$.

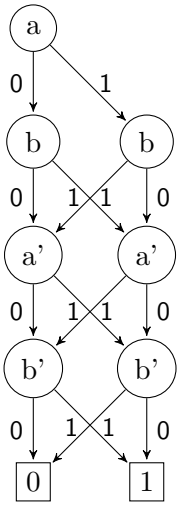


FIGURE 2.8: Transition Relations of Figure 2.7 in OBDD

2.3.2 Quantified Boolean Formulas

Quantified boolean formula (QBF) is a succinct representation of boolean formula, by introducing the existential and universal quantifiers, which can be applied to the boolean variables. For example, the formua $\forall p \exists q \exists r (p \wedge q \wedge r)$, is a QBF, in which p, q, r are boolean variables.

Representing Boolean Formulas by QBF

For any two boolean formulas $\phi(true)$ and $\phi(false)$, the connection of these two formulas can be represented as $\forall x \phi(x)$, the disjunction of these two formulas can be represented

as $\exists x\phi(x)$, where x is a boolean variable. In this way, the set of states which satisfy the atomic proposition p in Example 2.6 can be represented as $\exists b(a \wedge b)$.

2.4 Tools

In this section, we present the automated theorem proving and model checking tools that will be used in the implementations in the following sections. For theorem proving, the deduction modulo based theorem prover iProver Modulo [Bur10, Bur11] is taken into account. For model checkers, the symbolic model checker NuSMV [CCGR99] and QBF-based model checker VERDS [Zha12, Zha14] are considered.

2.4.1 iProver Modulo

Instead of implementing polarized resolution modulo from scratch, it is embeded into iProver [Kor08], and this is so called iProver Modulo [Bur11]. Thus, in the following part we will show iProver first, then present iProver Modulo.

iProver

iProver is a first-order theorem prover developed by Konstantin Korovin. It is implemented in a function language OCaml and integrates MiniSat solver [ES04] for propositional reasoning, which is implemented in C/C++.

iProver is based on an instantiation framework for first-order logic Inst-Gen [GK06]. In Inst-Gen calculus, the basic idea is to abstract the set of first-order clauses by a set of propositional clauses, in which all the variables are replaced by a distinguished constant. If the set of propositional clauses is unsatisfiable, then conclude with the first-order clauses unsatisfiable. Otherwise, new instances should be generated by applying the inference rule called Inst-Gen, and for the set of first-order clauses with the newly generated instances, redo the abstraction, until either an unsatisfiable clauses is generated or return satisfiable when the set of clauses cannot be refined further.

Moreover, a complete saturation algorithm for ordered resolution is implemented in iProver, based on the same data structures as Inst-Gen Loop. In the saturation algorithm, a number of simplifications such as forward and backward subsumption, forward and backward subsumption resolution, tautology deletion and global subsumption are implemented.

iProver accepts cnf problems of TPTP [Sut09] format. For example, the clauses in Example 2.2 can be written as follows.

```
cnf(c1, negated_conjecture, even(mul(X,Y) | ~even(X))).
cnf(c2, negated_conjecture, even(mul(X,Y) | ~even(Y))).
cnf(c3, negated_conjecture, even(s(s(Z))) | even(Z)).
cnf(c4, negated_conjecture, even(zero)).
cnf(c5, negated_conjecture, ~even(mul(s(s(zero)), s(s(s(zero)))))).
```

Note that the predicates and function symbols are represented using lower words, while the variables are represented by words startting with a upper alphabeta. Assume that iProver is installed and all the clauses above are put in the file even.p, then the problem can be proved by inputting the command

```
iproveropt even.p
```

iProver Modulo

iProver Modulo is an implementation of Ordered Polarized Resolution Modulo, which is developed by Guillaume Burel. To represent the one-way clauses, the developer chose to change the semantics of the TPTP format whenever the one-way clauses are used. That is to say, when the command-line argument - -modulo is set to true, the clause whose role is axiom is understood as a one-way clause. In the clause whose role is axiom, the first literal is taken as the seletcted literal. For instance, the input of the problem in Example 2.3 can be represented as follows.

```
cnf(c1, axiom, even(mul(X,Y) | ~even(X))).
cnf(c2, axiom, even(mul(X,Y) | ~even(Y))).
cnf(c3, axiom, even(s(s(Z))) | even(Z)).
cnf(c4, axiom, even(zero)).
cnf(c5, negated_conjecture, ~even(mul(s(s(zero)), s(s(s(zero)))))).
```

Suppose the name of the input file is even-one-way.p, this example can be proved by inputting the command

```
iproveropt ‘cat basic_resolution_options‘ --modulo true
           --res_out_proof true even_one_way.p
```

2.4.2 VERDS

VERDS [Zha12] is a symbolic model checking tool developed by Wenhui Zhang. It is an integration of TBD-based model checking approach [Zha13], SAT-based bounded model checking approach [BCCZ99, BCC+03, Zha09], and QBF-based bounded model checking approach [Zha14]. The specification language of VERDS is CTL (eCTL for QBF-based bounded model checking). For the modeling language of VERDS, refer to [Zha12]. To explain how do bounded model checking with QBF, we present a temporal verification of simple mutual exclusion algorithm [Pet81], which is presentated in Figure 2.9.

```
bool flag[0]   = false;
bool flag[1]   = false;
int turn;

P0:                                |   P1:
  flag[0] = true;                  |     flag[1] = true;
  turn = 1;                        |     turn = 0;
  while (flag[1] && turn == 1)     |     while (flag[0] && turn == 0)
  {                                |     {
     // busy wait                  |        //busy wait
  }                                |     }
     // critical section           |        // critical section
     ...                           |        ...
     // end of critical section    |        // end of critical section
  flag[0] = false;                 |     flag[1] = false;
```

FIGURE 2.9: Mutual Exclusion Algorithm

In this algorithm, two variables, $flag$ and $turn$ are used. The value of $flag[n]$ is $true$ indicates that the process n is trying to enter the critical section. The value of $turn$ is n means that the process n has the priority to enter the critical section. Thus, entrance to the critical section is granted for process $P0$ if $P1$ does not want to enter the critical section or if $P1$ has given priority to $P0$ by setting $turn$ to 0.

The modeling of this algorithm and the specification of the temporal properties using VERDS is as follows.

```
VVM
VAR   f[0..1]: 0..1;
      t: 0..1;
INIT  f[0]=0; f[1]=0; t=0;
PROC  p0: p0m(f[],t,0);
      p1: p0m(f[],t,1);
```

```
SPEC  AG(!(p0.a=critical&p1.a=critical));
      AG((!p0.a=wait|AF(p0.a=critical|p1.a=critical))
        &(!p1.a=wait|AF(p0.a=critical|p1.a=critical)));
      AG((!p0.a=wait|AF(p0.a=critical))
        &(!p1.a=wait|AF(p1.a=critical)));
      AG((!p0.a=wait|EF(p0.a=critical))
        &(!p1.a=wait|EF(p1.a=critical)));

MODULE p0m(f[],t,i)
VAR   a: {start,wait,critical,noncritical};
INIT  a=start;
TRANS a=start:                (f[1-i],t,a):=(1,1-i,wait);
      a=wait&(f[i]=0|t=i):    (a):=(critical);
      a=wait&!(f[i]=0|t=i):   (a):=(wait);
      a=critical:             (f[1-i],a):=(0,noncritical);
      a=critical:             (a):=(critical);
      a=noncritical:          (f[1-i],t,a):=(1,1-i,wait);
```

Assume that the name of the file is mutex.vvm. To check the i-th property of this files, the command is as follows:

```
verds -QBF -ck i mutex.vvm
```

2.4.3 NuSMV

NuSMV [CCGR99] is a symbolic model checking tool developed jointly by FBK-IRST and Carnegie Mellon University. It permits to check finite state systems against the specifications in CTL and LTL. NuSMV is the first model checking tool based on Binary Decision Diagrams (BDDs) [McM93]. The modeling of the mutual exclusion algorithm in Figure 2.9 and the specification of the temporal properties using NuSMV is as follows.

```
MODULE main
VAR    s0: {start, noncritical, wait, critical};
       s1: {start, noncritical, wait, critical};
       f0: boolean;
       f1: boolean;
       turn: boolean;
       pr0: process prc(s0, s1, f0, f1, turn, FALSE);
```

```
       pr1: process prc(s1, s0, f1, f0, turn, TRUE);
ASSIGN init(turn) := FALSE;
SPEC   AG(!((s0 =critical)&(s1 =critical)))
SPEC   AG((!(s0=wait)|AF(s0=critical|s1=critical))
         &(!(s1=wait)|AF(s0=critical|s1=critical)))
SPEC   AG((!(s0=wait)|AF(s0=critical))&(!(s1=wait)|AF(s1=critical)));
SPEC   AG((!(s0=wait)|EF(s0=critical))&(!(s1=wait)|EF(s1=critical)));

MODULE prc(state0,state1,flag0,flag1,turn,turn0)
ASSIGN init(state0):=start;
next(state0):=
case
   (state0=start):{wait};
   (state0=wait)&(flag1=TRUE)&(turn=!turn0):wait;
   (state0=wait)&((flag1=FALSE)|(turn=turn0)):critical;
   (state0=critical):{critical,noncritical};
   (state0=noncritical):{start};
   TRUE:state0;
esac;
next(turn) :=
case
   turn = turn0 & state0 = start: !turn0;
   TRUE : turn;
esac;
next(flag0) :=
case
   state0 = start: TRUE;
   state0 = noncritical: FALSE;
   TRUE: flag0;
esac;
```

Suppose that the input in contained in the file mutex.smv. The readers can check the specifications using the following commend:

```
NuSMV mutex.smv
```

If the counterexamples are not needed, the commend can be replaced by

```
NuSMV -dcx mutex.smv
```

3

Propositional Encoding of Graph Problems

As was mentioned in the previous chapter, the aim of this dissertation is to address model checking problem using an off-the-shelf automated theorem prover. The models to be checked are normally abstracted as finite state machines, labeled transition systems, Kripke structures, ... All of them can be treated as a variation of directed graphs. The work in this chapter is to solve the graph traversal problems, that are used in model checking, with automated theorem provers.

Safety and liveness are two important problems in model checking. The safety property says that something "bad" will never happen and the liveness property says that something "good" will eventually happen. To prove a system is safe or not, we need to prove that all the accessible states are not "bad" or find out a finite path to the "bad" state. Then a problem is generated: *How do we know that we have visited all the states that are accessible?* For the liveness of a system, we need to find an infinite path such that all the states on the path are not "good" or on each infinite path, there exists a "good" state. Thus we should solve this problem: *How do we know that we have found an infinite path?* In this chapter, these two problems on directed graph with finite vertices are considered.

Outline of This Chapter In Section 3.1, the terminology used in this chapter are illustrated. Section 3.2 is the main work of this chapter, where the strategy of finding out a cycle and finding out all the accessible vertices from a given vertex are presented. Then in Section 3.3, the correctness of the strategies are proved. In Section 3.4, a selection function and a special subsumption rule are defined to improve the efficiency of the resolution method. Finally in Section 3.5, the implementation of our method on testing some random graphs is presented.

3.1 Basic Definitions

We denote a graph as $G = \langle V, E \rangle$, in which V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges of the graph. To express these problems, we consider a propositional language which contains two kinds of atomic propositions B_i and W_i for each natural number.

Definition 3.1 (Walk)**.** Let $G = \langle V, E \rangle$ be a graph. The sequence of vertices $l = s_0, s_1, \ldots, s_k$ is a walk if and only if $\forall 0 \leqslant i < k$, $(s_i, s_{i+1}) \in E$. The walk l is *closed* if and only if $\exists 0 \leqslant j \leqslant k$ such that $s_k = s_j$. The walk l is *blocked* if and only if s_k has no successors.

The method we proposed is inspired by graph traversal algorithms. In the following sections, we introduce some terminology inspired by graph traversal algorithms.

Definition 3.2 (Black and White Literals)**.** Let G be a graph and $\{s_1, ..., s_n\}$ be the set of all the vertices in G. For any $1 \leqslant i \leqslant n$, the literal B_i is called a *black* literal and the literal W_i is called a *white* literal.

Intuitively, the black literals are the vertices that have already been visited, while the white literals are the ones that are not visited yet.

Definition 3.3 (Original Clause)**.** Let G be a graph and $\{s_1, \ldots, s_n\}$ be the set of all the vertices in G. For each graph traversal problem starting from s_i $(1 \leqslant i \leqslant n)$, we define the *original* clause $\mathsf{ori}(s_i, G)$ as

$$B_i \vee W_1 \vee \cdots \vee W_n.$$

Definition 3.4 (Traversal Clause)**.** A clause with only white and black literals is called a *traversal clause*.

Definition 3.5 (Success Clause)**.** Let C be a traversal clause, if there is no i, such that both B_i and W_i are in C, then C is called a *success clause*.

Among the three kinds of clauses, the original clause is related to the starting point of the graph traversal algorithm, the traversal clause is the current process of the traveling, and the success clause means that the solution is found out and the traversal procedure can be finished. Obviously, the original clauses and success clauses are also traversal clauses.

3.2 Closed-walk and Blocked-walk Detection

In this section, we present two algorithms respectively to decide

1. starting from a given vertex of a graph, whether a closed walk exists or not.
2. starting from a given vertex of a graph, whether a blocked walk exists or not.

3.2.1 Encoding of Closed-walk Detection Problem

For this encoding, we view the graph as a set of rewrite rules, and the initial situation is denoted by the original clause.

E-coloring rule Let G be a graph and $V = \{s_1, \ldots, s_n\}$ be the set of all the vertices in G. For each pair of vertices $\langle s_i, s_j \rangle$ in V, if there exists an edge from s_i to s_j, then we formalize this edge as an *E-coloring rewrite rule*

$$W_i \hookrightarrow B_j.$$

Correspondingly, the one-way clause for the rewrite rule is $\underline{\neg W_i} \vee B_j$ (called *E-coloring clause*). The set of all the E-coloring clauses for graph G is denoted as $EC(G)$.

Resolution for Closed-walk Detection Let G be a graph and s be a vertex of G, then the problem of checking whether, starting from s, there exists a closed walk can be encoded as the set of clauses $\{\mathsf{ori}(s, G)\} \cup EC(G)$. By applying resolution rules among these clauses, a success clause can be derived, if and only if there exists a closed walk starting from s.

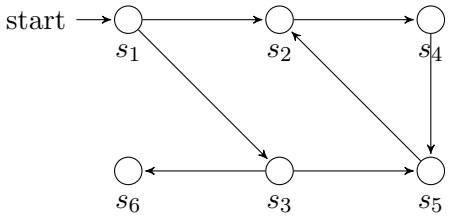


FIGURE 3.1: Closed-walk Detection Example

Example 3.1. *Consider the graph in Figure 3.1, check whether there exists a closed walk starting from s_1. For this problem, the original clause is*

$$B_1 \vee W_1 \vee W_2 \vee W_3 \vee W_4 \vee W_5 \vee W_6$$

and the set of E-coloring clauses for this graph are

$$\underline{\neg W_1} \vee B_2, \quad \underline{\neg W_1} \vee B_3, \quad \underline{\neg W_2} \vee B_4, \quad \underline{\neg W_3} \vee B_5, \quad \underline{\neg W_3} \vee B_6, \quad \underline{\neg W_4} \vee B_5, \quad \underline{\neg W_5} \vee B_2.$$

Resolution steps *for the original clause, apply Resolution rule with E-coloring clause* $\underline{\neg W_1} \vee B_2$*, which yields*

$$B_1 \vee B_2 \vee W_2 \vee W_3 \vee W_4 \vee W_5 \vee W_6,$$

then apply Resolution rule with E-coloring clause $\underline{\neg W_2} \vee B_4$*, which yields*

$$B_1 \vee B_2 \vee B_4 \vee W_3 \vee W_4 \vee W_5 \vee W_6,$$

then apply Resolution rule with E-coloring clause $\underline{\neg W_4} \vee B_5$*, which yields*

$$B_1 \vee B_2 \vee B_4 \vee W_3 \vee B_5 \vee W_5 \vee W_6,$$

then apply Resolution rule with E-coloring clause $\underline{\neg W_5} \vee B_2$*, and the generated clause*

$$B_1 \vee B_2 \vee B_4 \vee W_3 \vee B_5 \vee W_6,$$

is a success clause, meaning that in Figure 3.1, there exists a closed walk starting from s_1*.*

3.2.2 Encoding of Blocked-walk Detection Problem

For this encoding strategy, the graph is represented by a set of rewrite rules. Unlike the E-coloring rules in the previous subsection, a coloring rule for this problem is from a vertex to *ll* its successors. The initial situation is denoted by the original clause.

A-coloring rule Let G be a graph and $V = \{s_1, \ldots, s_n\}$ be the set of vertices of G. For each vertex s_i in V, assume that starting from s_i, there are edges to $s_{i_1}, \ldots, s_{i_j}$, then we formalize the set of edges starting from s_i as an A-coloring rule

$$W_i \hookrightarrow B_{i_1} \vee \cdots \vee B_{i_j}.$$

Correspondingly, the one-way clause for the rewrite rule is $\underline{\neg W_i} \vee B_{i_1} \vee \cdots \vee B_{i_j}$ (called A-coloring clause). The set of all the A-coloring clauses for graph G is denoted as $AC(G)$.

Resolution for Blocked-Walk Detection Let G be a graph and s be a vertex of G, then the the problem of checking that starting from s, whether there exists a blocked walk can be encoded as the set of clauses $\{\mathsf{ori}(s, G)\} \cup AC(G)$. By applying resolution rules among these clauses, a success clause can be derived, if and only if *there is no blocked walk* starting from s.

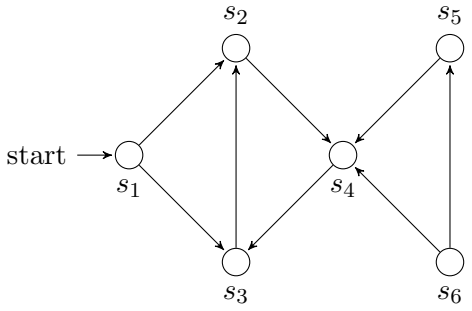


FIGURE 3.2: Block-walk Detection Example

Example 3.2. *Consider the graph in Figure 3.2 and the problem of whether there exists a blocked walk starting from s_1. For this problem, the original clause is*

$$B_1 \vee W_1 \vee W_2 \vee W_3 \vee W_4 \vee W_5 \vee W_6$$

and the set of A-coloring clauses for this graph are

$$\underline{\neg W_1} \vee B_2 \vee B_3, \quad \underline{\neg W_2} \vee B_4, \quad \underline{\neg W_3} \vee B_2, \quad \underline{\neg W_4} \vee B_3, \quad \underline{\neg W_5} \vee B_4, \quad \underline{\neg W_6} \vee B_4.$$

Resolution steps *For the original clause, apply Resolution rule with A-coloring clause $\underline{\neg W_1} \vee B_2 \vee B_3$, which yields*

$$B_1 \vee B_2 \vee B_3 \vee W_2 \vee W_3 \vee W_4 \vee W_5 \vee W_6,$$

then apply resolution rule with A-coloring clause $\underline{\neg W_2} \vee B_4$, which yields

$$B_1 \vee B_2 \vee B_3 \vee B_4 \vee W_3 \vee W_4 \vee W_5 \vee W_6,$$

then apply resolution rule with A-coloring clause $\underline{\neg W_3} \vee B_2$, which yields

$$B_1 \vee B_2 \vee B_3 \vee B_4 \vee W_4 \vee W_5 \vee W_6,$$

then apply resolution rule with A-coloring clause $\underline{\neg W_4} \vee B_3$, and the generated clause

$$B_1 \vee B_2 \vee B_3 \vee B_4 \vee W_5 \vee W_6,$$

is a success clause, meaning that there is no blocked walk starting from s_1.

3.3 Correctness of the Encoding Strategies

In this section, we prove the correctness of the two strategies for encoding of closed-walk and blocked-walk detection. Before proving the main theorems, several notations and lemmas are needed, which are used in the later proofs.

Notations Let C_1, C_2, C_3 be clauses, Γ be a set of clauses:

- if C_3 is generated by applying resolution between C_1 and C_2, then write the resolution step as $C_1 \xrightarrow{C_2} C_3$; if the resolution is based on a selection function δ, then the resolution step is written as $C_1 \xrightarrow{C_2}_{\delta} C_3$.
- if C_2 is generated by applying resolution between C_1 and a clause in Γ, then write the resolution step as $C_1 \xrightarrow{\Gamma} C_2$; if the resolution is based on a selection function δ, then the resolution step is written as $C_1 \xrightarrow{\Gamma}_{\delta} C_2$.
- if C_1 is generated by one step of resolution on some clauses in Γ, then write the resolution step as $\Gamma \longrightarrow \Gamma, C_1$; if the resolution is based on a selection function δ, then the resolution step is written as $\Gamma \longrightarrow_{\delta} \Gamma, C_1$.

Lemma 3.6. *For any two traversal clauses, we cannot apply resolution rules between them.*

Proof. All the literals in traversal clauses are positive. □

Lemma 3.7. *If resolution rules can be applied between a traversal clause and a coloring clause, then one and only one traversal clause can be derived.*

Proof. As all the literals in the traversal clause are positive and there is only one negative literal in the coloring clause, straightforwardly, only one traversal clause can be derived. □

Proposition 3.8. *Let* M *be a set of coloring clauses,* $C_1, \ldots, C_n$ *be traversal clauses and* S *be a success clause. If* $M, C_1, \ldots, C_n \mapsto S$*, then there exists* $1 \leqslant i \leqslant n$*, such that* $M, C_i \mapsto S$*, and the length of the later derivation is at most equal to the former one.*

Proof. By induction on the size of the derivation $M, C_1, \ldots, C_n \mapsto S$.

- If S is a member of $C_1, \ldots, C_n$, then there exists the derivation $M, S \mapsto S$ without applying any resolution rules.
- If S is not a member of $C_1, \ldots, C_n$, then in each step of the derivation, by Lemma 3.6, the resolution rules can only be applied between a traversal clause and a coloring clause. Assume the derivation is $M, C_1, \ldots, C_n \longrightarrow M, C_1, \ldots, C_n, C' \mapsto S$, in which, by Lemma 3.7, C' is a traversal clause. Then for the derivation $M, C_1, \ldots, C_n, C' \mapsto S$, by induction hypothesis, $M, C' \mapsto S$ or there exists $1 \leqslant i \leqslant n$ such that $M, C_i \mapsto S$, with the steps of the derivation at most equal to $M, C_1, \ldots, C_n, C' \mapsto S$. If $M, C_i \mapsto S$, then the steps of the derivation are less than $M, C_1, \ldots, C_n \mapsto S$, thus this derivation is as needed. If $M, C' \mapsto S$, then by Lemma 3.6, there exists C_j in $C_1, \ldots, C_n$, such that $C_j \xrightarrow{M} C'$, thus the derivation $M, C_j \mapsto S$, with the derivation steps at most equal to $M, C_1, \ldots, C_n \mapsto S$, is as needed.

□

Proposition 3.9. *Let M be a set of coloring clauses, C be a traversal clause, and S be a success clause. If $M, C \mapsto S(\pi_1)$[1], then there exists a derivation path $C(C_0) \xrightarrow{M} C_1 \xrightarrow{M} C_2 \cdots \xrightarrow{M} C_n(S)$.*

Proof. By induction on the size of the derivation π_1.

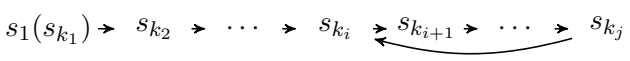
- If C is a success clause, then the derivation path can be built directly.
- Otherwise, by Lemma 3.6, in each step of the derivation, the resolution rules can only be applied between a traversal clause and a coloring clause. Assume the derivation is $M, C \longrightarrow M, C, C' \mapsto S$, then for the derivation $M, C, C' \mapsto S$, by Proposition 3.8, there exists a derivation $M, C \mapsto S(\pi_2)$[2] or $M, C' \mapsto S$, with the length less than π_1. For π_2, by induction hypothesis, there exists a derivation path $C(C_0) \xrightarrow{M} C_1 \cdots \xrightarrow{M} C_n(S)$, and this is just the derivation as needed. For $M, C' \mapsto S$, by induction hypothesis, there exists a derivation path $C' \xrightarrow{M} C'_1 \cdots \xrightarrow{M} C'_m(S)$. As $C \xrightarrow{M} C'$, the derivation path $C \xrightarrow{M} C' \xrightarrow{M} C'_1 \cdots \xrightarrow{M} C'_m(S)$ is as needed.

□

[1] we denote the derivation as π_1.
[2] we denote the derivation as π_2.

Theorem 3.10. *Let G be a graph and s be a vertex in G. Starting from s, there exists a closed walk if and only if starting from $\{\mathsf{ori}(s, G)\} \cup EC(G)$, a success clause can be derived.*

Proof. **For the right direction**, we assume that the path is



By the method of generating E-coloring clauses of a graph, there exist E-coloring clauses:

$$\underline{\neg W_{k_1}} \vee B_{k_2}, \underline{\neg W_{k_2}} \vee B_{k_3}, \ldots, \underline{\neg W_{k_{i-1}}} \vee B_{k_i}, \underline{\neg W_{k_i}} \vee B_{k_{i+1}}, \ldots, \underline{\neg W_{k_j}} \vee B_{k_i}.$$

Then starting from the original clause $C_1 = B_1 \vee W_1 \vee \cdots \vee W_n$, the derivation

$$C_1 \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_{i-1} \xrightarrow{D_{i-1}} C_i \xrightarrow{D_i} \cdots C_j \xrightarrow{D_j} C_{j+1}$$

can be built, in which C_{j+1} is a success clause and for each $1 \leqslant m \leqslant j$, D_m is the E-coloring clause $\underline{\neg W_{k_m}} \vee B_{k_{m+1}}$.

For the left direction, by Proposition 3.9, starting from the original clause $C_1 = B_1 \vee W_1 \vee \cdots \vee W_n$, there exists a derivation path

$$C_1 \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_{i-1} \xrightarrow{D_{i-1}} C_i \xrightarrow{D_i} \cdots C_j \xrightarrow{D_j} C_{j+1},$$

in which C_{j+1} is a success clause and for each $1 \leqslant m \leqslant j$, D_m is an E-coloring clause. As C_{j+1} is a success clause, for each black literal B_i in the clause C_{j+1}, there exists an E-coloring clause $\underline{\neg W_i} \vee B_{k_i}$ in $D_1, \ldots, D_j$. Thus for each black literal B_i in the clause C_{j+1}, there exists a vertex s_{k_i} such that there is an edge from s_i to s_{k_i}. As the number of black literals in C_{j+1} is finite, for each vertex s_i, if B_i is a member of C_{j+1}, then starting from s_i, there exists a path which contains a cycle. As the literal B_1 is in C_{j+1}, starting from s_1, there exists a path to a cycle. □

Before proving the correctness of the strategy for block walk detection, one more lemma is needed.

Lemma 3.11. *Let G be a graph and s_1 be a vertex of G. Starting from s_1, if all the reachable vertices are traversed in the order $s_1, s_2, \ldots, s_k$ and each reachable vertex has at least one successor, then starting from $\{\mathsf{ori}(s_1, G)\} \cup AC(G)$, there exists a derivation*

path $C_1(\mathsf{ori}(s_1, G)) \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_k \xrightarrow{D_k} C_{k+1}$, *in which* C_{k+1} *is a success clause and* $\forall 1 \leqslant i \leqslant k$, D_i *is an A-coloring clause of the form* $\underline{\neg W_i} \vee B_{i_1} \vee \cdots \vee B_{i_j}$.

Proof. As $s_1, s_2 \ldots, s_k$ are all the reachable vertices starting from s_1, for a vertex s, if there exists an edge from one of the vertices in $s_1, s_2, \ldots, s_k$ to s, then s is a member of $s_1, s_2, \ldots, s_k$. Thus, after the derivation $C_1 \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_j \xrightarrow{D_j} C_{j+1}$, for each black literal B_i, the white literal W_i is not in C_{j+1}, thus C_{j+1} is a success clause. □

Theorem 3.12. *Let* G *be a graph and* s_1 *be a vertex of* G. *Starting from* s_1, *there is no blocked walk if and only if, starting from* $\{\mathsf{ori}(s_1, G)\} \cup AC(G)$, *a success clause can be derived.*

Proof. **For the right direction**, assume that all the reachable vertices starting from s_1 are traversed in the order $s_1, s_2, \ldots, s_k$. For the resolution part, by Lemma 3.11, starting from the original clause, a success clause can be derived.

For the left direction, by Proposition 3.9, starting from the original clause $C_1 = \mathsf{ori}(s_1, G)$, there exists a derivation path

$$C_1 \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_j \xrightarrow{D_j} C_{j+1},$$

in which C_{j+1} is a success clause and $\forall 1 \leqslant i \leqslant j$, D_i is an A-coloring clause with $\neg W_{k_i}$ underlined. As there is no i such that both B_i and W_i are in C_{j+1}, for the vertices in $s_{k_1}, s_{k_2}, \ldots, s_{k_j}$, the successors of each vertex is a subset of $s_{k_1}, s_{k_2}, \ldots, s_{k_j}$. As the black literal B_1 is in the clause C_{j+1}, by the definition of success clause, the white literal W_1 is not in C_{j+1}, thus s_1 is a member of $s_{k_1}, s_{k_2}, \ldots, s_{k_j}$. Then recursively, for each vertex s, if s is reachable from s_1, then s is in $s_{k_1}, s_{k_2}, \ldots, s_{k_j}$. Thus starting from s_1, all the vertices reachable have successors. □

3.4 Simplification Rules

A drawback of the traditional automatic theorem proving methods is that they are only practical for graphs of relatively small size. In this section, we analyze the reason why the method is not as efficient as traditional traversal methods. To address the problems of our strategies, we design some new simplification rules. Finally the completeness of the system with new rules is proved.

3.4.1 Selection Function

We define a selection function, which applies on a traversal clause and returns a set of literals that have priority when applying resolution rules. We show that the number of resolution steps strongly depend on the literals that are selected. More precisely, the number of literals that are selected will also affect the number of resolution steps.

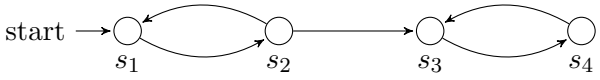


FIGURE 3.3: Example for Selection Function-1

Example 3.3. *For the graph in Figure 3.3, we prove the property:*

starting from s_1, there exists a closed walk.

The original clause of the graph is

$$B_1 \vee W_1 \vee W_2 \vee W_3 \vee W_4,$$

and the E-coloring clauses of the graph are

$$\underline{\neg W_1} \vee B_2, \quad \underline{\neg W_2} \vee B_1, \quad \underline{\neg W_2} \vee B_3, \quad \underline{\neg W_3} \vee B_4, \quad \underline{\neg W_4} \vee B_3.$$

Starting from the original clause, we can apply resolution as follows: First, apply resolution with E-coloring clause $\underline{\neg W_1} \vee B_2$*, which yields*

$$B_1 \vee B_2 \vee W_2 \vee W_3 \vee W_4. \tag{3.1}$$

Then for (3.1), apply resolution with E-coloring clause $\underline{\neg W_2} \vee B_1$*, which yields*

$$B_1 \vee B_2 \vee W_3 \vee W_4. \tag{3.2}$$

The clause (3.2) is a success clause. However, from (3.1), if we apply resolution with another E-coloring clause instead, we will need more resolution steps to get a success clause.

By the definition of success clause, the pair of literals B_i and W_i cannot both be members of the traversal clause. Thus for the pair of literals B_i and W_i in a traversal clause, the idea of selecting W_i to apply resolution rules will at least not increase the total number of resolution steps to get a success clause. Otherwise, from Example 3.3, we can see that some useless steps of resolution may be involved in derivation.

Definition 3.13 (Grey literals)**.** Let C be a traversal clause. For the pair of white literals and black literals $\langle W_i, B_i \rangle$, if both W_i and B_i are the members of C, then W_i is called a *grey literal* of C. The set of grey literals of C is defined as follows:

$$\mathsf{grey}(C) = \{W_i \mid \text{both } B_i \text{ and } W_i \text{ are in } C\}$$

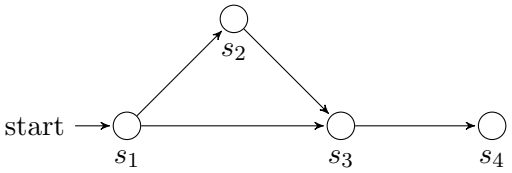


FIGURE 3.4: Example for Selection Function-2

Example 3.4. *For the graph in Figure 3.4, we prove the property:*

starting from s_1, there is no blocked walk.

The original clause is

$$B_1 \vee W_1 \vee W_2 \vee W_3 \vee W_4,$$

and the A-coloring clauses of the graph are

$$\underline{\neg W_1} \vee B_2 \vee B_3, \quad \underline{\neg W_2} \vee B_3, \quad \underline{\neg W_3} \vee B_4$$

Resolution steps *For the original clause, apply resolution with A-coloring clause $\underline{\neg W_1} \vee B_2 \vee B_3$, which yields*

$$B_1 \vee B_2 \vee B_3 \vee W_2 \vee W_3 \vee W_4. \tag{3.3}$$

Then for (3.3), we can apply resolution rules with A-coloring clauses $\underline{\neg W_2} \vee B_3$ and $\underline{\neg W_3} \vee B_4$, and two new traversal clauses are generated:

$$B_1 \vee B_2 \vee B_3 \vee W_3 \vee W_4, \tag{3.4}$$

$$B_1 \vee B_2 \vee B_3 \vee B_4 \vee W_2 \vee W_4. \tag{3.5}$$

Then for (3.4), apply resolution rule with A-coloring clause $\underline{\neg W_3} \vee B_4$, which yields

$$B_1 \vee B_2 \vee B_3 \vee B_4 \vee W_4, \tag{3.6}$$

and for this clause, we cannot apply resolution rules any more. For (3.5), we can apply resolution rule with A-coloring clause $\underline{\neg W_2} \vee B_3$, and the clause generated is the same

as (3.6). Obviously, the resolution steps for generating (3.5) and the steps started from (3.5) are redundant.

To avoid the redundant steps in Example 3.4, each time we select only one grey literal. So the selection function can be defined as follows.

Definition 3.14 (Selection Function)**.** For any traversal clause C, the selection function δ is defined as:

$$\delta(C) = \begin{cases} \mathsf{single}(\mathsf{grey}(C)), & \mathsf{grey}(C) \neq \emptyset \\ C, & \text{Otherwise} \end{cases}$$

in which single is a process to select only one literal from a set of literals.

Notations The *Polarized Resolution Modulo with Selection Function* δ is written as PRM^{δ}. We write $\Gamma \mapsto^{\delta}_{\mathcal{R}} C$ if the clause C can be derived from the set of clauses Γ in the system $PRM^{\delta}_{\mathcal{R}}$.

3.4.2 Elimination Rule

As we shall see in the following example, selecting literals at the base of PRM, OR, OPRM, and this method are not sufficient. We also have to restrict the method at the level of clauses. In spite of several clause elimination procedures, for instance tautology elimination, subsumption elimination, etc., had been applied to the procedure of resolution method [HJB10], none of them works efficiently to our problem.

Example 3.5. *For the graph in Figure 3.4, we prove the property:*

starting from s_1, there exists a closed walk.

The original clause is

$$B_1 \vee W_1 \vee W_2 \vee W_3 \vee W_4,$$

and the E-coloring clauses of the graph are

$$\underline{\neg W_1} \vee B_2, \quad \underline{\neg W_1} \vee B_3, \quad \underline{\neg W_2} \vee B_3, \quad \underline{\neg W_3} \vee B_4$$

Resolution steps *For the original clause, apply resolution rules with* $\underline{\neg W_1} \vee B_2$ *and* $\underline{\neg W_1} \vee B_3$*, two new traversal clauses are generated:*

$$B_1 \vee B_2 \vee W_2 \vee W_3 \vee W_4, \tag{3.7}$$

$$B_1 \vee B_3 \vee W_2 \vee W_3 \vee W_4, \tag{3.8}$$

for (3.7), apply resolution rule with $\underline{\neg W_2} \vee B_3$*, which yields*

$$B_1 \vee B_2 \vee B_3 \vee W_3 \vee W_4, \tag{3.9}$$

then for (3.9), apply resolution rule with $\underline{\neg W_3} \vee B_4$*, which yields*

$$B_1 \vee B_2 \vee B_3 \vee B_4 \vee W_4, \tag{3.10}$$

the clause (3.10) cannot be reduced any more. Note that for all the clauses that are generated, only the traversal clause (3.8) remains to be resolved. For (3.8), we can apply reolution rule with $\underline{\neg W_3} \vee B_4$*, this yields*

$$B_1 \vee B_3 \vee W_2 \vee B_4 \vee W_4 \tag{3.11}$$

which has the same selected literal as (3.10). Thus, this step of resolution is redundant.

To avoid the likewise redundant steps showed in Example 3.5, a new subsumption rule is defined as follows.

Definition 3.15 (Path Subsumption Rule(PSR))**.** Let M be a set of A(E)-coloring clauses and C be a traversal clause. If we have $C, M \mapsto^{\delta}_{\mathcal{R}} C_1$ and $C, M \mapsto^{\delta}_{\mathcal{R}} C_2$, if $\mathsf{grey}(C_1) = \mathsf{grey}(C_2)$, the rule below can be used:

$$\frac{C_1 \quad C_2}{C_i}\ \mathsf{grey}(C_1) = \mathsf{grey}(C_2), i = 1 \text{ or } 2$$

meaning that one of the two clauses can be deleted, without breaking the final result.

After each step of resolution, we try to apply PSR on the set of traversal clauses before applying other resolution rules. By PSR, the clause (3.8) in Example 3.5 is redundant, thus will be deleted during the derivation.

3.4.3 Completeness

For the completeness of our method, we first prove that PRM^{δ} is complete, then we prove that PRM^{δ} remains complete when we apply PSR eagerly.

Proposition 3.16 (Completeness of PRM^{δ})**.** *Let M be a set of A(E)-coloring clauses and $C_1, \ldots, C_n$ be traversal clauses. If $M, C_1, \ldots, C_n \mapsto S$, in which the clause S is a success clause, then starting from $M, C_1, \ldots, C_n$, we can build a derivation by selecting the resolved literals with selection function δ in Definition 3.14 and get a success clause.*

Proof. By Proposition 3.8 and Proposition 3.9, there exists $1 \leqslant i \leqslant n$, such that $C_i(C_{i_0}) \xrightarrow{D_1} C_{i_1} \cdots \xrightarrow{D_n} C_{i_n}(S)$. As there are no white literals in any clauses of $D_1, \ldots, D_n$ and in each step of the resolution, the resolved literal in the traversal clause is a white literal, the order of white literals to be resolved in the derivation by applying Resolution rule with coloring clauses in $D_1, \ldots, D_n$ will not affect the result. Thus use selection function δ to select white literals to be resolved, until we get a traversal clause S' such that there are no grey literals in it. By the definition of success clause, S' is a success clause. □

Lemma 3.17. *Let M be a set of A(E)-coloring clauses and C be a traversal clause. Assume $C(H_0) \xrightarrow{D_1}_{\delta} H_1 \xrightarrow{D_2}_{\delta} \cdots H(H_i) \xrightarrow{D_i}_{\delta} \cdots \xrightarrow{D_n}_{\delta} H_n$ in which H_n is a success clause and for each $1 \leqslant j \leqslant n$, the coloring clause D_j is in M, and $M, C \mapsto^{\delta} K$ such that $\mathsf{grey}(H) = \mathsf{grey}(K)$. If $K, D_1, \ldots, D_n \mapsto^{\delta} K'$, and K' is not a success clause, then there exists a coloring clause D_k in $D_1, \ldots, D_n$ and a traversal clause K'', such that $K' \xrightarrow{D_k}_{\delta} K''$.*

Proof. As K' is not a success clause, assume that the literals B_i and W_i are in K'. As W_i cannot be introduced in each step of resolution between a traversal clause and a coloring clause, W_i is in C and K. As the literal B_i is in clause K', during the derivation of K', there must be some clauses which contains B_i:

- if the literal B_i is in K, as W_i is also in K, W_i is a grey literal of K. As $\mathsf{grey}(H) = \mathsf{grey}(K)$, the literal B_i is also in H, and as B_i cannot be selected during the derivation, it remains in the traversal clauses $H_{i+1}, \ldots, H_n$.
- if the literal B_i is introduced by applying Resolution rule with coloring clause D_j in $D_1, \ldots, D_n$, which is used in the derivation of H_n as well, so the literal B_i is also a member of H_n.

In both cases, the literal B_i is in H_n. As H_n is a success clause, the literal W_i is not a member of H_n. As W_i is in C, there exists a coloring clause D_k in $D_1, \ldots, D_n$ with the literal $\neg W_i$ selected. Thus, $K' \xrightarrow{D_k}_\delta K''$. □

Lemma 3.18. *Let M be a set of A(E)-coloring clauses and C be a traversal clause. If we have $M, C \mapsto^\delta H$ and $M, C \mapsto^\delta K$, such that $\mathsf{grey}(H) = \mathsf{grey}(K)$, then starting from M, H a success clause can be derived if and only if starting from M, K a success clause can be derived.*

Proof. Without loss of generality, prove that if starting from M, H we can get to a success clause, then starting from M, K, we can also get to a success clause. By Proposition 3.9, starting from C, there exists $H_0(C) \xrightarrow{M}_\delta H_1 \xrightarrow{M}_\delta \cdots H_i(H) \xrightarrow{M}_\delta \cdots \xrightarrow{M}_\delta H_n$, in which H_n is a success clause. More precisely, $H_0(C) \xrightarrow{D_1}_\delta H_1 \xrightarrow{D_2}_\delta \cdots H_i(H) \xrightarrow{D_{i+1}}_\delta \cdots \xrightarrow{D_n}_\delta H_n$, where for each $1 \leqslant j \leqslant n$, the coloring clause D_j is in M. Then by Lemma 3.17, starting from M, K, we can always find a coloring clause in $D_1, \ldots, D_n$ to apply resolution with the new generated traversal clause, until we get a success clause. As the white literals in the generated traversal clauses decrease by each step of resolution, we will eventually get a success clause. □

Theorem 3.19 (Completeness)**.** *PRM_δ with PSR is complete.*

Proof. By Lemma 3.18, each time after we apply PSR, the satisfiability is preserved. □

3.5 Implementation

In this section, we discuss the implementation issues, and then present the evaluation data for some graphs.

3.5.1 How to deal with success clause

In normal resolution based proof search algorithms, the derivation will not stop until (i) an empty clause is derived, in this case the input set of clauses is unsatisfiable or (ii) no new clauses can be generated by applying resolution rules, in this case the input set of clauses is satisfiable. However, for the specific problems in this paper, the derivation should stop when a success clause is derived, which is different from "Satisfiable" or "Unsatisfiable". To implement our method in automatic theorem provers, there may have two ways to deal with the success clauses:

- give a set of rewrite rules, when a success clause is derived, make sure that this clause can be rewritten into empty clause.
- take success clause as the same role of empty clause, in this case when a success clause is derived, the derivation stop and report the input set of clauses is unsatisfiable.

For the first case, one way is to introduce class variables and take the atomic propositions B_i and W_i as binary predicates. Thus B_i is replaced by $B(s_i, Y)$ and W_i is replaced by $W(s_i, Y)$. Thus the success clause

$$B_1 \vee B_2 \vee \cdots \vee B_i \vee W_{i+1} \vee \cdots \vee W_k$$

is replaced by

$$B(s_1, Y) \vee B(s_2, Y) \vee \cdots \vee B(s_i, Y) \vee W(s_{i+1}, Y) \vee \cdots \vee W(s_k, Y).$$

The rewrite rules added are

1. $B(x, add(y, Z)) \hookrightarrow \neg x = y \wedge B(x, Z)$
2. $W(x, nil) \hookrightarrow \bot$
3. $W(x, add(y, Z)) \hookrightarrow x = y \vee W(x, Z)$
4. $x = x \hookrightarrow T$
5. for each two vertices s_i and s_j, if they are not the same vertex, then $s_i = s_j \hookrightarrow \bot$

This method is a variation of the theory defined in [DJ13b]. The main problem of this method is that, for any two different vertices in a graph, a rewrite rule should be added to the system to express the non-equalities.

For the second case, a procedure to check whether a clause is a success clause should be added to the loop-body of the program. For the position where to embed this procedure, a simple proof search algorithm is given as follows:

```
program main_loop
  initial
    original clause in U, A(E)-coloring clauses in P
```

```
  while U != empty
   c := select(U)
   U := U \ {c} (* remove c from U *)
   if c is an empty or a success clause, then return "Unsat"
   P := P + {c} (* add c to P *)
   U := U + generate(c,P)
  done
  return "Sat"
end.
```

where `select(U)` selects a clause from `U`, `grey(c)` is the set of grey literals in `c` and `generate(c,P)` produces all the clauses by applying an inference rule between `c` and a clause in `P`.

3.5.2 Embedding path subsumption rule into the proof-search algorithm

Normally, to run the path subsumption rule, each time before applying resolution rules between the selected traversal clause in the passive set `U` and the coloring clauses in the active set `P`, we need to give a comparison between the selected clause and each traversal clause in `P`. To make it simple, before the loop part for the resolution steps, a new empty set `G` is given, and for the selected traversal clause in `U`, if the grey literal of the traversal clause are in `G`, then just add the clause to the active set, otherwise, add the grey literal to `G` and apply resolution between this clause and the coloring clauses.

Algorithm By adding path subsumption rule into the algorithm above, the new algorithm is as follows:

```
program main_loop
  initial
    original clause in U, coloring clauses in P
    G is empty (* G is a set of sets of grey literals *)
  while U != empty
   c := select(U)
   U := U \ {c} (* remove c from U *)
   if c is an empty or a success clause, then return "Unsat"
   g := delta(c) (* delta is the literal selection function *)
```

```
   if g is not a member of G then
      P := P + {c} (* add c to P *)
      G := G + {g}
      U := U + generate(c,P)
  done
  return "Sat"
end.
```

3.5.3 Experimental Evaluation

The procedure of checking success clauses, the selection function, and the path subsumption rule are embedded into iProver modulo [Bur10]. The data of the experiments on some randomly generated graphs are illustrated in Table 3.1.

TABLE 3.1: Closed-walk and Blocked-walk Detection Results

Graph				Result and Time			
Prop	**N(v)**	**N(e)**	**Num**	**Sat**	**Succ**	PRM_δ	PRM_δ+PSR
Closed Walk	1.0×10^3	1.0×10^3	100	95	5	25m40s	25m0s
	1.0×10^3	1.5×10^3	100	50	50	1h06m40s	1h02m46s
	1.0×10^3	2.0×10^3	100	23	77	1h09m44s	1h09m46s
Blocked Walk	1.0×10^3	1.0×10^3	100	100	0	7m04s	
	1.0×10^3	1.5×10^3	100	100	0	10m29s	
	1.0×10^3	2.0×10^3	100	100	0	17m48s	
	1.0×10^3	2.5×10^3	100	100	0	35m16s	
	1.0×10^3	3.0×10^3	100	100	0	1h06m28s	
	1.0×10^3	1.0×10^4	100	0	100	24h50m43s	

For the closed-walk detection problem, by applying PSR, the total time in all the 100 graphs does not reduce. By checking the running time of each graph, we find that in most of the testing cases, PSR is inactive, because most of the vertices do not have the chance to be visited again. Thus, the time saved by applying PSR was offset by the time wasted in running this rule. For the blocked walk detection, the running time increases while we have more edges in the graphs, that is because the more edges in the graphs, the more vertices can be visited.

3.6 Summary

In this chapter, two graph problems, closed walk and blocked walk detection, are considered. To make it simple, we encoded the problems with propositional formulas, and

the edge relationship are encoded as rewrite rules. To improve the efficiency of the implementation, a selection function and a new subsumption elimination rule are defined. At last, an implementation about solving these two problems is presented.

At the beginning of this chapter, we mentioned that when checking the safety of a transition system, all the states of the system that are accessible from the initial state should be traversed. As each state in the system has at least one successor (refer to the definition of Kripke structure), this problem can be treated as a blocked-walk detection problem, and a success clause can be derived when all the accessible states are visited. For the liveness, we need to find an infinite "bad" path, thus can be treated as a closed-walk detection problem. An infinite path (closed walk) is found out when a success clause is derived.

As the number of literals in the original clause is equal to the number of vertices in the graph, if the graph is large enough, the space resources during the implementation will be run out. In spite of [G.S83] had given the idea of introducing new atoms as abbreviations or 'definitions' for sub-formulas, this cannot be used directly to our case. In the next chapter, we will encode the vertices with boolean vectors.

4 CTL Model Checking in Deduction Modulo

In this chapter, we express Branching-time temporal logic (CTL) [CGP99] for a given finite transition system in Deduction Modulo [DHK03, Dow10]. This way, the proof-search algorithms designed for Deduction Modulo, such as Resolution Modulo [Bur10] or Tableaux Modulo [DDG+13], can be used to build proofs in CTL.

Outline of This Chapter In Section 4.1, an alternative new semantics for CTL on finite structures is given. In Section 4.2, the rewrite rules for each CTL operator are given and the soundness and completeness of this presentation of CTL are proved, using the semantics presented in the previous section.

4.1 Alternative Semantics

In this section we develop an alternative semantics of CTL using finite paths only. In the traditional semantics of CTL, the semantics of some temporal propositions are expressed with infinite paths. However, in deduction modulo, the infinite paths cannot be expressed directly. Thus, an alternative semantics of CTL on finite models, in which all the temporal propositions are expressed with finite paths, is given. Then we prove that the alternative semantics are logically equal with the traditional semantics of CTL.

4.1.1 Paths with the Last State Repeated

A finite state system can be represented by a Kripke structure, which is a transition system. It is used in model checking to represent the behavior of a system.

Definition 4.1 (Kripke Structure)**.** Let AP be a set of atomic formulas. A Kripke structure M over AP is a three tuple $M = (S, \mathsf{next}, \mathsf{L})$ where

- S is a finite set of states.
- $\mathsf{next} : S \to \mathcal{P}^+(S)$ is a function that gives each state a (non-empty) set of successors.
- $\mathsf{L} : S \to \mathcal{P}(AP)$ is a function that labels each state with a subset of AP.

Paths with the Last State Repeated (lsr-paths) A finite path is a *lsr-path* if and only if the last state on the path occurs twice. For instance s_0, s_1, s_0 is a lsr-path. Note that we use $\rho = \rho_0\rho_1 \dots \rho_j$ to denote a lsr-path. A lsr-path ρ with $\rho_0 = s$ is denoted as $\rho(s)$, with $\rho_i = \rho_j$ is denoted as $\rho(i, j)$. The length of a path l is expressed by $\mathsf{len}(l)$ and the concatenation of two paths l_1, l_2 is $l_1 \hat{} l_2$.

Lemma 4.2 (From infinite paths to lsr-paths and vice-versa)**.** *Let M be a Kripke structure.*

1. *If π is an infinite path of M, then $\exists i \geq 0$ such that π_0^i is a lsr-path.*
2. *If $\rho(i, j)$ is a lsr-path of M, then $\rho_0^i \hat{} (\rho_{i+1}^j)^\omega$ is an infinite path.*

Proof. For the first case, as M is finite, there exists at least one state in π which occurs twice. If π_i is the first state which occurs twice, then π_0^i is a lsr-path. The second case is trivial. □

Lemma 4.3 (The reachibility between two states by lsr-paths)**.** *Let M be a Kripke structure.*

1. *For the path $l = s_0, s_1, \dots, s_k$, there exists a path $l' = s'_0, s'_1, \dots, s'_i$, in which no state occurs twice, such that $s'_0 = s_0$, $s'_i = s_k$, and $\forall 0 < j < i$, s'_j is on l.*
2. *If there is a path from s to s', then there exists a lsr-path $\rho(s)$ such that s' is on ρ.*

Proof. For the first case, l' can be built by deleting the cycles from l. The second case is straightforward by the first case and Lemma 4.2.

□

4.1.2 Alternative Semantics

Based on the definition of lsr-paths, the alternative semantics of CTL is given below.

Definition 4.4 (Alternative Semantics of CTL)**.** Let p be an atomic formula. Let $\varphi, \varphi_1, \varphi_2$ be CTL formulas. $M, s \models_a \varphi$ is defined inductively on the structure of φ as follows:

$M, s \models_a p \Leftrightarrow p \in L(s)$.

$M, s \models_a \neg\varphi_1 \Leftrightarrow M, s \not\models_a \varphi_1$.

$M, s \models_a \varphi_1 \wedge \varphi_2 \Leftrightarrow M, s \models_a \varphi_1$ and $M, s \models_a \varphi_2$.

$M, s \models_a \varphi_1 \vee \varphi_2 \Leftrightarrow M, s \models_a \varphi_1$ or $M, s \models_a \varphi_2$.

$M, s \models_a AX\varphi_1 \Leftrightarrow \forall s' \in \mathsf{next}(s),\ M, s' \models_a \varphi_1$.

$M, s \models_a EX\varphi_1 \Leftrightarrow \exists s' \in \mathsf{next}(s),\ M, s' \models_a \varphi_1$.

$M, s \models_a AF\varphi_1 \Leftrightarrow \forall \rho(s),\ \exists 0 \leq i < \mathsf{len}(\rho) - 1$ s.t. $M, \rho_i \models_a \varphi_1$.

$M, s \models_a EF\varphi_1 \Leftrightarrow \exists \rho(s),\ \exists 0 \leq i < \mathsf{len}(\rho) - 1$ s.t. $M, \rho_i \models_a \varphi_1$.

$M, s \models_a AG\varphi_1 \Leftrightarrow \forall \rho(s),\ \forall 0 \leq i < \mathsf{len}(\rho) - 1,\ M, \rho_i \models_a \varphi_1$.

$M, s \models_a EG\varphi_1 \Leftrightarrow \exists \rho(s),\ \forall 0 \leq i < \mathsf{len}(\rho) - 1,\ M, \rho_i \models_a \varphi_1$.

$M, s \models_a AU(\varphi_1, \varphi_2) \Leftrightarrow \forall \rho(s),\ \exists 0 \leq i < \mathsf{len}(\rho) - 1$ s.t. $M, \rho_i \models_a \varphi_2$ and $\forall 0 \leq j < i$, $M, \rho_j \models_a \varphi_1$.

$M, s \models_a EU(\varphi_1, \varphi_2) \Leftrightarrow \exists \rho(s),\ \exists 0 \leq i < \mathsf{len}(\rho) - 1$ s.t. $M, \rho_i \models_a \varphi_2$ and $\forall 0 \leq j < i$, $M, \rho_j \models_a \varphi_1$.

$M, s \models_a AR(\varphi_1, \varphi_2) \Leftrightarrow \forall \rho(s),\ \forall 0 \leq i < \mathsf{len}(\rho) - 1$, either $M, \rho_i \models_a \varphi_2$ or $\exists 0 \leq j < i$ s.t. $M, \rho_j \models_a \varphi_1$.

$M, s \models_a ER(\varphi_1, \varphi_2) \Leftrightarrow \exists \rho(s),\ \forall 0 \leq i < \mathsf{len}(\rho) - 1$, either $M, \rho_i \models_a \varphi_2$ or $\exists 0 \leq j \leq i$ s.t. $M, \rho_j \models_a \varphi_1$.

Remark1 The translation between infinite paths and lsr-paths is not a bijection. For instance, from the infinite path $s_0, s_1, s_0, (s_2, s_3)^\omega$, the lsr-path s_0, s_1, s_0 is derivable, but from s_0, s_1, s_0, only $s_0, (s_1, s_0)^\omega$ can be constructed.

Remark2: Alternative Semantics vs. Bounded Semantics In bounded semantics of CTL [Zha09], the transition system M is refined to a k-Model $M_k = \langle S, Ph_k, L\rangle$ where Ph_k is the set of all different finite paths with length $k+1$. Obviously, when $k < |S|$, the bounded semantics of CTL looses the completeness. Even when a temporal property is satisfiable in the k-model, the alternative semantics also have advantage in the size of paths that are used to express the semantics. Let's look at the example as follows.

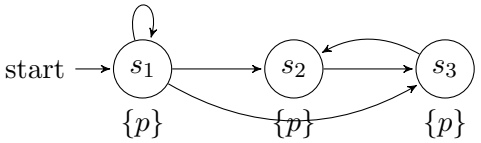


FIGURE 4.1: Semantics Comparison Example

Example 4.1. *For the Kripke structure in Figure 4.1. To prove that* $M, s_1 \models AGp$ *using bounded model checking, we need to prove that* p *holds on all the states in the paths of* Ph_3 *starting from* s_1 *(Figure 4.2). In alternative semantics, we only need to prove that* p *holds on the states in the lsr-paths of Figure 4.3.*

s_1	s_1	s_1	s_1
↓	↓	↓	↓
s_1	s_1	s_2	s_3
↓	↓	↓	↓
s_2	s_3	s_3	s_2
↓	↓	↓	↓
s_3	s_2	s_2	s_3

FIGURE 4.2: 3-Paths Starting from s_1

s_1	s_1	s_1
↓	↓	↓
s_1	s_2	s_3
	↓	↓
	s_3	s_2
	↓	↓
	s_2	s_3

FIGURE 4.3: Lsr-paths Starting from s_1

4.1.3 Soundness and Completeness

We now prove the soundness and completeness of the alternative semantics of CTL. The method is to prove the equivalence between the alternative semantics and the traditional

semantics of CTL mentioned in Section 2, that is, $M, s \models \varphi$ if and only if $M, s \models_a \varphi$. To simplify the proofs, all the CTL formulas are translated into negation normal form.

Lemma 4.5. *Let φ be a CTL formula of NNF. If $M, s \models \varphi$, then $M, s \models_a \varphi$.*

Proof. By induction on the structure of φ. The cases $\varphi = p$, $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $AX\varphi_1$, $EX\varphi_1$ are trivial. For the other cases, the proof is as follows.

- Let $\varphi = AF\varphi_1$. We prove the contrapositive. If there is a lsr-path $\rho(s)(j,k)$ such that $\forall 0 \leq i < k$, $M, \rho_i \not\models \varphi_1$, then by Lemma 4.2, there exists an infinite path $\rho_0^j{}^\frown(\rho_{j+1}^k)^\omega$, which is a counterexample of $M, s \models AF\varphi_1$. Thus for each lsr-path $\rho(s)$, $\exists 0 \leq i < \mathsf{len}(\rho) - 1$ such that $M, \rho_i \models \varphi_1$ holds. Then by induction hypothesis , for each lsr-path $\rho(s)$, $\exists 0 \leq i < \mathsf{len}(\rho) - 1$ such that $M, \rho_i \models_a \varphi_1$ holds, and thus $M, s \models_a AF\varphi_1$ holds.

- Let $\varphi = EF\varphi_1$. By the semantics of CTL, there exists an infinite path $\pi(s)$ and $\exists i \geq 0$ such that $M, \pi_i \models \varphi_1$ holds, and $M, \pi_i \models_a \varphi_1$ holds by induction hypothesis. Then by Lemma 4.3, there exists a lsr-path $\rho(s)$ such that π_i is on ρ, and thus $M, s \models_a EF\varphi_1$ holds.

- Let $\varphi = AG\varphi_1$. We prove the contrapositive. If there is a lsr-path $\rho(s)(j,k)$ and $\exists 0 \leq i < k$ such that $M, \rho_i \not\models \varphi_1$, then by Lemma 4.2, there exists an infinite path $\rho_0^j{}^\frown(\rho_{j+1}^k)^\omega$, which is a counterexample of $M, s \models AG\varphi_1$. Thus for each lsr-path $\rho(s)(j,k)$ and $\forall 0 \leq i < k$, $M, \rho_i \models \varphi_1$ holds. Then by induction hypothesis, for each lsr-path $\rho(s)(j,k)$ and $\forall 0 \leq i < k$, $M, \rho_i \models_a \varphi_1$ holds, and thus $M, s \models_a AG\varphi_1$ holds.

- Let $\varphi = EG\varphi_1$. By the semantics of CTL, there exists an infinite path $\pi(s)$ such that $\forall i \geq 0$, $M, \pi_i \models \varphi_1$ holds. Then by Lemma 4.2, $\exists k \geq 0$ such that π_0^k is a lsr-path and by induction hypothesis, $\forall 0 \leq i < k$, $M, \pi_i \models_a \varphi_1$ holds. Thus $M, s \models_a EG\varphi_1$ holds.

- Let $\varphi = AU(\varphi_1, \varphi_2)$. We prove the contrapositive. Assume that there exists a lsr-path $\rho(s)(l,k)$ such that $\forall 0 \leq i < k$, $M, \rho_i \not\models \varphi_2$ or $\forall 0 \leq i < k$, if $M, \rho_i \models \varphi_2$ holds, then $\exists 0 \leq j < i$, $M, \rho_j \not\models \varphi_1$. Then by Lemma 4.2, there exists an infinite path $\rho_0^l{}^\frown(\rho_{l+1}^k)^\omega$, which is a counterexample of $M, s \models AU(\varphi_1, \varphi_2)$. Thus for each lsr-path $\rho(s)$, $\exists 0 \leq i < \mathsf{len}(\rho) - 1$ such that $M, \rho_i \models \varphi_2$ holds and $\forall 0 \leq j < i$, $M, \rho_j \models \varphi_1$ holds. Then by induction hypothesis, for each lsr-path $\rho(s)$, $\exists 0 \leq i < \mathsf{len}(\rho) - 1$ such that $M, \rho_i \models_a \varphi_2$ holds and $\forall 0 \leq j < i$, $M, \rho_j \models_a \varphi_1$ holds. Thus $M, s \models_a AU(\varphi_1, \varphi_2)$ holds.

- Let $\varphi = EU(\varphi_1, \varphi_2)$. By the semantics of CTL, there exists an infinite path $\pi(s)$ and $\exists i \geq 0$ such that $M, \pi_i \models \varphi_2$ and $\forall 0 \leq j < i$, $M, \pi_j \models \varphi_1$. From the path π_0^i, by Lemma 4.3, there exists a path $\pi_0'^m$ without repeating states such that $\pi'_0 = \pi_0$, $\pi'_m = \pi_i$, and $\forall 0 < n < m$, π'_n is on π_0^i. Then by induction hypothesis, $M, \pi'_m \models_a \varphi_2$ and $\forall 0 \leq n < m$, $M, \pi'_n \models_a \varphi_1$. Thus $M, s \models_a EU(\varphi_1, \varphi_2)$ holds.

- Let $\varphi = AR(\varphi_1, \varphi_2)$. We prove the contrapositive. If there exists a lsr-path $\rho(s)$ and $\exists 0 \leq i < \mathsf{len}(\rho) - 1$ such that $M, \rho_i \not\models \varphi_2$ and $\forall 0 \leq j < i$, $M, \rho_j \not\models \varphi_1$. Then ρ_0^i is a counterexample of $M, s \models AR(\varphi_1, \varphi_2)$. Thus for each lsr-path $\rho(s)$ and $\forall 0 \leq i < \mathsf{len} - 1$, either $M, \rho_i \models \varphi_2$ or $\exists 0 \leq j < i$ such that $M, \rho_j \models \varphi_1$. By induction hypothesis, for each $\rho(s)$ and $\forall 0 \leq i < \mathsf{len} - 1$, either $M, \rho_i \models_a \varphi_2$ or $\exists 0 \leq j < i$ such that $M, \rho_j \models_a \varphi_1$. Thus $M, s \models_a AR(\varphi_1, \varphi_2)$ holds.

- Let $\varphi = ER(\varphi_1, \varphi_2)$. By the semantics of CTL, there exists an infinite path $\pi(s)$ such that $\forall j \geq 0$, either $M, \pi_j \models \varphi_2$ holds or $\exists 0 \leq i < j$ such that $M, \pi_i \models \varphi_1$ holds. By Lemma 4.2, $\exists k \leq 0$ such that π_0^k is a lsr-path and by induction hypothesis, $\forall 0 \leq m < k$, either $M, \pi_m \models_a \varphi_2$ holds or $\exists 0 \leq n < m$ such that $M, \pi_n \models_a \varphi_1$ holds. Thus $M, s \models_a ER(\varphi_1, \varphi_2)$ holds.

□

Lemma 4.6. *Let φ be a CTL formula of NNF. If $M, s \models_a \varphi$, then $M, s \models \varphi$.*

Proof. By induction on the structure of the formula φ. The cases $\varphi = p$, $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $AX\varphi_1$, $EX\varphi_1$ are trivial. For the other cases, the proof is as follows.

- Let $\varphi = AF\varphi_1$. If there is an infinite path $\pi(s)$ such that $\forall j \geq 0$, $M, \pi_j \not\models_a \varphi_1$, then by Lemma 4.2, there exists $k \geq 0$ such that π_0^k is a lsr-path, which is a counterexample of $M, s \models_a AF\varphi_1$. Thus for each infinite path $\pi(s)$, $\exists j \geq 0$ such that $M, \pi_j \models_a \varphi_1$ holds. Then by induction hypothesis, for each infinite path $\pi(s)$, $\exists j \geq 0$ such that $M, \pi_j \models \varphi_1$ holds and thus $M, s \models AF\varphi_1$ holds.

- Let $\varphi = EF\varphi_1$. By the alternative semantics of CTL, there exists a lsr-path $\rho(s)$ and $\exists 0 \leq i < \mathsf{len}(\rho) - 1$ such that $M, s_i \models_a \varphi_1$ holds and by induction hypothesis, $M, s_i \models \varphi_1$ holds. As there exists a path from s to s_i, we get $M, s \models EF\varphi_1$ holds.

- Let $\varphi = AG\varphi_1$. Assume that there exists an infinite path $\pi(s)$ and $\exists i \geq 0$, $M, \pi_i \not\models_a \varphi_1$. By Lemma 4.3, there exists a lsr-path $\rho(s)$ such that π_i is on ρ, which is a counterexample of $M, s \models_a AG\varphi_1$. Thus for each infinite path $\pi(s)$ and $\forall i \geq 0$, $M, \pi_i \models_a \varphi_1$ holds. Then by induction hypothesis, for each infinite path $\pi(s)$ and $\forall i \geq 0$, $M, \pi_i \models \varphi_1$ holds and thus $M, s \models AG\varphi_1$ holds.

- Let $\varphi = EG\varphi_1$. By the alternative semantics of CTL, there exists a lsr-path $\rho(s)(i,k)$ such that $\forall 0 \leq j < k$, $M, \rho_j \models_a \varphi_1$ and by induction hypothesis, $M, \rho_j \models \varphi_1$. As $\rho_0^i \hat{} (\rho_{i+1}^k)^\omega$ is an infinite path, thus $M, s \models EG\varphi_1$ holds.

- Let $\varphi = AU(\varphi_1, \varphi_2)$. Assume that there exists an infinite path $\pi(s)$ and $\forall j \geq 0$, either $M, \pi_j \not\models_a \varphi_2$ or $\exists 0 \leq i < j$ such that $M, \pi_i \not\models_a \varphi_1$. Then by Lemma 4.2, $\exists k \geq 0$ such that π_0^k is a lsr-path, which is a counterexample of $M, s \models_a AU(\varphi_1, \varphi_2)$. Thus for each infinite path $\pi(s)$, $\exists i \geq 0$ such that $M, \pi_i \models_a \varphi_2$ and $\forall 0 \leq m < i$, $M, \pi_m \models_a \varphi_1$. Then by induction hypothesis, for each infinite path $\pi(s)$, $\exists i \geq 0$ such that $M, \pi_i \models \varphi_2$ and $\forall 0 \leq m < i$, $M, \pi_m \models \varphi_1$. Thus $M, s \models AU(\varphi_1, \varphi_2)$ holds.

- Let $\varphi = EU(\varphi_1, \varphi_2)$. By the alternative semantics of CTL, there exists a lsr-path $\rho(s)$ and $\exists 0 \leq i < \mathsf{len}(\rho) - 1$ such that $M, \rho_i \models_a \varphi_2$ and $\forall 0 \leq j < i$, $M, \rho_j \models_a \varphi_1$. Then by induction hypothesis, $M, \rho_i \models \varphi_2$ holds and $\forall 0 \leq j < i$, $M, \rho_j \models \varphi_1$ holds. Thus $M, s \models EU(\varphi_1, \varphi_2)$ holds.

- Let $\varphi = AR(\varphi_1, \varphi_2)$. Assume that there exists a path $\pi(s)$ and $\exists j \geq 0$ such that $M, \pi_j \not\models_a \varphi_2$ and $\forall 0 \leq i < j$, $M, \pi_i \not\models_a \varphi_1$. By Lemma 4.3, there exists a finite path $\pi_0'^m$ without repeating states such that $\pi'_0 = \pi_0$, $\pi'_m = \pi_j$, and $\forall 0 < n < m$, π'_n is on π_0^j. By the alternative semantics of CTL, $\pi_0'^m$ is a counterexample of $M, s \models_a AR(\varphi_1, \varphi_2)$. Thus for each infinite path $\pi(s)$, by induction hypothesis, $\forall j \geq 0$, either $M, \pi_j \models \varphi_2$ or $\exists 0 \leq i < j$ such that $M, \pi_i \models \varphi_1$. By the semantics of CTL, $M, s \models AR(\varphi_1, \varphi_2)$ holds.

- Let $\varphi = ER(\varphi_1, \varphi_2)$. By the alternative semantics of CTL, there exists a lsr-path $\rho(s)(j,k)$ such that $\forall 0 \leq i < k$, either $M, \rho_i \models_a \varphi_2$ or $\exists 0 \leq m < i$ such that $M, \rho_m \models_a \varphi_1$. Then by induction hypothesis, either $M, \rho_i \models \varphi_2$ or $\exists 0 \leq m < i$ such that $M, \rho_m \models \varphi_1$. By Lemma 4.2, $\rho_0^j \hat{} (\rho_{j+1}^k)^\omega$ is an infinite path, thus by the semantics of CTL, $M, s \models ER(\varphi_1, \varphi_2)$ holds.

□

Theorem 4.7 (Soundness and Completeness)**.** *Let φ be a CTL formula. $M, s \models \varphi$ iff $M, s \models_a \varphi$.*

The soundness and completeness of the alternative semantics follows from Lemma 4.5 and Lemma 4.6.

4.2 Rewrite Rules of CTL on Finite Models

The work in this section is to express CTL formulas in Deduction Modulo and prove that for a CTL formula φ, the translation of $M, s \models_a \varphi$ is provable if and only if $M, s \models_a \varphi$ holds.

4.2.1 One-sided Sequent Calculus Modulo

In this chapter, to simplify the proofs, all the CTL formulas are in negation normal form and instead of using usual sequents of the form $A_1, ..., A_n \vdash B_1, ..., B_p$, we use one-sided sequents [TS96], where all the propositions are put on the right hand side of the sequent sign $\vdash$ and the sequent above is transformed into $\vdash \neg A_1, ..., \neg A_n, B_1, ..., B_p$. Moreover, implication is defined from disjunction and negation ($A \Rightarrow B$ is just an abbreviation for $\neg A \vee B$), and negation is pushed inside the propositions using **De Morgan's laws**. For each atomic proposition P we also have a dual atomic proposition $P^\perp$ corresponding to its negation, and the operator $\perp$ extends to all the propositions. So that the axiom rule can be formulated as

$$\frac{}{\vdash P, P^\perp}\ \text{axiom}$$

The *One-sided Sequent Calculus Modulo*, which takes the rewrite rules into account, is presented in Figure 4.4.

$$\frac{}{\vdash_{\mathcal{R}} A, B}\ \text{axiom}\ A \overset{*}{\hookrightarrow} P, B \overset{*}{\hookrightarrow} P^\perp \qquad \frac{\vdash_{\mathcal{R}} A, \Delta \qquad \vdash_{\mathcal{R}} B, \Delta}{\vdash_{\mathcal{R}} \Delta}\ \text{cut}\ A \overset{*}{\hookrightarrow} C, B \overset{*}{\hookrightarrow} C^\perp$$

$$\frac{\vdash_{\mathcal{R}} \Delta}{\vdash_{\mathcal{R}} A, \Delta}\ \text{weak} \qquad \frac{\vdash_{\mathcal{R}} B, C, \Delta}{\vdash_{\mathcal{R}} A, \Delta}\ \text{contr}\ A \overset{*}{\hookrightarrow} B, A \overset{*}{\hookrightarrow} C$$

$$\frac{}{\vdash_{\mathcal{R}} A, \Delta}\ \top\ A \overset{*}{\hookrightarrow} \top \qquad \frac{\vdash_{\mathcal{R}} B, \Delta \qquad \vdash_{\mathcal{R}} C, \Delta}{\vdash_{\mathcal{R}} A, \Delta}\ \wedge\ A \overset{*}{\hookrightarrow} B \wedge C$$

$$\frac{\vdash_{\mathcal{R}} B, \Delta}{\vdash_{\mathcal{R}} A, \Delta}\ \vee_1\ A \overset{*}{\hookrightarrow} B \vee C \qquad \frac{\vdash_{\mathcal{R}} C, \Delta}{\vdash_{\mathcal{R}} A, \Delta}\ \vee_2\ A \overset{*}{\hookrightarrow} B \vee C$$

$$\frac{\vdash_{\mathcal{R}} C, \Delta}{\vdash_{\mathcal{R}} A, \Delta}\ \exists\ A \overset{*}{\hookrightarrow} \exists x B, (t/x)B \overset{*}{\hookrightarrow} C \qquad \frac{\vdash_{\mathcal{R}} B, \Delta}{\vdash_{\mathcal{R}} A, \Delta}\ \forall\ A \overset{*}{\hookrightarrow} \forall x B, x \notin FV(\Delta)$$

FIGURE 4.4: One-sided Sequent Calculus Modulo

Note that as our system is negation free, all occurrences of atomic propositions are positive. Thus, the rule $P \hookrightarrow A$ does not correspond to an equivalence $P \Leftrightarrow A$ but to an implication $A \Rightarrow P$. In other words, our one-sided presentation of deduction modulo is

closer to polarized deduction modulo (Figure 2.1) with positive rules only, than to the usual deduction modulo. The sequent $\vdash_{\mathcal{R}} \Delta$ has a cut-free proof is represented as $\vdash_{\mathcal{R}}^{cf} \Delta$ has a proof.

4.2.2 First-order Representation

In this subsection, we represent the CTL model checking problems with a two-sorted first-order language. In this language, the CTL operators are treated as function symbols.

Language As in [DJ13b], we consider a two-sorted language $\mathcal{L}$, which contains

- constants $s_1, \ldots, s_n$ for each state of M.
- predicate symbols $\varepsilon_0, \varepsilon_{\sqcap_0}, \varepsilon_{\sqcup_0}, \varepsilon_1, \varepsilon_{\sqcap_1}, \varepsilon_{\sqcup_1}$, in which the binary predicates ε_0, $\varepsilon_{\sqcap_0}$ and $\varepsilon_{\sqcup_0}$ apply to all the CTL formulas, while the ternary predicates ε_1, $\varepsilon_{\sqcap_1}$ and $\varepsilon_{\sqcup_1}$ only apply to the CTL formulas starting with the temporal connectives AG, EG, AR and ER.
- binary predicate symbols *mem* for the membership, r for the next-notation.
- a constant nil and a binary function symbol con.

We use x, y, z to denote the variables of the state terms, X, Y, Z to denote the class variables. A class is in fact a set of states, here we prefer to use "*class theory*", rather than "*(monadic) second order logic*", is to emphasis that this formalism is a theory and not a logic.

CTL Term To express CTL in Deduction Modulo, firstly, we translate the CTL formula φ into a term $|\varphi|$ (called CTL term). The term form of a CTL formula is defined as follows:

$\lvert p\rvert = \overline{p},\ p \in AP$	$\lvert p^{\perp}\rvert = \mathsf{not}(\overline{p}),\ p \in AP$
$\lvert\varphi \wedge \psi\rvert = \mathsf{and}(\lvert\varphi\rvert, \lvert\psi\rvert)$	$\lvert\varphi \vee \psi\rvert = \mathsf{or}(\lvert\varphi\rvert, \lvert\psi\rvert)$
$\lvert AX\varphi\rvert = \mathsf{ax}(\lvert\varphi\rvert)$	$\lvert EX\varphi\rvert = \mathsf{ex}(\lvert\varphi\rvert)$
$\lvert AF\varphi\rvert = \mathsf{af}(\lvert\varphi\rvert)$	$\lvert EF\varphi\rvert = \mathsf{ef}(\lvert\varphi\rvert)$
$\lvert AG\varphi\rvert = \mathsf{ag}(\lvert\varphi\rvert)$	$\lvert EG\varphi\rvert = \mathsf{eg}(\lvert\varphi\rvert)$
$\lvert AU(\varphi, \psi)\rvert = \mathsf{au}(\lvert\varphi\rvert, \lvert\psi\rvert)$	$\lvert EU(\varphi, \psi)\rvert = \mathsf{eu}(\lvert\varphi\rvert, \lvert\psi\rvert)$
$\lvert AR(\varphi, \psi)\rvert = \mathsf{ar}(\lvert\varphi\rvert, \lvert\psi\rvert)$	$\lvert ER(\varphi, \psi)\rvert = \mathsf{er}(\lvert\varphi\rvert, \lvert\psi\rvert)$

Note that we use Φ, Ψ to denote the variables of the CTL terms. Both sets and paths are represented with the symbols con and nil. For the set $S' = \{s_i, \ldots, s_j\}$, we use $[S']$ to denote its term form $\mathsf{con}(s_i, \mathsf{con}(\ldots, \mathsf{con}(s_j, \mathsf{nil})\ldots))$. For the path $s_i^j = s_i, \ldots, s_j$, we use $[s_i^j]$ to denote the term $\mathsf{con}(s_j, \mathsf{con}(\ldots, \mathsf{con}(s_i, \mathsf{nil})\ldots))$. And then the formula φ holds on s is expressed as $\varepsilon_0(|\varphi|, s)$.

Definition 4.8 (Semantics of $\mathcal{L}$)**.** Semantics of the formulas in the language $\mathcal{L}$ is as follows:

$M \models \varepsilon_0(|\varphi|, s) \Leftrightarrow M, s \models_a \varphi$.

$M \models r(s, [S']) \Leftrightarrow S' = \mathsf{next}(s)$.

$M \models mem(s, [s_0^i]) \Leftrightarrow s$ is on the path s_0^i.

$M \models \varepsilon_{\sqcap_0}(|\varphi|, [S']) \Leftrightarrow \forall s \in S',\ M \models \varepsilon_0(|\varphi|, s)$.

$M \models \varepsilon_{\sqcup_0}(|\varphi|, [S']) \Leftrightarrow \exists s \in S'$ such that $M \models \varepsilon_0(|\varphi|, s)$.

$M \models \varepsilon_1(\mathsf{ag}(|\varphi_1|), s, [s_0^i]) \Leftrightarrow$ for each lsr-path $s_0^i \hat{} s_{i+1}^k (s_{i+1} = s)$, and $\forall i < j < k$, $M \models \varepsilon_0(|\varphi_1|, s_j)$.

$M \models \varepsilon_1(\mathsf{eg}(|\varphi_1|), s, [s_0^i]) \Leftrightarrow$ there exists a lsr-path $s_0^i \hat{} s_{i+1}^k (s_{i+1} = s)$, and $\forall i < j < k$, $M \models \varepsilon_0(|\varphi_1|, s_j)$.

$M \models \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, [s_0^i]) \Leftrightarrow$ for each lsr-path $s_0^i \hat{} s_{i+1}^k (s_{i+1} = s)$, and $\forall i < j < k$, either $M \models \varepsilon_0(|\varphi_2|, s_j)$ or $\exists i < m < j$ such that $M \models \varepsilon_0(|\varphi_1|, s_m)$.

$M \models \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, [s_0^i]) \Leftrightarrow$ there exists a lsr-path $s_0^i \hat{} s_{i+1}^k (s_{i+1} = s)$, and $\forall i < j < k$, either $M \models \varepsilon_0(|\varphi_2|, s_j)$ or $\exists i < m < j$ such that $M \models \varepsilon_0(|\varphi_1|, s_m)$.

$M \models \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), [S'], [s_0^i]) \Leftrightarrow \forall s \in S',\ M \models \varepsilon_1(\mathsf{ag}(|\varphi_1|), s, [s_0^i])$.

$M \models \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), [S'], [s_0^i]) \Leftrightarrow \forall s \in S',\ M \models \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, [s_0^i])$.

$M \models \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi_1|), [S'], [s_0^i]) \Leftrightarrow \exists s \in S'$ such that $M \models \varepsilon_1(\mathsf{eg}(|\varphi_1|), s, [s_0^i])$.

$M \models \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), [S'], [s_0^i]) \Leftrightarrow \exists s \in S'$ such that $M \models \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, [s_0^i])$.

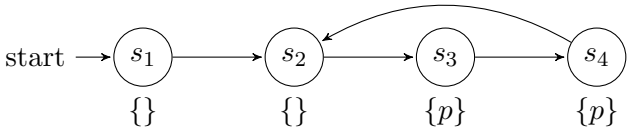


FIGURE 4.5: Example for the Semantics of $\mathcal{L}$

Example 4.2. *In Figure 4.5, we have* $M \models \varepsilon_1(\mathsf{eg}(\overline{p}), s_3, \mathsf{con}(s_2, \mathsf{con}(s_1, \mathsf{nil})))$ *because there exists a lsr-path, for instance* s_1, s_2, s_3, s_4, s_2 *such that* p *holds on* s_3 *and* s_4.

Note that when a formula $\varepsilon_1(|\varphi|, s, [s_i^j])$ is valid in M, for instance $M \models \varepsilon_1(\mathsf{eg}(|\varphi|), s, [s_i^j])$, $EG\varphi$ may not hold on the state s.

4.2.3 Rewrite System

The rewrite system $\mathcal{R}$ is composed by three components,

1. rules for the Kripke structure M (denoted as $\mathcal{R}_M$),
2. rules for the class variables (denoted as $\mathcal{R}_c$),
3. rules for the semantics encoding of the CTL operators (denoted as $\mathcal{R}_{CTL}$).

The rules of $\mathcal{R}_M$ The rules of $\mathcal{R}_M$ are as follows:

- for each atomic formula $p \in AP$ and each state $s \in S$, if $p \in \mathsf{L}(s)$, then $\varepsilon_0(\overline{p}, s) \hookrightarrow \top$ is in $\mathcal{R}_M$, otherwise take $\varepsilon_0(\mathsf{not}(\overline{p}), s) \hookrightarrow \top$ as a rewrite rule of $\mathcal{R}_M$.
- for each state $s \in S$, take $r(s, [\mathsf{next}(s)]) \hookrightarrow \top$ as a rewrite rule of $\mathcal{R}_M$.

The rules of $\mathcal{R}_c$ For the class variables, as the domain of the model is finite, the property of membership can be expressed by the following two axioms [DJ13b],

$$\forall x(x = x),$$

$$\forall x \forall y \forall Z((x = y \vee mem(x, Z)) \Leftrightarrow mem(x, \mathsf{con}(y, Z))).$$

The rewrite rules for these axioms are

$$x = x \hookrightarrow \top,$$

$$mem(x, \mathsf{con}(y, Z)) \hookrightarrow x = y \vee mem(x, Z).$$

To avoid introducing "=" , these two rules are replaced by $\mathcal{R}_c$:

$$mem(x, \mathsf{con}(x, Z)) \hookrightarrow \top,$$

$$mem(x, \mathsf{con}(y, Z)) \hookrightarrow mem(x, Z),$$

The rules of $\mathcal{R}_{CTL}$ The rewrite rules for the predicates carrying the semantic definition of the CTL formulas, are shown in Figure 4.6.

$$\varepsilon_0(\mathsf{or}(\Phi,\Psi),x) \hookrightarrow \varepsilon_0(\Phi,x) \vee \varepsilon_0(\Psi,x)$$
$$\varepsilon_0(\mathsf{and}(\Phi,\Psi),x) \hookrightarrow \varepsilon_0(\Phi,x) \wedge \varepsilon_0(\Psi,x)$$
$$\varepsilon_0(\mathsf{ax}(\Phi),x) \hookrightarrow \exists X(r(x,X) \wedge \varepsilon_{\sqcap_0}(\Phi,X))$$
$$\varepsilon_0(\mathsf{ex}(\Phi),x) \hookrightarrow \exists X(r(x,X) \wedge \varepsilon_{\sqcup_0}(\Phi,X))$$
$$\varepsilon_0(\mathsf{af}(\Phi),x) \hookrightarrow \varepsilon_0(\Phi,x) \vee \exists X(r(x,X) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(\Phi),X))$$
$$\varepsilon_0(\mathsf{ef}(\Phi),x) \hookrightarrow \varepsilon_0(\Phi,x) \vee \exists X(r(x,X) \wedge \varepsilon_{\sqcup_0}(\mathsf{ef}(\Phi),X))$$
$$\varepsilon_0(\mathsf{ag}(\Phi),x) \hookrightarrow \varepsilon_1(\mathsf{ag}(\Phi),x,\mathsf{nil})$$
$$\varepsilon_0(\mathsf{eg}(\Phi),x) \hookrightarrow \varepsilon_1(\mathsf{eg}(\Phi),x,\mathsf{nil})$$
$$\varepsilon_0(\mathsf{au}(\Phi,\Psi),x) \hookrightarrow \varepsilon_0(\Psi,x) \vee (\varepsilon_0(\Phi,x) \wedge \exists X(r(x,X) \wedge \varepsilon_{\sqcap_0}(\mathsf{au}(\Phi,\Psi),X)))$$
$$\varepsilon_0(\mathsf{eu}(\Phi,\Psi),x) \hookrightarrow \varepsilon_0(\Psi,x) \vee (\varepsilon_0(\Phi,x) \wedge \exists X(r(x,X) \wedge \varepsilon_{\sqcup_0}(\mathsf{eu}(\Phi,\Psi),X)))$$
$$\varepsilon_0(\mathsf{ar}(\Phi,\Psi),x) \hookrightarrow \varepsilon_1(\mathsf{ar}(\Phi,\Psi),x,\mathsf{nil})$$
$$\varepsilon_0(\mathsf{er}(\Phi,\Psi),x) \hookrightarrow \varepsilon_1(\mathsf{er}(\Phi,\Psi),x,\mathsf{nil})$$
$$\varepsilon_{\sqcap_0}(\Phi,\mathsf{con}(x,X)) \hookrightarrow \varepsilon_0(\Phi,x) \wedge \varepsilon_{\sqcap_0}(\Phi,X)$$
$$\varepsilon_{\sqcap_0}(\Phi,\mathsf{nil}) \hookrightarrow \top$$
$$\varepsilon_{\sqcup_0}(\Phi,\mathsf{con}(x,X)) \hookrightarrow \varepsilon_0(\Phi,x) \vee \varepsilon_{\sqcup_0}(\Phi,X)$$
$$\varepsilon_1(\mathsf{ag}(\Phi),x,Y) \hookrightarrow mem(x,Y) \vee(\varepsilon_0(\Phi,x) \wedge \exists X(r(x,X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(\Phi),X,\mathsf{con}(x,Y))))$$
$$\varepsilon_1(\mathsf{eg}(\Phi),x,Y) \hookrightarrow mem(x,Y) \vee(\varepsilon_0(\Phi,x) \wedge \exists X(r(x,X) \wedge \varepsilon_{\sqcup_1}(\mathsf{eg}(\Phi),X,\mathsf{con}(x,Y))))$$
$$\varepsilon_1(\mathsf{ar}(\Phi,\Psi),x,Y) \hookrightarrow mem(x,Y) \vee(\varepsilon_0(\Psi,x) \wedge (\varepsilon_0(\Phi,x) \vee \exists X(r(x,X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(\Phi,\Psi),X,\mathsf{con}(x,Y)))))$$
$$\varepsilon_1(\mathsf{er}(\Phi,\Psi),x,Y) \hookrightarrow mem(x,Y) \vee(\varepsilon_0(\Psi,x) \wedge (\varepsilon_0(\Phi,x) \vee \exists X(r(x,X) \wedge \varepsilon_{\sqcup_1}(\mathsf{er}(\Phi,\Psi),X,\mathsf{con}(x,Y)))))$$
$$\varepsilon_{\sqcap_1}(\Phi,\mathsf{con}(x,X),Y) \hookrightarrow \varepsilon_1(\Phi,x,Y) \wedge \varepsilon_{\sqcap_1}(\Phi,X,Y)$$
$$\varepsilon_{\sqcap_1}(\Phi,\mathsf{nil},Y) \hookrightarrow \top$$
$$\varepsilon_{\sqcup_1}(\Phi,\mathsf{con}(x,X),Y) \hookrightarrow \varepsilon_1(\Phi,x,Y) \vee \varepsilon_{\sqcup_1}(\Phi,X,Y)$$

FIGURE 4.6: Rewrite Rules of CTL Connectives($\mathcal{R}_{CTL}$)

Example 4.3. *The rewrite rule*

$$\varepsilon_1(\mathsf{eg}(|\varphi|), s, [s_i^j]) \hookrightarrow mem(s, [s_i^j]) \vee (\varepsilon_0(|\varphi|, s) \wedge \exists X(r(s, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi|), X, \mathsf{con}(s, [s_i^j]))))$$

expresses that $M \models \varepsilon_1(\mathsf{eg}(|\varphi|), s, [s_i^j])$ *holds, if and only if*

$$s_i^j \hat{} s \text{ is a lsr-path (that is } s \text{ occurs in } s_i^j), \text{ OR}$$

$$M \models \varepsilon_0(|\varphi|, s) \text{ and } M \models \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi|), [\mathsf{next}(s)], \mathsf{con}(s, [s_i^j])) \text{ holds.}$$

Remark *Why do we encode the relation "r" as "a state to all its successors", rather than "a state to one successor"?* If the relation is "state-to-state", then the encoding of the temporal formula $AX\Phi$ would be

$$\varepsilon_0(\mathsf{ax}(\Phi), x) \hookrightarrow \forall y(r(x, y) \Rightarrow \varepsilon_0(\Phi, y)),$$

in which a free variable y would be introduced. However, in the sequent $\vdash_{\mathcal{R}} r(s, y)^{\perp}, \varepsilon_0(\overline{p}, y)$, neither $r(s, y)^{\perp}$, nor $\varepsilon_0(\overline{p}, y)$ can be reduced any more. As this sequent cannot be proved by the axiom rule, thus there exists no proof for this sequent. To avoid introducing free variables, the relation is represented as "state-to-all successors" in this dissertation. Then the temporal formula $AX\Phi$ is encoded as

$$\varepsilon_0(\mathsf{ax}(\Phi), x) \hookrightarrow \exists Y(r(x, Y) \wedge \varepsilon_{\sqcap_0}(\Phi, Y)).$$

In this way, the sequent $\vdash_{\mathcal{R}} \exists Y(r(s, Y) \wedge \varepsilon_{\sqcap_0}(\overline{p}, Y))$ can be proved by replacing Y with $[\mathsf{next}(s)]$.

4.2.4 Soundness and Completeness

Now we prove the soundness and completeness of the deduction system modulo the set of rewrite rules $\mathcal{R}$, to make sure that our strategy of solving model checking problems with Deduction Modulo preserves the termination and correctness.

Lemma 4.9 (Soundness)**.** *For any CTL formula* φ *of NNF, if the sequent* $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi|, s)$ *has a proof, then* $M \models \varepsilon_0(|\varphi|, s)$*.*

Proof. More generally, we prove that for any CTL proposition φ of NNF,

- if $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi|, s)$ has a proof, then $M \models \varepsilon_0(|\varphi|, s)$.

- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_0}(|\varphi|, [S'])$ has a proof, then $M \models \varepsilon_{\sqcap_0}(|\varphi|, [S'])$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_0}(|\varphi|, [S'])$ has a proof, then $M \models \varepsilon_{\sqcup_0}(|\varphi|, [S'])$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(|\varphi|, s, [s_i^j])$ has a proof, where φ is either of the form $AG\varphi_1$, $EG\varphi_1$, $AR(\varphi_1, \varphi_2)$, $ER(\varphi_1, \varphi_2)$, then $M \models \varepsilon_1(|\varphi|, s, [s_i^j])$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_1}(|\varphi|, [S'], [s_i^j])$ has a proof, where φ is either of the form $AG\varphi_1$, $AR(\varphi_1, \varphi_2)$, then $M \models \varepsilon_{\sqcap_1}(|\varphi|, [S'], [s_i^j])$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_1}(|\varphi|, [S'], [s_i^j])$ has a proof, where φ is either of the form $EG\varphi_1$, $ER(\varphi_1, \varphi_2)$, then $M \models \varepsilon_{\sqcup_1}(|\varphi|, [S'], [s_i^j])$.

By induction on the size of the proof. Consider the different case for φ, we have 18 cases (2 cases for the atomic proposition and the negation of the atomic proposition, 2 cases for and and or, 10 cases for the temporal connectives ax, ex, af, ef, ag, eg, au, eu, ar, er, 4 cases for the predicate symbols $\varepsilon_{\sqcap_0}$, $\varepsilon_{\sqcup_0}$, $\varepsilon_{\sqcap_1}$, $\varepsilon_{\sqcup_0}$), but each case is easy. For brevity, we just prove two cases. The full proof is in **Appendix A**.

- Suppose the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{af}(|\varphi|), s)$ has a proof. As $\varepsilon_0(\mathsf{af}(|\varphi|), s) \hookrightarrow \varepsilon_0(|\varphi|, s) \vee \exists X(r(s, X) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi|), X))$, the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models \varepsilon_0(|\varphi|, s)$ holds by IH, then $M \models \varepsilon_0(\mathsf{af}(|\varphi|), s)$ holds by its semantic definition. For $\vee_2$, $M \models \exists X(r(s, X) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi|), X))$ holds by IH, thus there exists S' such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi|), [S'])$ holds. Then we get $S' = \mathsf{next}(s)$ and for each state s' in S', $M \models \varepsilon_0(\mathsf{af}(|\varphi|), s')$ holds. Now assume $M \not\models \varepsilon_0(\mathsf{af}(|\varphi|), s)$, then there exists a lsr-path $\rho(s)(j, k)$ such that $\forall 0 \leq i < k$, $M \not\models \varepsilon_0(|\varphi|, \rho_i)$. For the path $\rho(s)(j, k)$,
  - if $j \neq 0$, then the path ρ_1^k is a lsr-path, which is a counterexample of $M \models \varepsilon_0(\mathsf{af}(|\varphi|), \rho_1)$.
  - if $j = 0$, then the path $\rho_1^k \hat{} \rho_1$ is a lsr-path, which is a counterexample of $M \models \varepsilon_0(\mathsf{af}(|\varphi|), \rho_1)$.

  Thus $M \models \varepsilon_0(\mathsf{af}(|\varphi|), s)$ holds by its semantic definition.
- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ag}(|\varphi|), s, [s_i^j])$ has a proof. As $\varepsilon_1(\mathsf{ag}(|\varphi|), s, [s_i^j]) \hookrightarrow mem(s, [s_i^j]) \vee (\varepsilon_0(|\varphi|, s) \wedge \exists X(r(s, X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi|), X, \mathsf{con}(s, [s_i^j]))))$, the last rule in the proof is $\vee_1$ or $\vee_2$. If the last rule is $\vee_1$, then $M \models mem(s, [s_i^j])$ holds by IH. Thus $s_i^j \hat{} s$ is a lsr-path and $M \models \varepsilon_1(\mathsf{ag}(|\varphi|), s, [s_i^j])$ holds by its semantic definition. If the rule is $\vee_2$, then $M \models \varepsilon_0(|\varphi|, s)$ and $M \models \exists X(r(s, X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi|), X, \mathsf{con}(s, [s]_i^j)))$ holds by IH. Thus there exists S' such that $M \models r(s, [S']) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi|), [S'], \mathsf{con}(s, [s_i^j]))$

holds. Then by the semantic definition, $S' = \mathsf{next}(s)$ and for each state $s' \in S'$, $M \models \varepsilon_1(\mathsf{ag}(|\varphi|), s', \mathsf{con}(s, [s_i^j]))$ holds. Thus $M \models \varepsilon_1(\mathsf{ag}(|\varphi|), s, [s_i^j])$ holds by its semantic definition.

□

Lemma 4.10 (Completeness)**.** *For a CTL formula φ of NNF, if $M \models \varepsilon_0(|\varphi|, s)$, then the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi|, s)$ has a proof.*

Proof. By induction on the structure of φ. For brevity, here we just prove some of the cases. The full proof is in **Appendix A**.

- Suppose $M \models \varepsilon_0(\mathsf{af}(|\varphi_1|), s)$ holds. By the semantics of $\mathcal{L}$, there exists a state s' on each lsr-path starting from s such that $M \models \varepsilon_0(|\varphi_1|, s')$ holds. Thus there exists a finite tree T such that

  - T has root s;
  - for each internal node s' in T, the children of s' are labelled by the elements of $\mathsf{next}(s')$;
  - for each leaf s', s' is the first node in the branch starting from s such that $M \models \varepsilon_0(|\varphi_1|, s')$ holds.

  By IH, for each leaf s', there exists a proof $\Pi_{(\varphi_1, s')}$ for the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi_1|, s')$. Then, to each subtree T' of T, we associate a proof $|T'|$ of the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{af}(|\varphi_1|), s')$ where s' is the root of T', by induction, as follows,

  - if T' contains a single node s', then the proof $|T'|$ is as follows:

$$\dfrac{\Pi_{(\varphi_1, s')}}{\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{af}(|\varphi_1|), s')} \vee_1$$

  - if $T' = s'(T_1, \ldots, T_n)$, then the proof $|T'|$ is as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash_{\mathcal{R}}^{cf} r(s', [\mathsf{next}(s')])} \top \quad \dfrac{|T_1| \quad \ldots \quad |T_n|}{\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi_1|), [\mathsf{next}(s')])} \wedge^n}{\vdash_{\mathcal{R}}^{cf} r(s', [\mathsf{next}(s')]) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi_1|), [\mathsf{next}(s')])} \wedge}{\vdash_{\mathcal{R}}^{cf} \exists X(r(s', X) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi_1|), X))} \exists}{\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{af}(|\varphi_1|), s')} \vee_2$$

  This way, $|T|$ is a proof of the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{af}(|\varphi_1|), s)$.

- Suppose $M \models \varepsilon_0(\mathsf{ag}(|\varphi_1|), s)$ holds. By the semantics of $\mathcal{L}$, for each state s' on each lsr-path starting from s, $M \models \varepsilon_0(|\varphi_1|, s')$ holds. Thus there exists a finite tree T such that

- T has root s;
- for each internal node s' in T, the children of s' are labelled by the elements of $\mathsf{next}(s')$;
- the branch starting from s to each leaf is a lsr-path;
- for each internal node s' in T, $M \models \varepsilon_0(|\varphi_1|, s')$ holds and by IH, there exists a proof $\Pi_{(\varphi_1,s')}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s')$.

Then, to each subtree T' of T, we associate a proof $|T'|$ of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ag}(|\varphi_1|), s', [s_0'^{k-1}])$ where s' is the root of T' and $s_0'^k (s'_k = s')$ is the branch from s to s', by induction, as follows,

- if T' contains a single node s', then $s_0'^k$ is a lsr-path and the proof is as follows:

$$\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} mem(s', [s_0'^{k-1}])}\top}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ag}(|\varphi_1|), s', [s_0'^{k-1}])}\vee_2$$

- if $T' = s'(T_1, \ldots, T_n)$, the proof is as follows:

$$\dfrac{\dfrac{\Pi_{s'}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s')} \quad \dfrac{\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')])}\top \quad \dfrac{|T_1| \quad \ldots \quad |T_n|}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), [\mathsf{next}(s')], [s_0'^k])}\wedge^n}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')]) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), [\mathsf{next}(s')], [s_0'^k])}\wedge}{\vdash^{cf}_{\mathcal{R}} \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), X, [s_0'^k]))}\exists}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s') \wedge \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), X, [s_0'^k]))}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ag}(|\varphi_1|), s', [s_0'^{k-1}])}\vee_1$$

This way, as $\varepsilon_0(\mathsf{ag}(|\varphi_1|), s)$ can be rewritten into $\varepsilon_1(\mathsf{ag}(|\varphi_1|), s, \mathsf{nil})$, $|T|$ is a proof for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ag}(|\varphi_1|), s)$.

□

Theorem 4.11 (Soundness and Completeness)**.** *For a CTL proposition φ of NNF, the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi|, s)$ has a proof iff $M \models \varepsilon_0(|\varphi|, s)$ holds.*

The soundness and completeness of the One-sided Sequent Calculus Modulo for language $\mathcal{L}$ Follows from Lemma 4.9 and Lemma 4.10.

Theorem 4.12 (Soundness and Completeness)**.** *For a CTL proposition φ of NNF, $M, s \models \varphi$ holds iff the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi|, s)$ has a proof.*

This theorem can be proved using Theorem 4.7, Definition 4.8 and Theorem 4.11. See Figure 4.7.

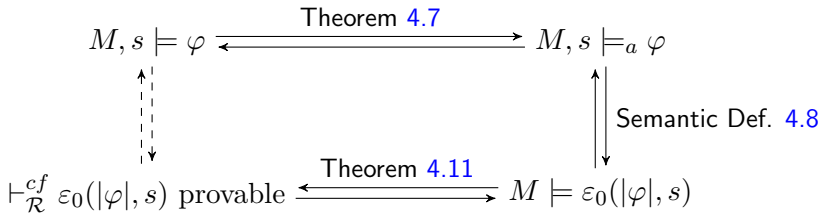


FIGURE 4.7: Proof of Theorem 4.12

4.3 Summary

In this chapter, an alternative semantics of Computation Tree Logic is defined, in which the temporal formulas are expressed with lsr-paths. In finite transition systems, the alternative semantics of a CTL formula is logically equivalent to the general semantics that are expressed with infinite paths. Based on the alternative semantics, a way of solving model checking problems with Deduction Modulo is presented. That is to express the transition system, the temporal operators with rewrite rules. The soundness and completeness of the deduction system modulo these rewrite rules showed that $M, s \models_a \phi$ holds if and only if the representation of $M, s \models_a \phi$ with the two-sorted language $\mathcal{L}$ in Section 4.2.2 is cut free provable.

The success to embed model checking problems into Deduction Modulo verifies the feasibility of solving the model checking problems on automated theorem provers.

5 Clausal Encoding of Temporal Properties

In Chapter 4, we have shown that CTL model checking problems can be solved by Deduction Modulo. From this theoritical basis, the proof-search algorithms designed for Deduction Modulo, such as Resolution Modulo [Dow10, DHK03] or Tableaux Modulo [DDG+13], can be used to build proofs in CTL. In this chapter, we present the procedure of encoding model checking problems as input of iProver Modulo, in which the Ordered Polarized Resolution Modulo proof method is embedded. The input file of the CTL model checking problem involves the following set of clauses:

- the one-way clauses for the encoding of the transition system, which includes the transition relations and atomic propositions;
- the one-way clauses for the connectives of CTL;
- the specification of the temporal properties to be checked.

Outline of This Chapter. In Section 5.1, the one-way clauses to present the rewrite system are illustrated. Section 5.2 presents an example of explicit model checking with Polarized Resolution Modulo. In Section 5.3, the symbolic representation of the transition system is discussed. To improve the efficiency of the proof-search algorithm, a literal selection function to restrict the application of Resolution is presented in Section 5.4. Finally, the experimental evaluation of the feasibility of the resolution method is presented.

5.1 Rewrite Rules to One-way Clauses

In Chapter 4, the rewrite system $\mathcal{R}$, which includes the rules of Kripke structure $\mathcal{R}_M$, the rules of class variables $\mathcal{R}_c$, the rules of semantics encoding of CTL connectives $\mathcal{R}_{CTL}$, was presented. The work in this section is to translate the rewrite rules into one-way clauses.

One-way Clauses of $\mathcal{R}_M$

For each atomic proposition $p \in AP$ and each state $s \in S$, if $\varepsilon_0(\overline{p}, s) \hookrightarrow \top$ is in $\mathcal{R}_M$, then take

$$\underline{\varepsilon_0(\overline{p}, s)}$$

as a one-way clause. If $\varepsilon_0(\mathsf{not}(\overline{p}), s) \hookrightarrow \top$ is in $\mathcal{R}_M$, then take

$$\underline{\neg\varepsilon_0(\overline{p}, s)}$$

as a one-way clause. For each rule $r(s, [\mathsf{next}(s)]) \hookrightarrow \top$ of $\mathcal{R}_M$, take

$$\underline{r(s, [\mathsf{next}(s)])}$$

as a one-way clause.

One-way Clauses of $\mathcal{R}_c$

The two rewrite rules for class variables, $mem(x, \mathsf{con}(x, Z)) \hookrightarrow \top$ and $mem(x, \mathsf{con}(y, Z)) \hookrightarrow mem(x, Z)$, are translated into one-way clauses

$$\underline{mem(x, \mathsf{con}(x, Z))}$$

$$\underline{mem(x, \mathsf{con}(y, Z))} \vee \neg mem(x, Z)$$

One-way Clauses of $\mathcal{R}_{CTL}$

The translation of $\mathcal{R}_{CTL}$, which is the set of rewrite rules for the encoding of CTL operators (Section 4.2.3), is presented in Figure 5.1.

$\underline{\varepsilon_0(\mathsf{or}(\Phi,\Psi),x)} \vee \neg\varepsilon_0(\Phi,x)$ $\quad$ $\underline{\varepsilon_0(\mathsf{or}(\Phi,\Psi),x)} \vee \neg\varepsilon_0(\Psi,x)$

$\underline{\varepsilon_0(\mathsf{and}(\Phi,\Psi),x)} \vee \neg\varepsilon_0(\Phi,x) \vee \neg\varepsilon_0(\Psi,x)$

$\underline{\varepsilon_0(\mathsf{ax}(\Phi),x)} \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcap_0}(\Phi,X)$ $\quad$ $\underline{\varepsilon_0(\mathsf{ex}(\Phi),x)} \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcup_0}(\Phi,X)$

$\underline{\varepsilon_0(\mathsf{af}(\Phi),x)} \vee \neg\varepsilon_0(\Phi,x)$ $\quad$ $\underline{\varepsilon_0(\mathsf{af}(\Phi),x)} \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcap_0}(\mathsf{af}(\Phi),X)$

$\underline{\varepsilon_0(\mathsf{ef}(\Phi),x)} \vee \neg\varepsilon_0(\Phi,x)$ $\quad$ $\underline{\varepsilon_0(\mathsf{ef}(\Phi),x)} \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcup_0}(\mathsf{ef}(\Phi),X)$

$\underline{\varepsilon_0(\mathsf{ag}(\Phi),x)} \vee \neg\varepsilon_1(\mathsf{ag}(\Phi),x,\mathsf{nil})$ $\quad$ $\underline{\varepsilon_1(\mathsf{ag}(\Phi),x,Y)} \vee \neg mem(x,Y)$

$\underline{\varepsilon_1(\mathsf{ag}(\Phi),x,Y)} \vee \neg\varepsilon_0(\Phi,x) \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcap_1}(\mathsf{ag}(\Phi),X,\mathsf{con}(x,Y))$

$\underline{\varepsilon_0(\mathsf{eg}(\Phi),x)} \vee \neg\varepsilon_1(\mathsf{eg}(\Phi),x,\mathsf{nil})$ $\quad$ $\underline{\varepsilon_1(\mathsf{eg}(\Phi),x,Y)} \vee \neg mem(x,Y)$

$\underline{\varepsilon_1(\mathsf{eg}(\Phi),x,Y)} \vee \neg\varepsilon_0(\Phi,x) \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\Phi),X,\mathsf{con}(x,Y))$

$\underline{\varepsilon_0(\mathsf{au}(\Phi,\Psi),x)} \vee \neg\varepsilon_0(\Psi,x)$

$\underline{\varepsilon_0(\mathsf{au}(\Phi,\Psi),x)} \vee \neg\varepsilon_0(\Phi,x) \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcap_0}(\mathsf{au}(\Phi,\Psi),X)$

$\underline{\varepsilon_0(\mathsf{eu}(\Phi,\Psi),x)} \vee \neg\varepsilon_0(\Psi,x)$

$\underline{\varepsilon_0(\mathsf{eu}(\Phi,\Psi),x)} \vee \neg\varepsilon_0(\Phi,x) \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcup_0}(\mathsf{eu}(\Phi,\Psi),X)$

$\underline{\varepsilon_0(\mathsf{ar}(\Phi,\Psi),x)} \vee \neg\varepsilon_1(\mathsf{ar}(\Phi,\Psi),x,\mathsf{nil})$ $\quad$ $\underline{\varepsilon_1(\mathsf{ar}(\Phi,\Psi),x,Y)} \vee \neg mem(x,Y)$

$\underline{\varepsilon_1(\mathsf{ar}(\Phi,\Psi),x,Y)} \vee \neg\varepsilon_0(\Psi,x) \vee \neg\varepsilon_0(\Phi,x)$

$\underline{\varepsilon_1(\mathsf{ar}(\Phi,\Psi),x,Y)} \vee \neg\varepsilon_0(\Psi,x) \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcap_1}(\mathsf{ar}(\Phi,\Psi),X,\mathsf{con}(x,Y))$

$\underline{\varepsilon_0(\mathsf{er}(\Phi,\Psi),x)} \vee \neg\varepsilon_1(\mathsf{er}(\Phi,\Psi),x,\mathsf{nil})$ $\quad$ $\underline{\varepsilon_1(\mathsf{er}(\Phi,\Psi),x,Y)} \vee \neg mem(x,Y)$

$\underline{\varepsilon_1(\mathsf{er}(\Phi,\Psi),x,Y)} \vee \neg\varepsilon_0(\Psi,x) \vee \neg\varepsilon_0(\Phi,x)$

$\underline{\varepsilon_1(\mathsf{er}(\Phi,\Psi),x,Y)} \vee \neg\varepsilon_0(\Psi,x) \vee \neg r(x,X) \vee \neg\varepsilon_{\sqcup_1}(\mathsf{er}(\Phi,\Psi),X,\mathsf{con}(x,Y))$

$\underline{\varepsilon_{\sqcap_0}(\Phi,\mathsf{nil})}$

$\underline{\varepsilon_{\sqcap_0}(\Phi,\mathsf{con}(x,X))} \vee \neg\varepsilon_0(\Phi,x) \vee \neg\varepsilon_{\sqcap_0}(\Phi,X)$

$\underline{\varepsilon_{\sqcup_0}(\Phi,\mathsf{con}(x,X))} \vee \neg\varepsilon_0(\Phi,x)$ $\quad$ $\underline{\varepsilon_{\sqcup_0}(\Phi,\mathsf{con}(x,X))} \vee \neg\varepsilon_{\sqcup_0}(\Phi,X)$

$\underline{\varepsilon_{\sqcap_1}(\Phi,\mathsf{nil},Y)}$

$\underline{\varepsilon_{\sqcap_1}(\Phi,\mathsf{con}(x,X),Y)} \vee \neg\varepsilon_1(\Phi,x,Y) \vee \neg\varepsilon_{\sqcap_1}(\Phi,X,Y)$

$\underline{\varepsilon_{\sqcup_1}(\Phi,\mathsf{con}(x,X),Y)} \vee \neg\varepsilon_1(\Phi,x,Y)$ $\quad$ $\underline{\varepsilon_{\sqcup_1}(\Phi,\mathsf{con}(x,X),Y)} \vee \neg\varepsilon_{\sqcup_1}(\Phi,X,Y)$

FIGURE 5.1: One-way Clauses of $\mathcal{R}_{CTL}$

5.2 Explicit Model Checking Example

In this section, we present a simple resolution example on explicit model checkig problem, which may help in understanding of the steps of the proof using *Polarized Resolution Modulo*. The inference rules of *Polarized Resolution Modulo* is in Figure 2.3. For convenience, we show the only inference rule used in the following example hereafter.

$$\textbf{Resolution}\ \frac{P \vee C \qquad \neg Q \vee D}{\sigma(C \vee D)}\ \sigma = mgu(P, Q)$$

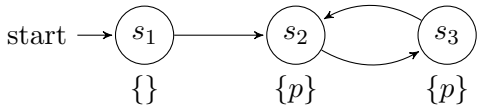


FIGURE 5.2: Explicit State Resolution Example

Example 5.1. *For the transition system M in Figure 5.2, we prove that $M, s_1 \models_a EXEGp$.*

The one-way clauses for the system are:

$$\underline{\neg\varepsilon_0(\overline{p}, s_1)} \qquad \underline{\varepsilon_0(\overline{p}, s_2)} \qquad \underline{\varepsilon_0(\overline{p}, s_3)}$$

$$\underline{r(s_1, \mathsf{con}(s_2, \mathsf{nil}))} \quad \underline{r(s_2, \mathsf{con}(s_3, \mathsf{nil}))} \quad \underline{r(s_3, \mathsf{con}(s_2, \mathsf{nil}))}$$

The translation of $M, s_1 \models_a EXEGp$ is $\varepsilon_0(\mathsf{ex}(\mathsf{eg}(\overline{p})), s_1)$ and the proof starts from

$$\neg\varepsilon_0(\mathsf{ex}(\mathsf{eg}(\overline{p})), s_1).$$

First apply Resolution *rule with one-way clause $\underline{\varepsilon_0(\mathsf{ex}(\Phi), x)} \vee \neg r(x, X) \vee \neg\varepsilon_{\sqcup_0}(\Phi, X)$, with $x = s_1$ and $\Phi = \mathsf{eg}(\overline{p})$, this yields*

$$\neg r(s_1, X) \vee \neg\varepsilon_{\sqcup_0}(\mathsf{eg}(\overline{p}), X).$$

Then apply Resolution *rule with one-way clause $\underline{r(s_1, \mathsf{con}(s_2, \mathsf{nil}))}$, with $X = \mathsf{con}(s_2, \mathsf{nil})$, this yields*

$$\neg\varepsilon_{\sqcup_0}(\mathsf{eg}(\overline{p}), \mathsf{con}(s_2, \mathsf{nil})).$$

Then apply Resolution *rule with one-way clause $\underline{\varepsilon_{\sqcup_0}(\Phi, \mathsf{con}(x, X))} \vee \neg\varepsilon_0(\Phi, x)$, with $x = s_2$, $X = \mathsf{nil}$ and $\Phi = \mathsf{eg}(\overline{p})$, this yields*

$$\neg\varepsilon_0(\mathsf{eg}(\overline{p}), s_2).$$

Then apply Resolution *rule with one-way clause* $\underline{\varepsilon_0(\mathsf{eg}(\Phi), x)} \vee \neg\varepsilon_1(\mathsf{eg}(\Phi), x, \mathsf{nil})$*, with* $\Phi = \overline{p}$ *and* $x = s_2$*, this yields*

$$\neg\varepsilon_1(\mathsf{eg}(\overline{p}), s_2, \mathsf{nil}).$$

Then apply Resolution *rule with one-way clause* $\underline{\varepsilon_1(\mathsf{eg}(\Phi), x, Y)} \vee \neg\varepsilon_0(\Phi, x) \vee \neg r(x, X) \vee \neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\Phi), X, \mathsf{con}(x, Y))$*, with* $\Phi = \overline{p}$*,* $x = s_2$ *and* $Y = \mathsf{nil}$*, this yields*

$$\neg\varepsilon_0(\overline{p}, s_2) \vee \neg r(s_2, X) \vee \neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\overline{p}), X, \mathsf{con}(s_2, \mathsf{nil})).$$

Then apply Resolution *rule with one-way clause* $\underline{\varepsilon_0(\overline{p}, s_2)}$*, this yields*

$$\neg r(s_2, X) \vee \neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\overline{p}), X, \mathsf{con}(s_2, \mathsf{nil})).$$

Then apply Resolution *rule with one-way clause* $\underline{r(s_2, \mathsf{con}(s_3, \mathsf{nil}))}$*, with* $X = \mathsf{con}(s_3, \mathsf{nil})$*, this yields*

$$\neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\overline{p}), \mathsf{con}(s_3, \mathsf{nil}), \mathsf{con}(s_2, \mathsf{nil})).$$

Then apply Resolution *rule with one-way clause* $\underline{\varepsilon_{\sqcup_1}(\Phi, \mathsf{con}(x, X), Y)} \vee \neg\varepsilon_1(\Phi, x, Y)$*, with* $\Phi = \mathsf{eg}(\overline{p})$*,* $x = s_3$*,* $X = \mathsf{nil}$ *and* $Y = \mathsf{con}(s_2, \mathsf{nil})$*, this yields*

$$\varepsilon_1(\mathsf{eg}(\overline{p}), s_3, \mathsf{con}(s_2, \mathsf{nil})).$$

Then apply Resolution *rule with one-way clause* $\underline{\varepsilon_1(\mathsf{eg}(\Phi), x, Y)} \vee \neg\varepsilon_0(\Phi, x) \vee \neg r(x, X) \vee \neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\Phi), X, \mathsf{con}(x, Y))$*, with* $\Phi = \overline{p}$*,* $x = s_3$ *and* $Y = \mathsf{con}(s_2, \mathsf{nil})$*, this yields*

$$\neg\varepsilon_0(\overline{p}, s_3) \vee \neg r(s_3, X) \vee \neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\overline{p}), X, \mathsf{con}(s_3, \mathsf{con}(s_2, \mathsf{nil}))).$$

Then apply Resolution *rule with one-way clause* $\underline{\varepsilon_0(\overline{p}, s_3)}$*, this yields*

$$\neg r(s_3, X) \vee \neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\overline{p}), X, \mathsf{con}(s_3, \mathsf{con}(s_2, \mathsf{nil}))).$$

Then apply Resolution *rule with one-way clause* $\underline{r(s_3, \mathsf{con}(s_2, \mathsf{nil}))}$*, with* $X = \mathsf{con}(s_2, \mathsf{nil})$*, this yields*

$$\neg\varepsilon_{\sqcup_1}(\mathsf{eg}(\overline{p}), \mathsf{con}(s_2, \mathsf{nil}), \mathsf{con}(s_3, \mathsf{con}(s_2, \mathsf{nil}))).$$

Then apply Resolution *rule with one-way clause* $\underline{\varepsilon_{\sqcup_1}(\Phi, \mathsf{con}(x, X), Y)} \vee \neg\varepsilon_1(\Phi, x, Y)$*, with* $\Phi = \mathsf{eg}(\overline{p})$*,* $x = s_3$*,* $X = \mathsf{nil}$ *and* $Y = \mathsf{con}(s_2, \mathsf{nil})$*, this yields*

$$\neg\varepsilon_1(\mathsf{eg}(\overline{p}), s_2, \mathsf{con}(s_3, \mathsf{con}(s_2, \mathsf{nil}))).$$

Then apply Resolution *rule with one-way clause* $\underline{\varepsilon_1(\mathsf{eg}(\Phi), x, Y)} \vee \neg mem(x, Y)$*, with* $x = s_2$ *and* $Y = \mathsf{con}(s_3, \mathsf{con}(s_2, \mathsf{nil}))$*, this yields*

$$\neg mem(s_2, \mathsf{con}(s_3, \mathsf{con}(s_2, \mathsf{nil}))).$$

Then apply Resolution *rule with one-way clause* $\underline{mem(x, \mathsf{con}(y, Z))} \vee \neg mem(x, Z)$*, with* $x = s_2$*,* $y = s_3$ *and* $Z = \mathsf{con}(s_2, \mathsf{nil})$*, this yields*

$$\neg mem(s_2, \mathsf{con}(s_2, \mathsf{nil})).$$

Then apply Resolution *rule with one-way clause* $\underline{mem(x, \mathsf{con}(x, Z))}$*, with* $x = s_2$ *and* $Z = \mathsf{nil}$*, this yields the empty clause. Thus* $M, s_1 \models_a EXEGp$ *holds.*

In this example, the reolution between an ordinary clause and a one-way clause is in fact an application of Extended Narrowing rule in PRM.

5.3 Symbolic Model Checking with Resolution Modulo

In Section 5.2, the states are represented by constants. However, in real designs the set of states may be very large and the size of the axioms to denote the transitions will increase exponentially. One good thing for the real designs is that some rules can be found for the set of states that holds on the same atomic propositions and for the transition relations between two set of states. Thus, for testing cases of real designs, it is convenient to represent a state by a function symbol with a set of boolean variables. To represent a Kripke structure $M = (S, \mathsf{next}, L)$ using boolean vectors, we must describe the set S, the relation next and the mapping L.

More Rewriting vs. More Parameters

The First Encoding Method Initially, the symbolically encoding of a state should be given. Intuitively, each bit of the boolean vector is represented by a boolean variable. Assume that the number of all the states is n and $2^{m-1} < n \leq 2^m$, then each state can be represented by $s(B_1, ..., B_m)$, in which B_i is a boolean variable for the terms tt and ff. In this kind a representation, normally each atomic proposition related to a boolean variable in the state. For example, in the state $s(B_1, ..., B_i, ..., B_m)$, B_i is related to the atomic proposition p_i, which means that p_i holds on the state $s(B_1, ..., B_i, ..., B_m)$ if and only if B_i is assigned to tt. Thus, the set of states which satisfy p_i can be represented as $s(B_1, ..., tt, ..., B_m)$. For the transition relations, if the state with the negation of

B_i and B_j is always a successor of $s(B_1, ..., B_i, ..., B_j, ..., B_m)$, then this successor can be represented as $s(B_1, ..., \mathsf{not}(B_i), ..., \mathsf{not}(B_j), ..., B_m)$. However, as the function symbol "not" is introduced to represent the relations, two term rewrite rules should be considered to reduce the term $\mathsf{not}(B)$, that is $\mathsf{not}(tt) \hookrightarrow ff$ and $\mathsf{not}(ff) \hookrightarrow tt$. As is said in Section 2, term rewrite rules are not considered in this dissertation, these two term rewrite rules are replaced by the rules $eq(\mathsf{not}(tt), ff) \hookrightarrow \top$ and $eq(\mathsf{not}(ff), tt) \hookrightarrow \top$.

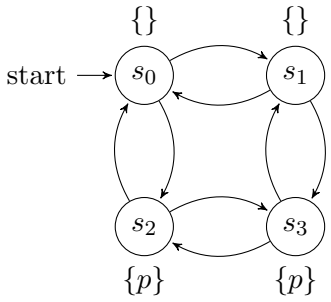


FIGURE 5.3: Symbolic Representation Example

Example 5.2. *The Kripke structure in Figure 5.3 can be expressed as follows:*

States $s_0 : s(ff, ff)$ $s_1 : s(ff, tt)$

$s_2 : s(tt, ff)$ $s_3 : s(tt, tt)$

Atomic Prop. $\varepsilon_0(\overline{p}, s(tt, B_2))$

Relations

$r(s(B_1, B_2), \mathsf{con}(s(B_1', B_2), \mathsf{con}(s(B_1, B_2'), \mathsf{nil}))) \vee \neg eq(\mathsf{not}(B_1), B_1') \vee \neg eq(\mathsf{not}(B_1), B_2')$

The Second Encoding Method In this method, each bit of the boolean vector is represented by a term $b(T_i, F_i)$, in which T_i and F_i are boolean variables, but can only be assigned to different values. Thus the vector with m bits can be represented by $s(b(T_1, F_1), ..., b(T_i, F_i), ..., b(T_m, F_m))$. And the set of states which satisfy p_i can be represented as $s(b(T_1, F_1), ..., b(tt, ff), ..., b(T_m, F_m))$. In this way of encoding, the negation of $b(T_i, F_i)$ can be represented as $b(F_i, T_i)$.

Example 5.3. *The Kripke structure in Figure 5.3 can be expressed as follows:*

States $s_0 : s(b(ff, tt), b(ff, tt))$ $s_1 : s(b(ff, tt), b(tt, ff))$

$s_2 : s(b(tt, ff), b(ff, tt))$ $s_3 : s(b(tt, ff), b(tt, ff))$

Atomic Prop. $\varepsilon_0(\overline{p}, s(b(tt, ff), B_2))$

Relations

$r(s(b(T_1, F_1), b(T_2, F_2)), \mathsf{con}(s(b(F_1, T_1), b(T_2, F_2)), \mathsf{con}(s(b(T_1, F_1), b(F_2, T_2)), \mathsf{nil})))$

Remark1 In the first encoding method, the state can be represented using less boolean variables. But when we encode the relations, some rewrite rules should be taken into account, to reduce the boolean terms into constants. In the second encoding method, the term rewrite rules are avoided by replacing each boolean variable with a term containing two opposite boolean variables. The experiments shows that our second encoding method runs faster than the first one. However, the term rewrite rules cannot be completely avoided in some cases, which will be shown in the later sections.

Remark2 Our method of encoding the systems is similar to the idea of QBF. In QBF, each proposition variable has two values, while in this chapter, each boolean varible can only be assigned to tt and ff.

5.4 Selection Function

iProver Modulo is an automated theorem prover based on ordered polarized resolution modulo. In this section, we specify a refinement of resolution by means of a selection function δ mapping each ordinary clause C to a subset of literals, which have priority to apply resolution rules.

Depth of CTL Terms The depth of a CTL term is as follows:

$\mathsf{dp}(\overline{p}) = 0,\ p \in AP$	$\mathsf{dp}(\mathsf{not}(\overline{p})) = 0\ ,\ p \in AP$
$\mathsf{dp}(\mathsf{ax}(\lvert\varphi\rvert)) = \mathsf{dp}(\lvert\varphi\rvert) + 1$	$\mathsf{dp}(\mathsf{and}(\lvert\varphi\rvert, \lvert\psi\rvert)) = \mathsf{max}(\mathsf{dp}(\lvert\varphi\rvert), \mathsf{dp}(\lvert\psi\rvert)) + 1$
$\mathsf{dp}(\mathsf{ex}(\lvert\varphi\rvert)) = \mathsf{dp}(\lvert\varphi\rvert) + 1$	$\mathsf{dp}(\mathsf{or}(\lvert\varphi\rvert, \lvert\psi\rvert)) = \mathsf{max}(\mathsf{dp}(\lvert\varphi\rvert), \mathsf{dp}(\lvert\psi\rvert)) + 1$
$\mathsf{dp}(\mathsf{af}(\lvert\varphi\rvert)) = \mathsf{dp}(\lvert\varphi\rvert) + 1$	$\mathsf{dp}(\mathsf{au}(\lvert\varphi\rvert, \lvert\psi\rvert)) = \mathsf{max}(\mathsf{dp}(\lvert\varphi\rvert), \mathsf{dp}(\lvert\psi\rvert)) + 1$
$\mathsf{dp}(\mathsf{ag}(\lvert\varphi\rvert)) = \mathsf{dp}(\lvert\varphi\rvert) + 1$	$\mathsf{dp}(\mathsf{ar}(\lvert\varphi\rvert, \lvert\psi\rvert)) = \mathsf{max}(\mathsf{dp}(\lvert\varphi\rvert), \mathsf{dp}(\lvert\psi\rvert)) + 1$
$\mathsf{dp}(\mathsf{ef}(\lvert\varphi\rvert)) = \mathsf{dp}(\lvert\varphi\rvert) + 1$	$\mathsf{dp}(\mathsf{eu}(\lvert\varphi\rvert, \lvert\psi\rvert)) = \mathsf{max}(\mathsf{dp}(\lvert\varphi\rvert), \mathsf{dp}(\lvert\psi\rvert)) + 1$
$\mathsf{dp}(\mathsf{eg}(\lvert\varphi\rvert)) = \mathsf{dp}(\lvert\varphi\rvert) + 1$	$\mathsf{dp}(\mathsf{er}(\lvert\varphi\rvert, \lvert\psi\rvert)) = \mathsf{max}(\mathsf{dp}(\lvert\varphi\rvert), \mathsf{dp}(\lvert\psi\rvert)) + 1$

Literals During the resolution steps, two kinds of literals in the ordinary clauses may appear, which are shown as follows:

- From the set of one-way clauses showed in Section 5.1, the literals may appear in the ordinary clauses are of the form: $\neg\varepsilon_0(\Phi, x)$, $\neg\varepsilon_{\sqcup_0}(\Phi, X)$, $\neg\varepsilon_{\sqcap_0}(\Phi, X)$, $\neg\varepsilon_1(\Phi, x, Y)$, $\neg\varepsilon_{\sqcup_1}(\Phi, X, Y)$, $\neg\varepsilon_{\sqcap_1}(\Phi, X, Y)$, $\neg r(x, X)$, $\neg mem(x, X)$.

- When term rewrite rules are used, for example, $s_{old} \hookrightarrow s_{new}$ is a term rewrite rule to rewrite the old expression s_{old} for the state s into a new expression s_{new}, the term rewrite rules are replaced by the formulas of the form $eq(l, r)$, in which l is the left-hand side of the rewrite rule, and r is the right hand-side of the rewrite rule.

Definition 5.1 (Selection Function for CTL Model Checking)**.** The selection function δ for CTL model checking is defined as follows:

$$\delta(C) = \begin{cases} a = \{l \mid l \text{ is negative literal with the predicate } eq\}, & a \neq \emptyset \\ b = \{l \mid l \text{ is negative literal with the predicate } mem\}, & b \neq \emptyset \\ c = \{l \mid l \text{ is negative literal with the predicate } r\}, & c \neq \emptyset \\ d = \mathsf{sel\_ctl}(C), \end{cases}$$

in which $\mathsf{sel\_ctl}(C)$ is defined by lexicographical order as follows:

$$\mathsf{sel\_ctl}(C) = \begin{cases} e = \{l \mid l \text{ is negative literal with the predicate } \varepsilon_0, \varepsilon_1 \\ \qquad \text{and } l \in \mathsf{min\_ctl}(C)\}, & e \neq \emptyset \\ f = \{l \mid l \text{ is negative literal with the predicate } \varepsilon_{\sqcup_0}, \varepsilon_{\sqcap_0}, \\ \qquad \varepsilon_{\sqcup_1}, \varepsilon_{\sqcap_1} \text{ and } l \in \mathsf{min\_ctl}(C)\}, \end{cases}$$

in which $\mathsf{min\_ctl}(C)$ is defined as follows:

$$\mathsf{min\_ctl}(C) = \{l \mid \text{the depth of the CTL term in } l \text{ is the minimum of all the CTL terms in } C\}$$

Now we prove the completeness of the Polarized Resolution Modulo with Selection Function δ (PRM^{δ}). The following theorem shows that Ordered Polarized Resolution Modulo is complete if the ordering $\succ$ on literals is well-founded and stable by substitution.

Theorem 5.2 ([Bur10])**.** *Given a set of clauses* Γ*, if* $\Gamma \vdash^{cf}_{\mathcal{R}}$ *then* $\Gamma \mapsto^{\succ}_{\mathcal{R}} \square$.

The completeness of our strategy is as follows.

Theorem 5.3 (Completeness)**.** *Given a CTL formula* φ *of NNF, if* $\{\neg\varepsilon_0(|\varphi|, s)\} \hookrightarrow_{\mathcal{R}} \square$*, then* $\{\neg\varepsilon_0(|\varphi|, s)\} \hookrightarrow^{\delta}_{\mathcal{R}} \square$.

Proof. To prove this theorem, we just need to prove that the ordering of literals defined by δ in an ordinary clause is well-founded and stable by substitution. In the definition of

δ, the order of the literals with CTL terms are depended on the depth of the CTL terms, thus ordering defined by δ is well founded. Still, by the definition of δ, the order of the literals in an ordinary clause is only depended on the form of the predicates and the CTL terms in which there are no variables. Thus, the ordering is stable by substitution. □

5.5 Implementation and Experimental Evaluation

The model checking approach has been implemented in the automated theorem prover iProver Modulo, and an experimental evaluation has been carried out. In this section, we first present an example with implementation steps in detail, then the experimental evaluation on two kinds of concurrent programs is illustrated.

5.5.1 An Implementation Example

The example is a *River Crossing Puzzle* problem [Bri13]. The description of this problem is as follows.

The Wolf-Goat-Cabbage Puzzle *A man once had to travel with a wolf, a goat and a cabbage. He had to take good care of them, since the wolf would like to eat the goat if he would get the chance, while the goat appeared to long for a tasty cabbage. After some traveling, he suddenly stood before a river. This river could only be crossed using the small boat laying nearby at a shore. The boat was only good enough to take himself and one of his loads across the river. The other two subjects/objects he had to leave on their own. How must the man row across the river back and forth, to take himself as well as his luggage safe to the other side of the river, without having one eating another?*

The process of solving this problem follows the process of traditional model checking problems. First, to build a transition system for this problem. Then, give a specification for the problem to be solved. The last step is to verify the specification automatically.

Modeling There are four subjects/objects in the problem: Man, Wolf, Goat, Cabbage. Thus each state can be represented by assigning the variables of the term $s(M, W, G, C)$ to $b(tt, ff)$ or $b(ff, tt)$. For example, $s(b(ff, tt), b(ff, tt), b(ff, tt), b(ff, tt))$ is the initial state. The set of atomic propositions is $\{p_m, p_w, p_g, p_c\}$. For each state s, the atomic proposition $p_x \in L(s)$ means that the subject/object x has crossed the river. Thus whether each proposition p_x holds on a state can be expressed by the following axioms (written in the form of one-way clauses for iProver Modulo).

```
cnf(prop1, axiom,  pi0(pm, s(b(tt,ff), W, G, C))).
cnf(prop2, axiom, ~pi0(pm, s(b(ff,tt), W, G, C))).
cnf(prop3, axiom,  pi0(pw, s(M, b(tt,ff), G, C))).
cnf(prop4, axiom, ~pi0(pw, s(M, b(ff,tt), G, C))).
cnf(prop5, axiom,  pi0(pg, s(M, W, b(tt,ff), C))).
cnf(prop6, axiom, ~pi0(pg, s(M, W, b(ff,tt), C))).
cnf(prop7, axiom,  pi0(pc, s(M, W, G, b(tt,ff)))).
cnf(prop8, axiom, ~pi0(pc, s(M, W, G, b(ff,tt)))).
```

For each state, the set of its successors can be represented by the following axioms.

```
cnf(r1, axiom, r(s(b(T,F), b(F,T), b(F,T), b(F,T)),
                con(s(b(F,T), b(F,T), b(F,T), b(F,T)),nil))).
cnf(r2, axiom, r(s(b(T,F), b(T,F), b(F,T), b(F,T)),
                con(s(b(F,T), b(T,F), b(F,T), b(F,T)),
                con(s(b(F,T), b(F,T), b(F,T), b(F,T)),nil)))).
cnf(r3, axiom, r(s(b(T,F), b(F,T), b(T,F), b(F,T)),
                con(s(b(F,T), b(F,T), b(T,F), b(F,T)),
                con(s(b(F,T), b(F,T), b(F,T), b(F,T)), nil)))).
cnf(r4, axiom, r(s(b(T,F), b(F,T), b(F,T), b(T,F)),
                con(s(b(F,T), b(F,T), b(F,T), b(T,F)),
                con(s(b(F,T), b(F,T), b(F,T), b(F,T)), nil)))).
cnf(r5, axiom, r(s(b(T,F), b(T,F), b(T,F), b(F,T)),
                con(s(b(F,T), b(T,F), b(T,F), b(F,T)),
                con(s(b(F,T), b(F,T), b(T,F), b(F,T)),
                con(s(b(F,T), b(T,F), b(F,T), b(F,T)), nil))))).
cnf(r6, axiom, r(s(b(T,F), b(T,F), b(F,T), b(T,F)),
                con(s(b(F,T), b(T,F), b(F,T), b(T,F)),
                con(s(b(F,T), b(F,T), b(F,T), b(T,F)),
                con(s(b(F,T), b(T,F), b(F,T), b(F,T)), nil))))).
cnf(r7, axiom, r(s(b(T,F), b(F,T), b(T,F), b(T,F)),
                con(s(b(F,T), b(F,T), b(T,F), b(T,F)),
                con(s(b(F,T), b(F,T), b(F,T), b(T,F)),
                con(s(b(F,T), b(F,T), b(T,F), b(F,T)), nil))))).
cnf(r8, axiom, r(s(b(T,F), b(T,F), b(T,F), b(T,F)),
                con(s(b(F,T), b(T,F), b(T,F), b(T,F)),
                con(s(b(F,T), b(F,T), b(T,F), b(T,F)),
                con(s(b(F,T), b(T,F), b(F,T), b(T,F)),
```

```
                con(s(b(F,T), b(T,F), b(T,F), b(F,T)), nil))))))).
```

Specification The question above can be specialized as follows:

$$M, s \models_a EU((p_w \wedge p_g \Rightarrow p_m) \wedge (\neg p_w \wedge \neg p_g \Rightarrow \neg p_m) \wedge (p_c \wedge p_g \Rightarrow p_m) \wedge (\neg p_c \wedge \neg p_g \Rightarrow \neg p_m), p_c \wedge p_g \wedge p_w \wedge p_m)$$

where s is the initial state. After translating the CTL formula into CTL-term, the negation form of the specification of the problem is presented as follows.

```
cnf(check, negated_conjecture,
    ~pi0(eu(and(or(not(pw),or(not(pg),pm)),and(or(pw,or(pg,not(pm))),
            and(or(not(pc),or(not(pg),pm)),or(pc,or(pg,not(pm))))))),
            and(pc,and(pg,and(pw,pm)))),
         s(b(ff,tt), b(ff,tt), b(ff,tt), b(ff,tt)))).
```

Verification Suppose that the one-way clauses and the clause of the negation form of the specification is contained in the file "wgc.p". The command to be used is as follows:

```
iproveropt ‘cat basic_resolution_options‘ --modulo true
--res_passive_queue_flag false --res_lit_sel ctl_sel
--res_out_proof true wgc.p
```

The verification result says that an empty clause is derived, which means that there exists a way to take all the subjects/objects safe to the other side of the river. By checking the proof steps from the beginning to the end, the following transitions are carried out, which is one of the solutions. Assume the original side of the river is A, the other side of the river is B.

```
s(b(ff,tt),b(ff,tt),b(ff,tt),b(ff,tt))
       Man takes Goat to B
s(b(tt,ff),b(ff,tt),b(tt,ff),b(ff,tt))
       Man goes back to A
s(b(ff,tt),b(ff,tt),b(tt,ff),b(ff,tt))
       Man takes Wolf to B
s(b(tt,ff),b(tt,ff),b(tt,ff),b(ff,tt))
       Man takes Goat back to A
```

```
s(b(ff,tt),b(tt,ff),b(ff,tt),b(ff,tt))
       Man takes Cabbage to B
s(b(tt,ff),b(tt,ff),b(ff,tt),b(tt,ff))
       Man goes back to A
s(b(ff,tt),b(tt,ff),b(ff,tt),b(tt,ff))
       Man takes Goat to B
s(b(tt,ff),b(tt,ff),b(tt,ff),b(tt,ff))
```

5.5.2 Experimental Evaluation

In this subsection, we give a comparison among

- Resolution Modulo method, which is implemented in iProver Modulo [Bur11];
- QBF-based method, which is implemented in VERDS [Zha12];
- and traditional symbolic model checking method, which is implemented in the famous tool NuSMV [CCGR99] version 2.5.4.

iProver Modulo is a prover by embedding Polarized Resolution Modulo into iProver [Kor08]. The comparison is by proving 24 CTL properties on two kinds of programs: *Programs with Concurrent Processes* and *Programs with Concurrent Sequential Processes*. All the programs and properties are from [Zha14]. The programs and properties are described as follows.

Programs with Concurrent Processes The parameters of the this kind of boolean programs are as follows:

a: the number of processes,

b: the number of all the boolean variables,

c: the number of shared boolean variables,

d: the number of local boolean variables in each process.

Initially, the shared boolean variables are set to a random value in $\{0, 1\}$, and the local boolean variables are set to 0. The behavior of each process is assign a new value to each variable in the process. The new value is the negation of a variable that are randomly chosen from the shared and local variables. A simple example is in Figure 5.4.

```
MODULE main              |  MODULE p1(v1,v2) |  MODULE p2(v1,v2)
VAR                      |  VAR              |  VAR
 v1:boolean ;            |   v4:boolean ;    |   v5:boolean ;
 v2:boolean ;            |  ASSIGN           |  ASSIGN
 v3:boolean ;            |   init(v4):=0;    |   init(v5):=0;
 p1:process p1(v1,v2);   |   next(v1):=!v4;  |   next(v1):=!v2;
 p2:process p2(v1,v2);   |   next(v2):=!v1;  |   next(v2):=!v5;
ASSIGN                   |   next(v4):=!v1;  |   next(v5):=!v1;
 init(v1):=1;            |                   |
 init(v2):=0;            |                   |
 init(v3):=0;            |                   |
 next(v1):=!v2;          |                   |
 next(v2):=!v3;          |                   |
 next(v3):=!v1;          |                   |
```

FIGURE 5.4: Program with Concurrent Processes

In this program, there are three process, two shared Boolean variables, and one local boolean variable in each process.

Programs with Concurrent Sequential Processes In this kind of programs, in addition to the parameters a, b, c, d specified above, the other parameters are specified as follows:

t: the number of transitions in a process,

p: the number of parallel assignments in each transition.

In each concurrent sequential process, besides the boolean variables, there is a local variable, which is used to represent the program locations, with t possible values. The shared boolean variables are initially set to random values in $\{0, 1\}$ and local variables in each process to 0. For each transition in a process, p pairs of shared boolean and local boolean variables are randomly chosen among the shared and local boolean variables, such that the first element of each pair is assigned a new value, which is the negation of the second element of the pair. The transitions are numbered from 0 to $t - 1$, and are executed consecutively. When the end of the sequence of the transitions is reached, then jump to the beginning of the sequence. A simple example is given in Figure 5.5.

In this program, there are two processes (the main processes does not included), three shared boolean variables, one local boolean variable for each process, and one local variable for the program location of each process.

```
MODULE main                 | MODULE p1(v1,v2,v3)| MODULE p2(v1,v2,v3)
VAR                         | VAR                | VAR
 v1:boolean ;               |  v4:boolean ;      |  v5:boolean ;
 v2:boolean ;               |  cp:{c0,c1,c2};    |  cp:{c0,c1,c2};
 v3:boolean ;               | ASSIGN             | ASSIGN
 p1:process p1(v1,v2,v3);|  init(v4):=0;      |  init(v5):=0;
 p2:process p2(v1,v2,v3);|  init(cp):=c0;     |  init(cp):=c0;
ASSIGN                      |  case              |  case
 init(v1):=1;               |   cp=c0:           |   cp=c0:
 init(v2):=0;               |    next(cp):= c1;  |    next(cp):= c1;
 init(v3):=1;               |    next(v1):= !v2; |    next(v2):= !v5;
                            |    next(v2):= !v4; |    next(v3):= !v1;
                            |   cp=c1:          |   cp=c1:
                            |    next(cp):= c2;  |    next(cp):= c2;
                            |    next(v1):= !v3; |    next(v1):= !v3;
                            |    next(v4):= !v2; |    next(v2):= !v5;
                            |   cp=c2:          |   cp=c2:
                            |    next(cp):= c0;  |    next(cp):= c0;
                            |    next(v2):= !v3; |    next(v5):= !v3;
                            |    next(v3):= !v1; |    next(v3):= !v1;
                            |  esac              |  esac
```

FIGURE 5.5: Program with Concurrent Sequential Processes

Temporal Properties The properties tested in the experiment are as follows.

$p_{01} : AG(\bigvee_{i=1}^{c} v_i)$	$p_{13} : AG(\bigwedge_{i=1}^{c} v_i)$
$p_{01} : AF(\bigvee_{i=1}^{c} v_i)$	$p_{14} : AF(\bigwedge_{i=1}^{c} v_i)$
$p_{03} : AG(v_1 \Rightarrow AF(v_2 \wedge \bigvee_{i=3}^{c} v_i))$	$p_{15} : AG(v_1 \Rightarrow AF(v_2 \vee \bigwedge_{i=3}^{c} v_i))$
$p_{04} : AG(v_1 \Rightarrow EF(v_2 \wedge \bigvee_{i=3}^{c} v_i))$	$p_{16} : AG(v_1 \Rightarrow EF(v_2 \vee \bigwedge_{i=3}^{c} v_i))$
$p_{05} : EG(v_1 \Rightarrow AF(v_2 \wedge \bigvee_{i=3}^{c} v_i))$	$p_{17} : EG(v_1 \Rightarrow AF(v_2 \vee \bigwedge_{i=3}^{c} v_i))$
$p_{06} : EG(v_1 \Rightarrow EF(v_2 \wedge \bigvee_{i=3}^{c} v_i))$	$p_{18} : EG(v_1 \Rightarrow EF(v_2 \vee \bigwedge_{i=3}^{c} v_i))$
$p_{07} : AU(v_1, AU(v_2, \bigvee_{i=3}^{c} v_i))$	$p_{19} : AU(v_1, AU(v_2, \bigwedge_{i=3}^{c} v_i))$
$p_{08} : AU(v_1, EU(v_2, \bigvee_{i=3}^{c} v_i))$	$p_{20} : AU(v_1, EU(v_2, \bigwedge_{i=3}^{c} v_i))$
$p_{09} : AU(v_1, AR(v_2, \bigvee_{i=3}^{c} v_i))$	$p_{21} : AU(v_1, AR(v_2, \bigwedge_{i=3}^{c} v_i))$
$p_{10} : AU(v_1, ER(v_2, \bigvee_{i=3}^{c} v_i))$	$p_{22} : AU(v_1, ER(v_2, \bigwedge_{i=3}^{c} v_i))$
$p_{11} : AR(AXv_1, AXAU(v_2, \bigvee_{i=3}^{c} v_i))$	$p_{23} : AR(AXv_1, AXAU(v_2, \bigwedge_{i=3}^{c} v_i))$
$p_{12} : AR(EXv_1, EXEU(v_2, \bigvee_{i=3}^{c} v_i))$	$p_{24} : AR(EXv_1, EXEU(v_2, \bigwedge_{i=3}^{c} v_i))$

Experimental Data

All the cases are tested on Intel® Core ™ i5-2400 CPU @ 3.10GHz × 4 with Linux and the testing time of each case is limited to 20 minutes. The comparison is based on two aspects: the number of testing cases that can be proved, and the time used if a problem can be proved in both.

Experimental Data for Programs with Concurrent Processes For this kind of programs, each testing case contains three processes ($a = 3$), and the number of all the boolean variables b vary over $\{12, 24\}$. Moreover, $c = b/2$, $d = c/a$. Each property is tested on 20 testing cases for each value of b. The experimental data is presented in Table 5.1 and 5.2. For the 960 testing cases of this kind of programs, 892 of them are solved by iProver Modulo, 861 of them are solved by VERDS, while in NuSMV, all of them are provable. For the testing cases that are both provable by iProver Modulo and VERDS, 216 of them run faster in iProver Modulo, while 448 of them have advantage in VERDS. For the testing cases that are both provable by iProver Modulo and NuSMV, 342 of them run faster in iProver Modulo, while 469 of them have advantage in NuSMV.

Experimental Data for Programs with Concurrent Sequential Processes For this kind of programs, each testing case contains two processes ($a = 2$), and the number of all the boolean variables b vary over $\{12, 16\}$. Moreover, $c = b/2$, $d = c/a$, $t = c$ and $p = 4$. Each property is tested on 20 testing cases for each value of b. The experimental data is presented in Table 5.3 and 5.4. For the 960 testing cases of this kind of programs, 816 of them are solved by iProver Modulo, 700 of them are solved by VERDS, while in NuSMV, all of them are provable. For the testing cases that are both provable by iProver Modulo and VERDS, 434 of them run faster in iProver Modulo, while 141 of them have advantage in VERDS. For the testing cases that are both provable by iProver Modulo and NuSMV, 516 of them run faster in iProver Modulo, while 290 of them have advantage in NuSMV.

Table 5.1: Experimental Results of Programs with Concurrent Processes

Tools		iProver Modulo			VERDS			NuSMV		
Prop	**Num**	**True**	**False**	**>20m**	**True**	**False**	**>20m**	**True**	**False**	**>20m**
p_{01}	40	-	40	-	-	40	-	-	40	-
p_{02}	40	40	-	-	40	-	-	40	-	-
p_{03}	40	2	37	1	-	37	3	3	37	-
p_{04}	40	19	-	21	-	-	40	40	-	-
p_{05}	40	31	6	3	34	5	1	34	6	-
p_{06}	40	38	-	2	40	-	-	40	-	-
p_{07}	40	40	-	-	40	-	-	40	-	-
p_{08}	40	40	-	-	40	-	-	40	-	-
p_{09}	40	32	8	-	32	8	-	32	8	-
p_{10}	40	40	-	-	40	-	-	40	-	-
p_{11}	40	10	30	-	10	30	-	10	30	-
p_{12}	40	40	-	-	40	-	-	40	-	-
p_{13}	40	-	40	-	-	40	-	-	40	-
p_{14}	40	3	37	-	3	37	-	3	37	-
p_{15}	40	5	33	2	-	33	7	7	33	-
p_{16}	40	19	-	21	-	-	40	40	-	-
p_{17}	40	34	3	3	37	2	1	37	3	-
p_{18}	40	38	-	2	40	-	-	40	-	-
p_{19}	40	5	35	-	5	35	-	5	35	-
p_{20}	40	15	20	5	17	21	2	17	23	-
p_{21}	40	3	37	-	3	37	-	3	37	-
p_{22}	40	3	37	-	3	37	-	3	37	-
p_{23}	40	-	40	-	-	40	-	-	40	-
p_{24}	40	20	12	8	25	10	5	25	15	-
Sum	960	477	415	68	449	412	99	539	421	-

Table 5.2: Speed Comparison of Programs with Concurrent Processes

Tools		iProver Modulo/VERDS			iProver Modulo/NuSMV		
Prop	**Num**	**True**	**False**	**Only**	**True**	**False**	**Only**
p_{01}	40	-	7/23	-	-	18/19	-
p_{02}	40	30/1	-	-	23/5	-	-
p_{03}	40	-	4/32	2/-	-/2	6/30	-/1
p_{04}	40	-	-	19/-	-/19	-	-/21
p_{05}	40	-/30	3/-	1/3	2/29	1/5	-/3
p_{06}	40	-/37	-	-/2	4/34	-	-/2
p_{07}	40	18/7	-	-	23/11	-	-
p_{08}	40	18/7	-	-	20/15	-	-
p_{09}	40	12/12	1/7	-	19/6	1/7	-
p_{10}	40	11/12	-	-	20/11	-	-
p_{11}	40	-/6	3/26	-	7/3	10/20	-
p_{12}	40	10/11	-	-	20/11	-	-
p_{13}	40	-	28/1	-	-	23/10	-
p_{14}	40	2/1	-/34	-	1/2	17/19	-
p_{15}	40	-	2/30	5/-	-/5	6/27	-/2
p_{16}	40	-	-	19/-	-/19	-	-/21
p_{17}	40	1/33	1/1	1/3	2/32	1/2	-/3
p_{18}	40	1/37	-/2	-/2	4/34	-	-/2
p_{19}	40	1/3	8/16	-	1/2	19/12	-
p_{20}	40	1/11	4/13	-/3	3/11	13/6	-/5
p_{21}	40	1/1	16/13	-	1/1	22/12	-
p_{22}	40	1/1	12/10	-	1/1	21/15	-
p_{23}	40	-	18/5	-	-	23/12	-
p_{24}	40	-/18	2/7	2/5	4/15	6/5	-/8
Sum	960	107/228	109/220	49/18	155/268	187/201	-/68

Table 5.3: Experimental Results of Programs with Concurrent Sequential Processes

Tools		iProver Modulo			VERDS			NuSMV		
Prop	**Num**	**True**	**False**	**>20m**	**True**	**False**	**>20m**	**True**	**False**	**>20m**
p_{01}	40	29	6	5	-	4	36	34	6	-
p_{02}	40	40	-	-	40	-	-	40	-	-
p_{03}	40	-	25	15	-	15	25	9	31	-
p_{04}	40	12	-	28	-	-	40	40	-	-
p_{05}	40	21	8	11	24	2	14	32	8	-
p_{06}	40	36	-	4	31	-	9	40	-	-
p_{07}	40	40	-	-	40	-	-	40	-	-
p_{08}	40	40	-	-	40	-	-	40	-	-
p_{09}	40	35	5	-	29	1	10	35	5	-
p_{10}	40	40	-	-	40	-	-	40	-	-
p_{11}	40	30	9	1	23	4	13	31	9	-
p_{12}	40	40	-	-	35	-	5	40	-	-
p_{13}	40	-	40	-	-	40	-	-	40	-
p_{14}	40	3	37	-	3	33	4	3	37	-
p_{15}	40	-	23	17	-	15	25	9	31	-
p_{16}	40	13	-	27	-	-	40	40	-	-
p_{17}	40	22	5	13	26	1	13	34	6	-
p_{18}	40	36	-	4	31	-	9	40	-	-
p_{19}	40	6	34	-	6	34	-	6	34	-
p_{20}	40	12	18	10	11	22	7	13	27	-
p_{21}	40	3	37	-	3	37	-	3	37	-
p_{22}	40	3	37	-	3	37	-	3	37	-
p_{23}	40	-	40	-	-	40	-	-	40	-
p_{24}	40	8	23	9	8	22	10	11	29	-
Sum	960	469	347	144	393	307	260	583	377	-

Table 5.4: Speed Comparison of Programs with Concurrent Sequential Processes

Tools		iProver Modulo/VERDS			iProver Modulo/NuSMV		
Prop	**Num**	**True**	**False**	**Only**	**True**	**False**	**Only**
p_{01}	40	-	-/4	31/-	-/29	-/6	-/5
p_{02}	40	34/1	-	-	40/-	-	-
p_{03}	40	-	12/3	10/-	-	7/18	-/15
p_{04}	40	-	-	12/-	12/-	-	-/28
p_{05}	40	6/12	2/-	9/6	-/21	3/5	-/11
p_{06}	40	8/14	-	9/4	-/36	-	-/4
p_{07}	40	28/3	-	-	39/1	-	-
p_{08}	40	25/3	-	-	40/-	-	-
p_{09}	40	22/2	-/1	10/-	25/8	-/5	-
p_{10}	40	27/7	-	-	34/4	-	-
p_{11}	40	13/7	4/-	12/-	21/9	1/8	-/1
p_{12}	40	19/5	-	5/-	34/6	-	-
p_{13}	40	-	37/-	-	38/2	-	-
p_{14}	40	2/1	27/6	4/-	3/-	12/25	-
p_{15}	40	-	11/3	9/1	-	3/20	-/17
p_{16}	40	-	-	13/-	-/13	-	-/27
p_{17}	40	8/12	1/-	6/6	-/22	1/4	-/13
p_{18}	40	13/14	-	9/4	-/36	-	-/4
p_{19}	40	4/-	17/11	-	6/-	32/2	-
p_{20}	40	5/2	11/3	2/5	12/-	11/7	-/10
p_{21}	40	3/-	27/3	-	3/-	37/-	-
p_{22}	40	2/1	28/5	-	3/-	36/1	-
p_{23}	40	-	27/9	-	-	40/-	-
p_{24}	40	1/3	10/6	3/2	7/1	16/1	-/9
Sum	960	220/87	214/54	144/28	317/188	199/102	-/144

For the total of 1920 testing cases, the data of the experiments shows that 1708 (88.96%) of them are solved by iProver Modulo, 1561 (81.30%) of them are solved by VERDS, while all of them are provable in NuSMV. For the testing cases that are both provable by iProver Modulo and VERDS, 650 of them run faster in iProver Modulo, while 589 of them have advantage in VERDS. For the testing cases that are both provable by iProver Modulo and NuSMV, 858 of them run faster in iProver Modulo, while 759 of them have advantage in NuSMV. All in all, iProver Modulo proves more theorems than VERDS, and in speed is faster. It does not prove more theorems than NuSMV, but when it works, it is often faster, especially in proving the temporal peoperties of the programs with more boolean variables.

Remark The resolution based verification in iProver Modulo and the QBF-based verification implemented in VERDS are both in the way of proving satisfiability, while NuSMV is a BDD-based model checking tool. The comparison here is not meant to draw a conclusion that which method is better, but to emphasize that, *Resolution Modulo* can be considered as a good way of solving model checking problems using an off-the-shelf automated theorem prover.

5.6 Summary

This work is a follow-up work of Chapter 4. In this chapter, the procedure to translate model checking problems into *Polarized Resolution Modulo* is presented. For given Kripke structures, the states can be represented both explicitly by a set of constants and symbolically by a set of boolean variables. In real designs, to build explicit Kripke structures for them and then encode the strutures symbolicly may not feasible because the structure can be too large, even when the final symbolic representation would be concise. Thus in the cases where Kripke structures are not given explicitly, we construct the symbolic representation for the transitions and atomic propositions directly from some concise high-level description of the system. Moreover, a selection function is defined to improve the efficiency of *Polarized Resolution Modulo*, specially to model checking problems. Finally, the experiments on two kinds of programs are implemented in three theorem proving/model checking tools: iProver Modulo, VERDS and NuSMV. The experimental data shows that, *Resolution Modulo* is not a competitor to the usual model checking techniques, but can be considered as a new way to quickly determine whether a temporal property is violated in transition system models.

6 Conclusion and Future Work

Model checking and theorem proving are two kinds of formal verification method which have complementary advantages: model checking is fully automatic, while theorem proving can prove more complex formulas. In this chapter, we will review the contribution of this dissertation and introduce the research directions for the application of *Resolution Modulo*.

6.1 Conclusion

In recent years, a lot of free automated reasoning software are developed, which performs excellent, but far from practical application in real life. In this dissertation, all the works in this dissertation are around the embedding of model checking problems into automated theorem provers.

In Chapter 3, we discussed the way to embed two graph problems, closed-walk detection (the method to find a cycle starting from a given vertex) and block-walk detection (the method to visit all the vertices that are reachable from a given vertex) into theorem provers. Like in graph traversal algorithms, to avoid visiting the same vertex repeatedly, a new elimination rule, called path subsumption rule, is defined. However, as was shown in Chapter 3, the running of path subsumption rule takes much time. Thus, to use the path subsumption rule or not depends on the structure of the graph. The encoding of these two problems laid the groundwork for embedding model checking problems into automated theorem provers.

In Chapter 4, the way to verify temporal formulas on finite state systems in Deduction Modulo was presented. In CTL, the semantics of some temporal formulas are expressed with infinite paths. However, in finite state systems, an alternative semantics of CTL

can be defined, where all the CTL formulas are expressed with some refined finite paths. The soundness and completeness of the alternative semantics shows that, if a Deduction Modulo system preserves the soundness and completeness with respect to the alternative semantics, then this system can be used to do model checking. To reach to this goal, the transition system is represented by a set of rewrite rules (axioms), the alternative semantics are encoded operationally by rewrite rules (using rewrite rules to represent the operational semantics of CTL). Then, the deduction system modulo these rewrite rules can be used to build proofs of the model checking problem.

Chapter 5 is an real implementation of the ideas in Chapter 4. In the *Resolution Modulo* based model checking method, the transition system is represented by a set of (one-way) clauses, which is transformed from the rewrite rules of the system (described in Chapter 4). Likewise, the logical equivalence between the temporal operators are represented as (one-way) clauses. In this chapter, the testing cases are encoded symbolically by a set of boolean variables. The experimental data shows that, *Resolution Modulo* can be considered as a new way to quickly determine whether a temporal property is violated in transition system models.

6.2 Future Work

The work presented in this dissertation provides a theoretical basis of solving finite state model checking problems with automated theorem provers. In real implementations, although the experimental evaluation illustrated a promising result, the efficiency problem still needs to be considered. Besides, whether the model checking problems on pushdown systems can be embedded into *Deduction Modulo* is still under consideration. Finally, we will analyze the feasibility of building an automatic proof system for temporal logic using *Resolution Modulo*.

6.2.1 Model Checking Finite Systems

If a temporal property does not hold on the initial state of the finite model, a derivation steps to get an empty clause can be presented. The trace to reach to the counterexample is contained in the derivations. To write a program of extracting the trace from the whole derivation steps may help the users find out the bugs in their designs quickly.

The data of the experiment evaluation in Chapter 5 shows that the proof-search method does not work efficiently on proving the formulas with nesting fixpoints, for instance $AFAG\phi$. One of the reasons is that during the search steps, a temporal formula for the

same state may be checked several times. For example, in the proof steps for the problem $M, s_1 \models AFAGp$ of Figure 6.1, the proof steps for the subproblem $M, s_4 \models AGp$ are implemented twice.

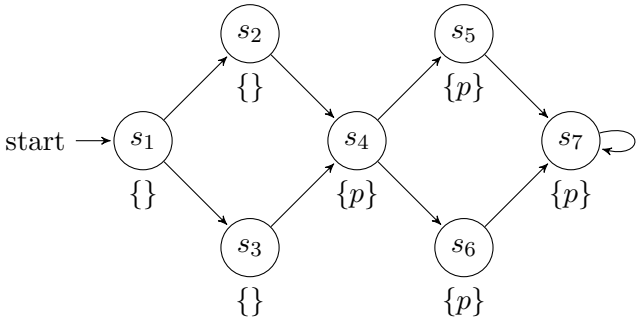


FIGURE 6.1: A Simple Example for the Redundant in Resolution Modulo

One possible way to solve this problem is to design new rewrite rules for the encoding of temporal connectives or new elimination rules similar to the path subsumption rule in Chapter 3.

As we mentioned in the preliminary, model checking problems can also be expressed by some other temporal logic formulas, and the expressive power of different logics are different. For example, in LTL, the formula $A(FG\phi)$ cannot be expressed equivalently by any CTL formula. Likewise, the CTL formula $AF(AG\phi)$ cannot be expressed with any LTL formula either. Besides, there are some well known temporal logics, which are extensions of LTL and CTL. For example, the Extended Computation Tree Logic (eCTL) [EH86] is a propositional branching-time temporal logic that extends the CTL with possibility to express simple fairness constraints. A more powerful temporal logic is CTL*, which allows all possible combinations of modalities. In the future, we will try to solve the model checking problems expressed by these logics with Deduction Modulo systems.

6.2.2 Model Checking Pushdown Systems

A pushdown system is a triplet $\mathcal{P} = (P, \Gamma, \Delta)$ where P is a finite set of control locations, Γ is a finite set of stack alphabet, and $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of transition rules. A configuration of $\mathcal{P}$ is a pair $\langle p, w \rangle$ where $p \in P$ is a control location and $w \in \Gamma^*$ is a stack content. An infinite path is an infinite sequence of configurations $\pi = \pi_0\pi_1...$ such that $\pi_i \rightarrow \pi_{i+1}$ for all $i \geq 0$.

Let AP be a set of atomic propositions. If the atomic propositions are interpreted on configurations and the labeling function is $L : P \rightarrow 2^{AP}$ or $L : P \times \Gamma \rightarrow 2^{AP}$, then the

model checking problems on pushdown systems can be solved by reachability analysis [BEM97]. In [EHRS00], the algorithm of model checking pushdown systems for linear-time temporal logic (LTL) was given, by computing the predecessors and successors of a configuration or a set of configurations. In [DJ15], Gilles Dowek and Ying Jiang showed that the reachability can be inductively defined by the rewrite rules. Moreover, for the temporal properties that are expressed with infinite paths, we have proved the following theorem, which says that an infinite path in the pushdown system can be simulated by a finite path.

Theorem 6.1. *Let* $\pi = \langle p_0, \gamma_0\omega_0\rangle, \langle p_1, \gamma_1\omega_1\rangle, ..., \langle p_n, \gamma_n\omega_n\rangle, ...$ *be an infinite path of the pushdown system* $\mathcal{P}$. *Then* $\exists i \geq 0$ *and* $j > i$ *s.t.* $\langle p_i, \gamma_i\omega_i\rangle, \ldots, \langle p_j, \gamma_j\omega_j\rangle$ *with* $p_i = p_j$, $\gamma_i = \gamma_j$ *and* $\forall i < k \leq j$, $|\omega_k| \geq |\omega_i|$.

Proof. Assume that $\langle p'_0, \gamma'_0\omega'_0\rangle, \langle p'_1, \gamma'_1\omega'_1\rangle, \ldots, \langle p'_n, \gamma'_n\omega'_n\rangle, \ldots$ is the infinite sequence satisfying

- $\langle p'_0, \gamma'_0\omega'_0\rangle = \langle p_m, \gamma_m\omega_m\rangle$ s.t. $\forall n \geq 0$, $|\omega_m| \leq |\omega_n|$
- $\forall i \geq 0$, if $\langle p'_i, \gamma'_i\omega'_i\rangle = \langle p_j, \gamma_j\omega_j\rangle$, then $\langle p'_{i+1}, \gamma'_{i+1}\omega'_{i+1}\rangle = \langle p_k, \gamma_k\omega_k\rangle$ s.t. $k > j$ and $\forall t > j$, $|\omega_k| \leq |\omega_t|$.

As $|P \times \Gamma|$ is finite, we know that $\exists 0 \leq i < j$ such that $\langle p'_i, \gamma'_i\rangle = \langle p'_j, \gamma'_j\rangle$ and $|\omega'_i| \leq |\omega'_j|$. If $\langle p'_i, \gamma'_i\omega'_i\rangle = \langle p_m, \gamma_m\omega_m\rangle$ and $\langle p'_j, \gamma'_j\omega'_j\rangle = \langle p_n, \gamma_n\omega_n\rangle$, then the path $\langle p_m, \gamma_m\omega_m\rangle, ..., \langle p_n, \gamma_n\omega_n\rangle$ is an example of the path required. □

Thus, embedding model checking problems without nesting modalities on pushdown systems into the existing theorem provers is feasible. The combinations of temporal operators with two or more levels of nesting is still under consideration.

6.2.3 Automated Proof of Temporal Logic

The automated proving method of modal logic, including temporal logics are booming in recent years [Fis91, ZHD14, Gor14]. Each time when a proof strategy is designed, they write their own program to implement it. One disadvantage of this way of programming is that, scalability of their programs is very weak. If the inferences rules of these existing methods can be written as rewrite rules, then similiar to our work in this dissertation, the formulas can be proved by the existing first-order or high-order theorem provers.

A

Soundness and Completeness of Theorem 4.11

Lemma A.1 (Soundness)**.** *For a CTL formula φ of NNF, if the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi|, s)$ has a proof, then $M \models \varepsilon_0(|\varphi|, s)$.*

Proof. More generally, we prove that for any CTL formula φ of NNF,

- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi|, s)$ has a proof, then $M \models \varepsilon_0(|\varphi|, s)$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_0}(|\varphi|, [S'])$ has a proof, then $M \models \varepsilon_{\sqcap_0}(|\varphi|, [S'])$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_0}(|\varphi|, [S'])$ has a proof, then $M \models \varepsilon_{\sqcup_0}(|\varphi|, [S'])$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(|\varphi|, s, [s_i^j])$ has a proof, where φ is either of the form $AG\varphi_1$, $EG\varphi_1$, $AR(\varphi_1, \varphi_2)$, $ER(\varphi_1, \varphi_2)$, then $M \models \varepsilon_1(|\varphi|, s, [s_i^j])$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_1}(|\varphi|, [S'], [s_i^j])$ has a proof, where φ is either of the form $AR(\varphi_1, \varphi_2)$, $AG\varphi_1$, then $M \models \varepsilon_{\sqcap_1}(|\varphi|, [S'], [s_i^j])$.
- if $\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_1}(|\varphi|, [S'], [s_i^j])$ has a proof, where φ is either of the form $ER(\varphi_1, \varphi_2)$, $EG\varphi_1$, then $M \models \varepsilon_{\sqcup_1}(|\varphi|, [S'], [s_i^j])$.

By induction on the size of the proof. Consider the different case for φ, we have 18 cases (2 cases for the atomic formula and negation of the atomic formula, 2 cases for the connectors and and or, 10 cases for the modalities ax, ex, af, ef, ag, eg, au, eu, ar, er, 4 cases for the predicate symbols $\varepsilon_{\sqcap_0}$, $\varepsilon_{\sqcup_0}$, $\varepsilon_{\sqcap_1}$, $\varepsilon_{\sqcup_0}$), but each case is easy.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\overline{p}, s)$ has a proof, then the rule $\varepsilon_0(\overline{p}, s) \hookrightarrow \top$ is in $\mathcal{R}_M$, thus $p \in L(s)$ and $M \models \varepsilon_0(\overline{p}, s)$ holds.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{not}(\overline{p}), s)$ has a proof, then the rule $\varepsilon_0(\mathsf{not}(\overline{p}), s) \hookrightarrow \top$ is in $\mathcal{R}_M$, thus $p \notin L(s)$ and $M \models \varepsilon_0(\mathsf{not}(\overline{p}), s)$ holds.

- Suppose that $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{and}(|\varphi_1|, |\varphi_2|), s)$ has a proof. As $\varepsilon_0(\mathsf{and}(|\varphi_1|, |\varphi_2|), s) \hookrightarrow \varepsilon_0(|\varphi_1|, s) \wedge \varepsilon_0(|\varphi_2|, s)$, the last rule of the proof is $\wedge$. By induction hypothesis (IH), $M \models \varepsilon_0(|\varphi_1|, s)$ and $\varepsilon_0(|\varphi_2|, s)$ holds. Thus $M \models \varepsilon_0(\mathsf{and}(|\varphi_1|, |\varphi_2|), s)$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{or}(|\varphi_1|, |\varphi_2|), s)$ has a proof. As $\varepsilon_0(\mathsf{or}(|\varphi_1|, |\varphi_2|), s) \hookrightarrow \varepsilon_0(|\varphi_1|, s) \vee \varepsilon_0(|\varphi_2|, s)$, the last rule of the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models \varepsilon_0(|\varphi_1|, s)$ holds by IH, thus $M \models \varepsilon_0(\mathsf{or}(|\varphi_1|, |\varphi_2|), s)$ holds by its semantic definition. For $\vee_2$, the proof is similar.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ax}(|\varphi|), s)$ has a proof. As $\varepsilon_0(\mathsf{ax}(|\varphi|), s) \hookrightarrow \exists X(r(s, X) \wedge \varepsilon_{\sqcap_0}(|\varphi|, X))$, the last rule of the proof is $\exists$. By IH, there exists S' such that $M \models r(s, [S']) \wedge \varepsilon_{\sqcap_0}(|\varphi|, [S'])$, thus $S' = \mathsf{next}(s)$ and for each state s' in S', $M \models \varepsilon_0(|\varphi|, s')$ holds. Then $M, s \models \varepsilon_0(\mathsf{ax}(|\varphi|), s)$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ex}(|\varphi|), s)$ has a proof. As $\varepsilon_0(\mathsf{ex}(|\varphi|), s) \hookrightarrow \exists X(r(s, X) \wedge \varepsilon_{\sqcup_0}(|\varphi|, X))$, the last rule of the proof is $\exists$. By IH, there exists S' such that $M \models r(s, [S']) \wedge \varepsilon_{\sqcup_0}(|\varphi|, [S'])$, thus $S' = \mathsf{next}(s)$ and there exists a state s' in S' such that $M \models \varepsilon_0(|\varphi|, s')$ holds. Then $M, s \models \varepsilon_0(\mathsf{ex}(|\varphi|), s)$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{af}(|\varphi|), s)$ has a proof. As $\varepsilon_0(\mathsf{af}(|\varphi|), s) \hookrightarrow \varepsilon_0(|\varphi|, s) \vee \exists X(r(s, X) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi|), X))$, the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models \varepsilon_0(|\varphi|, s)$ holds by IH, then $M \models \varepsilon_0(\mathsf{af}(|\varphi|), s)$ holds by its semantic definition. For $\vee_2$, $M \models \exists X(r(s, X) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi|), X))$ holds by IH, thus there exists S' such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi|), [S'])$ holds. Then we get $S' = \mathsf{next}(s)$ and for each state s' in S', $M \models \varepsilon_0(\mathsf{af}(|\varphi|), s')$ holds. Now assume $M \not\models \varepsilon_0(\mathsf{af}(|\varphi|), s)$, then there exists a lsr-path $\rho(s)(j, k)$ such that $\forall 0 \leq i < k$, $M \not\models \varepsilon_0(|\varphi|, \rho_i)$. For the path $\rho(s)(j, k)$,

  - if $j \neq 0$, then ρ^k_1 is a lsr-path, which is a counterexample of $M \models \varepsilon_0(\mathsf{af}(|\varphi|), \rho_1)$.
  - if $j = 0$, then $\rho^k_1 \hat{} \rho_1$ is a lsr-path, which is a counterexample of $M \models \varepsilon_0(\mathsf{af}(|\varphi|), \rho_1)$.

  Thus $M \models \varepsilon_0(\mathsf{af}(|\varphi|), s)$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ef}(|\varphi|), s)$ has a proof. As $\varepsilon_0(\mathsf{ef}(|\varphi|), s) \hookrightarrow \varepsilon_0(|\varphi|, s) \vee \exists X(r(s, X) \wedge \varepsilon_{\sqcup_0}(\mathsf{ef}(|\varphi|), X))$, the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models \varepsilon_0(|\varphi|, s)$ holds by IH, then $M \models \varepsilon_0(\mathsf{ef}(|\varphi|), s)$ holds by its semantic definition. For $\vee_2$, $M \models \exists X(r(s, X) \wedge \varepsilon_{\sqcup_0}(\mathsf{ef}(|\varphi|), X))$ holds by induction hypothesis, thus there exists

S' such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcup_0}(\mathsf{ef}(|\varphi|), [S'])$ holds. Then we get $S' = \mathsf{next}(s)$ and there exists a state s' in S' such that $M \models \varepsilon_0(\mathsf{ef}(|\varphi|), s')$ holds. Thus there exists a lsr-path $\rho'(s')$ and $\exists 0 \leq i < \mathsf{len}(\rho') - 1$ such that $M \models \varepsilon_0(|\varphi|, \rho'_i)$ holds. As there exists a path from s to ρ'_i, by Lemma 4.3, there exists a lsr-path $\rho(s)$, which contains ρ'_i, then $M \models \varepsilon_0(\mathsf{ef}(|\varphi|), s)$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ag}(|\varphi|), s, [s^j_i])$ has a proof. As $\varepsilon_1(\mathsf{ag}(|\varphi|), s, [s^j_i]) \hookrightarrow mem(s, [s^j_i]) \vee (\varepsilon_0(|\varphi|, s) \wedge \exists X(r(s, X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi|), X, \mathsf{con}(s, [s^j_i]))))$, the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models mem(s, [s^j_i])$ holds by IH, thus $s^j_i \hat{} s$ is a lsr-path and $M \models \varepsilon_1(\mathsf{ag}(|\varphi|), s, [s^j_i])$ holds by its semantic definition. For $\vee_2$, $M \models \varepsilon_0(|\varphi|, s)$ and $M \models \exists X(r(s, X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi|), X, \mathsf{con}(s, [s]^j_i)))$ holds by IH. Thus there exists S' such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi|), [S'], \mathsf{con}(s, [s^j_i]))$ holds. Then $S' = \mathsf{next}(s)$ and for each state $s' \in S'$, $M \models \varepsilon_1(\mathsf{ag}(|\varphi|), s', \mathsf{con}(s, [s^j_i]))$ holds. Thus $M \models \varepsilon_1(\mathsf{ag}(|\varphi|), s, [s^j_i])$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{eg}(|\varphi|), s, [s^j_i])$ has a proof. As $\varepsilon_1(\mathsf{eg}(|\varphi|), s, [s^j_i]) \hookrightarrow mem(s, [s^j_i]) \vee (\varepsilon_0(|\varphi|, s) \wedge \exists X(r(s, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi|), X, \mathsf{con}(s, [s^j_i]))))$, the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models mem(s, [s^j_i])$ holds by IH, thus $s^j_i \hat{} s$ is a lsr-path and $M \models \varepsilon_1(\mathsf{eg}(|\varphi|), s, [s^j_i])$ holds by its semantic definition. For $\vee_2$, $M \models \varepsilon_0(|\varphi|, s)$ and $M \models \exists X(r(s, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi|), X, \mathsf{con}(s, [s]^j_i)))$ holds by IH. Thus there exists S' such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi|), [S'], \mathsf{con}(s, [s^j_i]))$ holds. Then $S' = \mathsf{next}(s)$ and there exists $s' \in S'$ such that $M \models \varepsilon_1(\mathsf{eg}(|\varphi|), s', \mathsf{con}(s, [s^j_i]))$ holds. Thus $M \models \varepsilon_1(\mathsf{eg}(|\varphi|), s, [s^j_i])$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s)$ has a proof. As $\varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s) \hookrightarrow \varepsilon_0(|\varphi_2|, s) \vee (\varepsilon_0(|\varphi_1|, s) \wedge \exists X(r(s, X) \wedge \varepsilon_{\sqcap_0}(\mathsf{au}(|\varphi_1|, |\varphi_2|), X)))$, the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models \varepsilon_0(|\varphi_2|, s)$ holds by IH, then $M \models \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s)$ holds by its semantic definition. For $\vee_2$, $M \models \varepsilon_0(|\varphi_1|, s)$ and $M \models \exists X(r(s, X) \wedge \varepsilon_{\sqcap_0}(\mathsf{au}(|\varphi_1|, |\varphi_2|), X))$ holds by IH. Thus there exists S' such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcap_0}(\mathsf{au}(|\varphi_1|, |\varphi_2|), [S']))$ holds. Then we get $S' = \mathsf{next}(s)$ and for each state s' in S', $M \models \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s')$ holds. Now assume $M \not\models \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s)$, then there exists a lsr-path $\rho(s)(j, k)$ such that $\forall 0 \leq i < k$, $M \not\models \varepsilon_0(|\varphi_2|, \rho_i)$ or $\forall 0 \leq i < k$, if $M \models \varepsilon_0(|\varphi_2|, \rho_i)$, then $\exists 0 \leq m < i$, $M \not\models \varepsilon_0(|\varphi_1|, \rho_m)$. For the path $\rho(s)(j, k)$,

  - if $j \neq 0$, then the path ρ^k_1 is a lsr-path, which is a counterexample of $M \models \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), \rho_1)$.
  - if $j = 0$, then the path $\rho^k_1 \hat{} \rho_1$ is a lsr-path, which is a counterexample of $M \models \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), \rho_1)$.

  Thus $M \models \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s)$ holds.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s)$ has a proof. As $\varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s) \hookrightarrow \varepsilon_0(|\varphi_2|, s) \vee (\varepsilon_0(|\varphi_1|, s) \wedge \exists X(r(s, X) \wedge \varepsilon_{\sqcup_0}(\mathsf{eu}(|\varphi_1|, |\varphi_2|), X)))$, the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models \varepsilon_0(|\varphi_2|, s)$ holds by IH, thus $M \models \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s)$ holds by its semantic definition. For $\vee_2$, $M \models \varepsilon_0(|\varphi_1|, s)$ and $M \models \exists X(r(s, X) \wedge \varepsilon_{\sqcup_0}(\mathsf{eu}(|\varphi_1|, |\varphi_2|), X))$ holds by IH. Thus there exists S' such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcup_0}(\mathsf{eu}(|\varphi_1|, |\varphi_2|), [S']))$ holds. Then we get $S' = \mathsf{next}(s)$ and there exists a state s' in S' such that $M \models \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s')$ holds. Thus there exists a lsr-path $\rho'(s')(j, k)$ and $\exists 1 \leq m < k$ such that $M \models \varepsilon_0(|\varphi_2|, \rho'_m)$ holds and $\forall 0 \leq n < m$, $M \models \varepsilon_0(|\varphi_1|, \rho'_n)$ holds. For the path $\rho'(j, k)$,

  - if $\forall 0 \leq i < k$, $\rho'_i \neq s$, then $s \hat{} \rho'(j, k)$ is a lsr-path, in which $M \models \varepsilon_0(|\varphi_2|, \rho'_m)$ holds and $\forall 0 \leq n < m$, $M \models \varepsilon_0(|\varphi_1|, \rho'_n)$ holds,
  - if $\exists m < i < k$ such that $\rho'_i = s$, then $s \hat{} \rho'^{i}_{0}$ is a lsr-path, in which $M \models \varepsilon_0(|\varphi_2|, \rho'_m)$ holds and $\forall 0 \leq n < m$, $M \models \varepsilon_0(|\varphi_1|, \rho'_n)$ holds,
  - if $\exists 0 \leq i < m$ such that $\rho'_i = s$ and $i \leq j$, then ρ'^{k}_{i} is a lsr-path, in which $M \models \varepsilon_0(|\varphi_2|, \rho'_m)$ holds and $\forall i \leq n < m$ $M \models \varepsilon_0(|\varphi_1|, \rho'_n)$ holds,
  - if $\exists 0 \leq i < m$ such that $\rho'_i = s$ and $i > j$, then $\rho'^{k}_{i} \hat{} \rho'^{i}_{j+1}$ is a lsr-path, in which $M \models \varepsilon_0(|\varphi_2|, \rho'_m)$ holds and $\forall i \leq n < m$, $M \models \varepsilon_0(|\varphi_1|, \rho'_n)$ holds.

  Thus $M \models \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s)$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, [s^j_i])$ has a proof. For $\varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, [s^j_i])$, only the rewrite rule $\varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, [s^j_i]) \hookrightarrow mem(s, [s^j_i]) \vee (\varepsilon_0(|\varphi_2|, s) \wedge (\varepsilon_0(|\varphi_1|, s) \vee \exists X(r(s, X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), X, \mathsf{con}(s, [s^j_i])))))$ can be used, thus the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models mem(s, [s^j_i])$ holds by IH, thus $s^j_i \hat{} s$ is a lsr-path and $M \models \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, [s^j_i])$ holds by its semantic definition. For $\vee_2$, $M \models \varepsilon_0(|\varphi_2|, s)$ and $M \models \varepsilon_0(|\varphi_1|, s) \vee \exists X(r(s, X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), X, \mathsf{con}(s, [s^j_i])))$ holds by IH. If $M \models \varepsilon_0(|\varphi_1|, s)$ holds, then from the semantics of $M \models \varepsilon_0(|\varphi_2|, s)$ and $M \models \varepsilon_0(|\varphi_1|, s)$, we get $M \models \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, [s^j_i])$ holds by its semantic definition. If there exists a set S' of states, such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), [S'], \mathsf{con}(s, [s^j_i]))$ holds, then $S' = \mathsf{next}(s)$ and $\forall s' \in S'$, $M \models \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s', \mathsf{con}(s, [s^j_i]))$ holds. Thus $M \models \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, [s^j_i])$ holds by its semantic definition.

- Suppose $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, [s^j_i])$ has a proof. For $\varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, [s^j_i])$, only the rewrite rule $\varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, [s^j_i]) \hookrightarrow mem(s, [s^j_i]) \vee (\varepsilon_0(|\varphi_2|, s) \wedge (\varepsilon_0(|\varphi_1|, s) \vee \exists X(r(s, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), X, \mathsf{con}(s, [s^j_i])))))$ can be used, thus the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models mem(s, [s^j_i])$ holds by IH, thus $s^j_i \hat{} s$ is a lsr-path and $M \models \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, [s^j_i])$ holds by its semantic definition. For $\vee_2$, $M \models$

$\varepsilon_0(|\varphi_2|, s)$ and $M \models \varepsilon_0(|\varphi_1|, s) \vee \exists X(r(s, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), X, \mathsf{con}(s, [s_i^j])))$ holds by IH. If $M \models \varepsilon_0(|\varphi_1|, s)$ holds, then from the semantics of $M \models \varepsilon_0(|\varphi_2|, s)$ and $M \models \varepsilon_0(|\varphi_1|, s)$, we get $M \models \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, [s_i^j])$ holds by its semantic definition. If there exists a set S' of states, such that $M \models r(s, [S'])$ and $M \models \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), [S'], \mathsf{con}(s, [s_i^j]))$ holds, then $S' = \mathsf{next}(s)$ and there exists a state s' in S' such that $M \models \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s', \mathsf{con}(s, [s_i^j]))$ holds. Thus, by the definition of semantics, $M \models \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, [s_i^j])$ holds.

- Suppose that $\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcap_0}(|\varphi|, \mathsf{con}(s, [S']))$ has a proof. As $\varepsilon_{\sqcap_0}(|\varphi|, \mathsf{con}(s, [S'])) \hookrightarrow \varepsilon_0(|\varphi|, s) \wedge \varepsilon_{\sqcap_0}(|\varphi|, [S'])$, the last rule in the proof is $\wedge$. Thus $M \models \varepsilon_0(|\varphi|, s)$ and $M \models \varepsilon_{\sqcap_0}(|\varphi|, [S'])$ holds by IH. Then $M \models \varepsilon_{\sqcap_0}(|\varphi|, \mathsf{con}(s, [S']))$ holds by its semantic definition.

- Suppose that $\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcup_0}(|\varphi|, \mathsf{con}(s, [S']))$ has a proof. As $\varepsilon_{\sqcup_0}(|\varphi|, \mathsf{con}(s, [S'])) \hookrightarrow \varepsilon_0(|\varphi|, s) \vee \varepsilon_{\sqcup_0}(|\varphi|, [S'])$, the last rule in the proof is $\vee_1$ or $\vee_2$. For $\vee_1$, $M \models \varepsilon_0(|\varphi|, s)$ holds by IH, then $M \models \varepsilon_{\sqcup_0}(|\varphi|, \mathsf{con}(s, [S']))$ holds by its semantic definition. For $\vee_2$, $M \models \varepsilon_{\sqcup_0}(|\varphi|, [S'])$ holds by IH, then we exists a state $s' \in S'$ such that $M \models \varepsilon_0(|\varphi|, s')$ holds, thus $M \models \varepsilon_{\sqcup_0}(|\varphi|, \mathsf{con}(s, [S']))$ holds by its semantic definition.

- The proof of $\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcap_1}(|\varphi|, \mathsf{con}(s, [S']), [s_i^j])$ and $\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcup_1}(|\varphi|, \mathsf{con}(s, [S']), [s_i^j])$, are similar with $\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcap_0}(|\varphi|, \mathsf{con}(s, [S']))$ and $\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcup_0}(|\varphi|, \mathsf{con}(s, [S']))$.

□

Lemma A.2 (Completeness)**.** *For a CTL formula φ of NNF, if $M \models \varepsilon_0(|\varphi|, s)$, then the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi|, s)$ has a proof.*

Proof. By induction on the structure of φ.

- Suppose $M \models \varepsilon_0(\overline{p}, s)$ holds, in which $p \in AP$. By the semantics of $\mathcal{L}$, $p \in \mathsf{L}(s)$. Thus the rule $\varepsilon_0(\overline{p}, s) \hookrightarrow \top$ is in $\mathcal{R}_M$ and the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\overline{p}, s)$ is provable by the $\top$ rule.

- Suppose $M \models \varepsilon_0(\mathsf{not}(\overline{p}), s)$ holds, in which $p \in AP$. By the semantics of $\mathcal{L}$, $p \notin \mathsf{L}(s)$. Thus the rule $\varepsilon_0(\mathsf{not}(\overline{p}), s) \hookrightarrow \top$ is in $\mathcal{R}_M$ and the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{not}(\overline{p}), s)$ is provable by the $\top$ rule.

- Suppose $M \models \varepsilon_0(\mathsf{or}(|\varphi_1|, |\varphi_2|), s)$ holds. By the semantics of $\mathcal{L}$, $M \models \varepsilon_0(|\varphi_1|, s)$ or $M \models \varepsilon_0(|\varphi_2|, s)$ holds. Without loss of generality, assume that $M \models \varepsilon_0(|\varphi_1|, s)$ holds, then by induction hypothesis, there exists a proof $\Pi_{(\varphi_1, s)}$ for the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi_1|, s)$. The proof of the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{or}(|\varphi_1|, |\varphi_2|), s)$ is as follows:

$$\frac{\Pi_{(\varphi_1,s)}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{or}(|\varphi_1|,|\varphi_2|),s)} \vee_1$$

- Suppose $M \models \varepsilon_0(\mathsf{and}(|\varphi_1|,|\varphi_2|),s)$ holds. By the semantics of $\mathcal{L}$, $M \models \varepsilon_0(|\varphi_1|,s)$ and $M \models \varepsilon_0(|\varphi_2|,s)$ holds. By induction hypothesis, there exists a proof $\Pi_{(\varphi_i,s)}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_i|,s)(i = 1,2)$. Then the proof of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{and}(|\varphi_1|,|\varphi_2|),s)$ is as follows:

$$\frac{\Pi_{(\varphi_1,s)} \qquad \Pi_{(\varphi_2,s)}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{and}(|\varphi_1|,|\varphi_2|),s)} \wedge$$

- Suppose $M \models \varepsilon_0(\mathsf{ax}(|\varphi_1|),s)$ holds. Assume that $\mathsf{next}(s) = \{s_0,\ldots,s_k\}$, by the semantics of $\mathcal{L}$, $\forall 0 \leq i \leq k, M \models \varepsilon_0(|\varphi_1|,s_i)$ holds. Then by induction hypothesis, there exists a proof $\Pi_{(\varphi_1,s_i)}$ for each sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|,s_i)(0 \leq i \leq k)$. The proof of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ax}(|\varphi_1|),s)$ is as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s,[\mathsf{next}(s)])}\top \quad \dfrac{\Pi_{(\varphi_1,s_0)} \quad \ldots \quad \Pi_{(\varphi_1,s_k)}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_0}(|\varphi_1|,[\mathsf{next}(s)])}\wedge^k}{\vdash^{cf}_{\mathcal{R}} r(s,[\mathsf{next}(s)]) \wedge \varepsilon_{\sqcap_0}(|\varphi_1|,[\mathsf{next}(s)])}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ax}(|\varphi_1|),s)}\exists$$

- Suppose $M \models \varepsilon_0(\mathsf{ex}(|\varphi_1|),s)$ holds. Assume that $\mathsf{next}(s) = \{s_0,\ldots,s_k\}$, by the semantics of $\mathcal{L}$, $\exists 0 \leq i \leq k$ s.t. $M \models \varepsilon_0(|\varphi_1|,s_i)$. Then by induction hypothesis, there exists a proof $\Pi_{(\varphi_1,s_i)}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|,s_i)$. The proof of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ex}(|\varphi_1|),s)$ is as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s,[\mathsf{next}(s)])}\top \quad \dfrac{\dfrac{\Pi_{(\varphi_1,s_i)}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_0}(|\varphi_1|,\mathsf{con}(s_i,[S']))}\vee_1}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_0}(|\varphi_1|,[\mathsf{next}(s)])}\vee^i_2}{\vdash^{cf}_{\mathcal{R}} r(s,[\mathsf{next}(s)]) \wedge \varepsilon_{\sqcup_0}(|\varphi_1|,[\mathsf{next}(s)])}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ex}(|\varphi_1|),s)}\exists$$

- Suppose $M \models \varepsilon_0(\mathsf{af}(|\varphi_1|),s)$ holds. By the semantics of $\mathcal{L}$, there exists a state s' on each lsr-path starting from s s.t. $M \models \varepsilon_0(|\varphi_1|,s')$ holds. Thus there exists a finite tree T s.t.
  - T has root s;
  - for each internal node s' in T; the children of s' are labelled by the elements of $\mathsf{next}(s')$;
  - for each leaf s', s' is the first node in the branch starting from s s.t. $M \models \varepsilon_0(|\varphi_1|,s')$ holds.

By induction hypothesis, for each leaf s', there exists a proof $\Pi_{(\varphi_1,s')}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s')$. Then, to each subtree T' of T, we associate a proof $|T'|$ of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{af}(|\varphi_1|), s')$ where s' is the root of T', by induction, as follows,

- if T' contains a single node s', then the proof $|T'|$ is as follows:

$$\dfrac{\Pi_{(\varphi_1,s')}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{af}(|\varphi_1|), s')} \vee_1$$

- if $T' = s'(T_0, \ldots, T_n)$[1], then the proof $|T'|$ is as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')])} \top \quad \dfrac{|T_0| \quad \ldots \quad |T_n|}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi_1|), [\mathsf{next}(s')])} \wedge^n}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')]) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi_1|), [\mathsf{next}(s')])} \wedge}{\vdash^{cf}_{\mathcal{R}} \exists X(r(s', X) \wedge \varepsilon_{\sqcap_0}(\mathsf{af}(|\varphi_1|), X))} \exists}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{af}(|\varphi_1|), s')} \vee_2$$

This way, $|T|$ is a proof of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{af}(|\varphi_1|), s)$.

- Suppose $M \models \varepsilon_0(\mathsf{ag}(|\varphi_1|), s)$ holds. By the semantics of $\mathcal{L}$, for each state s' on each lsr-path starting from s, $M \models \varepsilon_0(|\varphi_1|, s')$ holds. Thus there exists a finite tree T s.t.
  - T has root s;
  - for each internal node s' in T, the children of s' are labelled by the elements of $\mathsf{next}(s')$;
  - the branch starting from s to each leaf is a lsr-path;
  - for each internal node s' in T, $M \models \varepsilon_0(|\varphi_1|, s')$ holds and by induction hypothesis, there exists a proof $\Pi_{(\varphi_1,s')}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s')$.

  Then, to each subtree T' of T, we associate a proof $|T'|$ of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ag}(|\varphi_1|), s', [s'^{k-1}_0])$ where s' is the root of T' and $s'^{k}_0(s'_k = s')$ is the branch from s to s', by induction, as follows,

  - if T' contains a single node s', then s'^{k}_0 is a lsr-path and the proof is as follows:

$$\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} mem(s', [s'^{k-1}_0])} \top}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ag}(|\varphi_1|), s', [s'^{k-1}_0])} \vee_2$$

  - if $T' = s'(T_0, \ldots, T_n)$, the proof is as follows:

[1] $s'(T_0, \ldots, T_n)$ is a tree, in which s' is the root, $T_0, \ldots, T_n$ are the sub-trees.

$$\dfrac{\dfrac{\Pi_{s'}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s')} \quad \dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')])}\top \quad \dfrac{|T_0| \quad \ldots \quad |T_n|}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), [\mathsf{next}(s')], [s_0'^k])}\wedge^n}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')]) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), [\mathsf{next}(s')], [s_0'^k])}\wedge}{\vdash^{cf}_{\mathcal{R}} \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), X, [s_0'^k]))}\exists}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s') \wedge \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ag}(|\varphi_1|), X, [s_0'^k]))}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ag}(|\varphi_1|), s', [s_0'^{k-1}])}\vee_1$$

This way, as $\varepsilon_0(\mathsf{ag}(|\varphi_1|), s)$ can be rewritten into $\varepsilon_1(\mathsf{ag}(|\varphi_1|), s, \mathsf{nil})$, $|T|$ is a proof for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ag}(|\varphi_1|), s)$.

- Suppose $M \models \varepsilon_0(\mathsf{ef}(|\varphi_1|), s)$ holds. By the semantics of $\mathcal{L}$, there exists a lsr-path s_0^k starting from s and $\exists 0 \leq j < k$ s.t. $M \models \varepsilon_0(\mathsf{ef}(|\varphi_1|), s_j)$ and by induction hypothesis, there exists a proof $\Pi_{(\varphi_1, s_j)}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s_j)$. To each subpath s_i^j of s_0^j, we associate a proof $|s_i^j|$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ef}(|\varphi_1|), s_i)$, by induction, as follows,

  - if s_i^j contains a single node s_j, then the proof $|s_i^j|$ is as follows:

$$\dfrac{\Pi_{(\varphi_1, s_j)}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ef}(|\varphi_1|), s_j)}\vee_1$$

  - Otherwise, assume $\mathsf{next}(s_i) = \{s'_0, \ldots, s'_n\}$ and $s_{i+1} = s'_m$, the proof $|s_i^j|$ is as follows:

$$\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s_i, [\mathsf{next}(s_i)])}\top \quad \dfrac{\dfrac{|s_{i+1}^j|}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_0}(\mathsf{ef}(|\varphi_1|), \mathsf{con}(s'_m, [S']))}\vee_1}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_0}(\mathsf{ef}(|\varphi_1|), [\mathsf{next}(s_i)])}\vee_2^m}{\vdash^{cf}_{\mathcal{R}} r(s_i, [\mathsf{next}(s_i)]) \wedge \varepsilon_{\sqcup_0}(\mathsf{ef}(|\varphi_1|), [\mathsf{next}(s_i)])}\wedge}{\vdash^{cf}_{\mathcal{R}} \exists X(r(s_i, X) \wedge \varepsilon_{\sqcup_0}(\mathsf{ef}(|\varphi_1|), X))}\exists}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ef}(|\varphi_1|), s_i)}\vee_2$$

  This way, $|s_0^j|$ is a proof of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ef}(|\varphi_1|), s)$.

- Suppose $M \models \varepsilon_0(\mathsf{eg}(|\varphi_1|), s)$ holds. By the semantics of $\mathcal{L}$, there exists a lsr-path s_0^k starting from s s.t. $\forall 0 \leq i < k$, $M \models \varepsilon_0(|\varphi_1|, s_i)$ holds and by induction hypothesis, there exists a proof $\Pi_{(\varphi_1, s_i)}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s_i)$. Then to each subpath s_j^k of s_0^k, we associate a proof $|s_j^k|$ of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{eg}(|\varphi_1|), s_j, [s_0^{j-1}])$, by induction, as follows,

  - if s_j^k contains a single node s_k, then s_1^j is a lsr-path. The proof is as follows:

$$\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} mem(s_j, [s_0^{j-1}])}\top}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{eg}(|\varphi_1|), s_j, [s_0^{j-1}])}\vee_2$$

– Otherwise, assume $\mathsf{next}(s_j) = \{s'_0, \ldots, s'_n\}$ and $s_{j+1} = s'_m$, the proof $|s^k_j|$ is as follows:

$$\dfrac{\dfrac{\Pi_{(\varphi_1,s_j)}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s_j)} \quad \dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s_j, [\mathsf{next}(s_j)])}\top \quad \dfrac{\dfrac{|s^k_{j+1}|}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi_1|), \mathsf{con}(s'_m, [S']), [s^j_0])}\vee_1}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi_1|), [\mathsf{next}(s_j)], [s^j_0])}\vee^m_2}{\vdash^{cf}_{\mathcal{R}} r(s_j, [\mathsf{next}(s_j)]) \wedge \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi_1|), [\mathsf{next}(s_j)], [s^j_0])}\wedge}{\vdash^{cf}_{\mathcal{R}} \exists X(r(s_j, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi_1|), X, [s^j_0]))}\exists}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s_j) \wedge \exists X(r(s_j, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{eg}(|\varphi_1|), X, [s^j_0]))}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{eg}(|\varphi_1|), s_j, [s^{j-1}_0])}\vee_1$$

This way, as $\varepsilon_0(\mathsf{eg}(|\varphi_1|), s)$ can be rewritten into $\varepsilon_1(\mathsf{eg}(|\varphi_1|), s, \mathsf{nil})$, $|s^k_0|$ is a proof for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{eg}(|\varphi_1|), s)$.

- Suppose $M \models \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s)$ holds. By the semantics of $\mathcal{L}$, for each lsr-path s^k_0 starting from s, $\exists 0 \leq i < k$, s.t. $M \models \varepsilon_0(|\varphi_2|, s_i)$ holds and $\forall 0 \leq j < i$, $M \models \varepsilon_0(|\varphi_1|, s_j)$. Thus there exists a finite tree T s.t.

  – T has root s;

  – for each internal node s' in T, the children of s' are labelled by the elements of $\mathsf{next}(s')$;

  – for each internal node s' in T, $M \models \varepsilon_0(|\varphi_1|, s')$ holds and by induction hypothesis, there exists a proof $\Pi_{(\varphi_1,s')}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s')$;

  – for each leaf s' is the first node in the branch starting from s s.t. $M \models \varepsilon_0(|\varphi_2|, s')$ holds and by induction hypothesis, there exists a proof $\Pi_{(\varphi_2,s')}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_2|, s')$.

  Then, to each subtree T' of T, we associate a proof $|T'|$ of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s')$ where s' is the root of T', by induction, as follows,

  – if T' contains a single node s', then s' is a leaf and the proof is as follows:

$$\dfrac{\Pi_{(\varphi_2,s')}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s')}\vee_1$$

  – if $T' = s'(T_0, \ldots, T_n)$, then the proof $|T'|$ is as follows:

$$\dfrac{\Pi_{(\varphi_1,s')} \quad \dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')])}\top \quad \dfrac{|T_0| \quad \ldots \quad |T_n|}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_0}(\mathsf{au}(|\varphi_1|, |\varphi_2|), [\mathsf{next}(s')])}\wedge^n}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')]) \wedge \varepsilon_{\sqcap_0}(\mathsf{au}(|\varphi_1|, |\varphi_2|), [\mathsf{next}(s')])}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s') \wedge \exists X(r(s', X) \wedge \varepsilon_{\sqcap_0}(\mathsf{au}(|\varphi_1|, |\varphi_2|), X))}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s')}\vee_2$$

This way, $|T|$ is a proof of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{au}(|\varphi_1|, |\varphi_2|), s)$.

- Suppose $M \models \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s)$ holds. By the semantics of $\mathcal{L}$, there exists a lsr-path s_0^k starting from s and $\exists 0 \leq j < k$, s.t. $M \models \varepsilon_0(|\varphi_2|, s_j)$ and $\forall 0 \leq i < j$, $M \models \varepsilon_0(|\varphi_1|, s_i)$. By induction hypothesis, for each state s', if $M \models \varepsilon_0(|\varphi_1|, s')$, then there exists a proof $\Pi_{(\varphi_1, s')}$ for the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi_1|, s')$ and if $M \models \varepsilon_0(|\varphi_2|, s')$, then there exists a proof $\Pi_{(\varphi_2, s')}$ for the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi_2|, s')$. To each subpath s_i^j of s_0^j, we associate a proof $|s_i^j|$ for the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s)$, by induction, as follows,

  - if s_i^j contains a single node s_j, then the proof is as follows:

$$\dfrac{\Pi_{(\varphi_2, s_j)}}{\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s_j)} \vee_1$$

  - Otherwise, assume $\mathsf{next}(s_i) = \{s'_0, \ldots, s'_n\}$ and $s_{i+1} = s'_m$, the proof $|s_i^j|$ is as follows:

$$\dfrac{\Pi_{(\varphi_1, s_i)} \qquad \dfrac{\dfrac{\dfrac{}{\vdash_{\mathcal{R}}^{cf} r(s_i, [\mathsf{next}(s_i)])} \top \qquad \dfrac{\dfrac{|s_{i+1}^j|}{\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcup_0}(\mathsf{eu}(|\varphi_1|, |\varphi_2|), \mathsf{con}(s'_m, [S']))} \vee_1}{\vdash_{\mathcal{R}}^{cf} \varepsilon_{\sqcup_0}(\mathsf{eu}(|\varphi_1|, |\varphi_2|), [\mathsf{next}(s_i)])} \vee_2^m}{\vdash_{\mathcal{R}}^{cf} r(s_i, [\mathsf{next}(s_i)]) \wedge \varepsilon_{\sqcup_0}(\mathsf{eu}(|\varphi_1|, |\varphi_2|), [\mathsf{next}(s_i)])} \wedge}{\vdash_{\mathcal{R}}^{cf} \exists X(r(s_i, X) \wedge \varepsilon_{\sqcup_0}(\mathsf{eu}(|\varphi_1|, |\varphi_2|), X))} \exists}{\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi_1|, s_i) \wedge \exists X(r(s_i, X) \wedge \varepsilon_{\sqcup_0}(\mathsf{eu}(|\varphi_1|, |\varphi_2|), X))} \wedge}{\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s_i)} \vee_2$$

  This way, $|s_0^j|$ is a proof of the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(\mathsf{eu}(|\varphi_1|, |\varphi_2|), s)$.

- Suppose $M \models \varepsilon_0(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s)$ holds. By the semantics of $\mathcal{L}$, for each lsr-path s_0^k starting from s, and $\forall 0 \leq j < k$, either $M \models \varepsilon_0(|\varphi_2|, s_j)$ holds or $\exists 0 \leq i < j$ s.t. $M \models \varepsilon_0(|\varphi_1|, s_i)$ holds. Thus there exists a finite tree T s.t.

  - T has root s;
  - for each internal node s' in T, the children of s' are labelled by the elements of $\mathsf{next}(s')$;
  - for each internal node s' in T, $M \models \varepsilon_0(|\varphi_2|, s')$ holds and by induction hypothesis, there exists a proof $\Pi_{(\varphi_2, s')}$ for the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi_2|, s')$.
  - for each leaf s', either the branch from s to s' is a lsr-path or $M \models \varepsilon_0(|\varphi_1|, s')$ holds and by induction hypothesis, there exists a proof $\Pi_{(\varphi_1, s')}$ for the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_0(|\varphi_1|, s')$.

  Then, to each subtree T' of T, we associate a proof $|T'|$ of the sequent $\vdash_{\mathcal{R}}^{cf} \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s', [s_0'^k])$ where s' is the root of T' and $s_0'^k (s' = s'_k)$ is the branch from s to s', by induction, as follows,

– if T' contains a single node s' and $s_0'^k$ is a lsr-path, the proof is as follows:

$$\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} mem(s', [s_0'^{k-1}])}\top}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s', [s_0'^{k-1}])}\vee_1$$

– if T' contains a single node s' and $M \models \varepsilon_0(|\varphi_1|, s')$ holds, the proof is as follows:

$$\dfrac{\Pi_{(\varphi_2,s')} \quad \dfrac{\Pi_{(\varphi_1,s')}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s') \vee \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), X, [s_0'^k]))}\vee_1}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_2|, s') \wedge (\varepsilon_0(|\varphi_1|, s') \vee \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), X, [s_0'^k])))}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s', [s_0'^{k-1}])}\vee_2$$

– if $T' = s'(T_0, \ldots, T_n)$, the proof is built as follows:

$$\dfrac{\Pi_{(\varphi_2,s')} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')])}\top \quad \dfrac{|T_0| \quad \ldots \quad |T_n|}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), [\mathsf{next}(s')], [s_0'^k])}\wedge^n}{\vdash^{cf}_{\mathcal{R}} r(s', [\mathsf{next}(s')]) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), [\mathsf{next}(s')], [s_0'^k])}\wedge}{\vdash^{cf}_{\mathcal{R}} \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), X, [s_0'^k]))}\exists}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s') \vee \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), X, [s_0'^k]))}\vee_2}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_2|, s') \wedge (\varepsilon_0(|\varphi_1|, s') \vee \exists X(r(s', X) \wedge \varepsilon_{\sqcap_1}(\mathsf{ar}(|\varphi_1|, |\varphi_2|), X, [s_0'^k])))}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s', [s_0'^{k-1}])}\vee_2$$

This way, as $\varepsilon_0(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s)$ can be rewritten into $\varepsilon_1(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s, \mathsf{nil})$, $|T|$ is a proof for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{ar}(|\varphi_1|, |\varphi_2|), s)$.

- Suppose $M \models \varepsilon_0(\mathsf{er}(|\varphi_1|, |\varphi_2|), s)$ holds. By the semantics of $\mathcal{L}$, there exists a lsr-path s_0^k starting from s and $\forall 0 \leq j < k$, either $M \models \varepsilon_0(|\varphi_2|, s_j)$ or $\exists 0 \leq i < j$ s.t. $M \models \varepsilon_0(|\varphi_1|, s_i)$. By IH, for each state s', if $M \models \varepsilon_0(|\varphi_1|, s')$, then there exists a proof $\Pi_{(\varphi_1,s')}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s')$ and if $M \models \varepsilon_0(|\varphi_2|, s')$, then there exists a proof $\Pi_{(\varphi_2,s')}$ for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_2|, s')$. Then to each subpath s_j^k of s_0^k, we associate a proof $|s_j^k|$ of the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s_j, [s_0^{j-1}])$, by induction, as follows,

  – if s_j^k contains a single node s_k, then s_0^j is a lsr-path and the proof is as follows:

$$\dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} mem(s_j, [s_0^{j-1}])}\top}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s_j, [s_0^{j-1}])}\vee_2$$

  – if $M \models \varepsilon_0(|\varphi_1|, s_j)$ holds, the proof is as follows:

$$\dfrac{\Pi_{(\varphi_2,s_j)} \quad \dfrac{\Pi_{(\varphi_1,s_j)}}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s_j) \vee \exists X(r(s_j, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), X, [s_0^j]))}\vee_1}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_2|, s_j) \wedge (\varepsilon_0(|\varphi_1|, s_j) \vee \exists X(r(s_j, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), X, [s_0^j])))}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s_j, [s_0^{j-1}])}\vee_1$$

– Otherwise, assume $\mathsf{next}(s_j) = \{s'_0, \ldots, s'_n\}$ and $s_{j+1} = s'_m$, the proof is as follows:

$$\dfrac{\Pi_{(\varphi_2, s_j)} \qquad \dfrac{\dfrac{}{\vdash^{cf}_{\mathcal{R}} r(s_j, [\mathsf{next}(s_j)])}\top \qquad \dfrac{\dfrac{|s^k_{j+1}|}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), \mathsf{con}(s'_m, [S']), [s^j_0])}\vee_1}{\vdash^{cf}_{\mathcal{R}} \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), [\mathsf{next}(s_j)], [s^j_0])}\vee_2^m}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_1|, s_j) \vee \exists X(r(s_j, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), X, [s^j_0]))}\exists}{\vdash^{cf}_{\mathcal{R}} \varepsilon_0(|\varphi_2|, s_j) \wedge (\varepsilon_0(|\varphi_1|, s_j) \vee \exists X(r(s_j, X) \wedge \varepsilon_{\sqcup_1}(\mathsf{er}(|\varphi_1|, |\varphi_2|), X, [s^j_0])))}\wedge}{\vdash^{cf}_{\mathcal{R}} \varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s_j, [s^{j-1}_0])}\vee_1$$

This way, as $\varepsilon_0(\mathsf{er}(|\varphi_1|, |\varphi_2|), s)$ can be rewritten into $\varepsilon_1(\mathsf{er}(|\varphi_1|, |\varphi_2|), s, \mathsf{nil})$, $|s^k_0|$ is a proof for the sequent $\vdash^{cf}_{\mathcal{R}} \varepsilon_0(\mathsf{er}(|\varphi_1|, |\varphi_2|), s)$.

□

B

wgc.p

The content of the input file for the wolf-goat-cabbage problem wgc.p is as follows.

```
% One-way clauses for the class variables
cnf(mem_1, axiom, mem(X1,con(X1,Z))).
cnf(mem_2, axiom, mem(X1,con(Y1,Z))| ~mem(X1,Z)).

% One-way clauses for the CTL connectives
cnf(not,axiom,~pi0(not(P),X1)|~pi0(P,X1)).
cnf(or, axiom,~pi0(or(P,Q),X1)|pi0(P,X1)|pi0(Q,X1)).
cnf(ad1,axiom,~pi0(and(P,Q),X1)|pi0(Q,X1)).
cnf(ad2,axiom,~pi0(and(P,Q),X1)|pi0(P,X1)).
cnf(ax,axiom,~pi0(ax(P),X1)|~r(X1,con(Z1,Z))|pin0(P,con(Z1,Z))).
cnf(ex,axiom,~pi0(ex(P),X1)|~r(X1,con(Z1,Z))|piu0(P,con(Z1,Z))).
cnf(eg1,axiom,~pi0(eg(P),X1)|~r(X1,con(Z1,Z))|piu0(eg(P),con(Z1,Z))).
cnf(eg2,axiom,~pi0(eg(P),X1)|pi0(P,X1)).
cnf(ag1,axiom,~pi0(ag(P),X1)|~r(X1,con(Z1,Z))|piu0(ag(P),con(Z1,Z))).
cnf(ag2,axiom,~pi0(ag(P),X1)|pi0(P,X1)).
cnf(af1,axiom,~pi0(af(P),X1)|pi1(af(P),X1,nil)).
cnf(af2,axiom,~pi1(af(P),X1,Y)|pi0(P,X1)|~r(X1,con(Z1,Z))
               |pin1(af(P),con(Z1,Z),con(X1,Y))).
cnf(af3,axiom,~pi1(af(P),X1,Y)|~mem(X1,Y)).
cnf(ef1,axiom,~pi0(ef(P),X1)|pi1(ef(P),X1,nil)).
cnf(ef2,axiom,~pi1(ef(P),X1,Y)|pi0(P,X1)|~r(X1,con(Z1,Z))
               |piu1(ef(P),con(Z1,Z),con(X1,Y))).
cnf(ef3,axiom,~pi1(ef(P),X1,Y)|~mem(X1,Y)).
cnf(au1,axiom,~pi0(au(P,Q),X1)|pi1(au(P,Q),X1,nil)).
```

```
cnf(au2,axiom,~pi1(au(P,Q),X1,Y)| pi0(Q,X1)| ~r(X1,con(Z1,Z))
               |pin1(au(P,Q),con(Z1,Z),con(X1,Y))).
cnf(au2,axiom,~pi1(au(P,Q),X1,Y)| pi0(Q,X1)| pi0(P,X1)).
cnf(au3,axiom,~pi1(au(P,Q),X1,Y)| ~mem(X1,Y)).
cnf(eu1,axiom,~pi0(eu(P,Q),X1)| pi1(eu(P,Q),X1,nil)).
cnf(eu2,axiom,~pi1(eu(P,Q),X1,Y)| pi0(Q,X1)| ~r(X1,con(Z1,Z))
               |piu1(eu(P,Q),con(Z1,Z),con(X1,Y))).
cnf(eu2,axiom,~pi1(eu(P,Q),X1,Y)|pi0(Q,X1)| pi0(P,X1)).
cnf(eu3,axiom,~pi1(eu(P,Q),X1,Y)|~mem(X1,Y)).
cnf(er1,axiom,~pi0(er(P,Q),X1)|pi0(P,X1)|~r(X1,con(Z1,Z))
               |piu0(er(P,Q),con(Z1,Z))).
cnf(er2,axiom,~pi0(er(P,Q),X1)|pi0(Q,X1)).
cnf(ar1,axiom,~pi0(ar(P,Q),X1)|pi0(P,X1)|~r(X1,con(Z1,Z))
               |pin0(ar(P,Q),con(Z1,Z))).
cnf(ar2,axiom,~pi0(ar(P,Q),X1)|pi0(Q,X1)).
cnf(land1,axiom,~pin0(P,con(X1,Z))|pin0(P,Z)).
cnf(land2,axiom,~pin0(P,con(X1,Z))|pi0(P,X1)).
cnf(land3,axiom,~pin1(P,con(X1,Z),Y)|pin1(P,Z,Y)).
cnf(land4,axiom,~pin1(P,con(X1,Z),Y)|pi1(P,X1,Y)).
cnf(lor1,axiom,~piu0(P,con(X1,Z))|pi0(P,X1)|piu0(P,Z)).
cnf(lor2,axiom,~piu0(P,nil)).
cnf(lor3,axiom,~piu1(P,con(X1,Z),Y)|pi1(P,X1,Y)|piu1(P,Z,Y)).
cnf(lor4,axiom,~piu1(P,nil,Y)).

% Kripke structure
cnf(prop1, axiom,  pi0(pm,s(b(tt,ff),Bw,Bg,Bc))).
cnf(prop2, axiom, ~pi0(pm,s(b(ff,tt),Bw,Bg,Bc))).
cnf(prop3, axiom,  pi0(pw,s(Bm,b(tt,ff),Bg,Bc))).
cnf(prop4, axiom, ~pi0(pw,s(Bm,b(ff,tt),Bg,Bc))).
cnf(prop5, axiom,  pi0(pg,s(Bm,Bw,b(tt,ff),Bc))).
cnf(prop6, axiom, ~pi0(pg,s(Bm,Bw,b(ff,tt),Bc))).
cnf(prop7, axiom,  pi0(pc,s(Bm,Bw,Bg,b(tt,ff)))).
cnf(prop8, axiom, ~pi0(pc,s(Bm,Bw,Bg,b(ff,tt)))).

cnf(r1, axiom, r(s(b(T,F), b(F,T), b(F,T), b(F,T)),
                 con(s(b(F,T), b(F,T), b(F,T), b(F,T)),nil))).
cnf(r2, axiom, r(s(b(T,F), b(T,F), b(F,T), b(F,T)),
                 con(s(b(F,T), b(T,F), b(F,T), b(F,T)),
```

```
                con(s(b(F,T), b(F,T), b(F,T), b(F,T)),nil))))).
cnf(r3, axiom, r(s(b(T,F), b(F,T), b(T,F), b(F,T)),
                con(s(b(F,T), b(F,T), b(T,F), b(F,T)),
                con(s(b(F,T), b(F,T), b(F,T), b(F,T)), nil)))).
cnf(r4, axiom, r(s(b(T,F), b(F,T), b(F,T), b(T,F)),
                con(s(b(F,T), b(F,T), b(F,T), b(T,F)),
                con(s(b(F,T), b(F,T), b(F,T), b(F,T)), nil)))).
cnf(r5, axiom, r(s(b(T,F), b(T,F), b(T,F), b(F,T)),
                con(s(b(F,T), b(T,F), b(T,F), b(F,T)),
                con(s(b(F,T), b(F,T), b(T,F), b(F,T)),
                con(s(b(F,T), b(T,F), b(F,T), b(F,T)), nil))))).
cnf(r6, axiom, r(s(b(T,F), b(T,F), b(F,T), b(T,F)),
                con(s(b(F,T), b(T,F), b(F,T), b(T,F)),
                con(s(b(F,T), b(F,T), b(F,T), b(T,F)),
                con(s(b(F,T), b(T,F), b(F,T), b(F,T)), nil))))).
cnf(r7, axiom, r(s(b(T,F), b(F,T), b(T,F), b(T,F)),
                con(s(b(F,T), b(F,T), b(T,F), b(T,F)),
                con(s(b(F,T), b(F,T), b(F,T), b(T,F)),
                con(s(b(F,T), b(F,T), b(T,F), b(F,T)), nil))))).
cnf(r8, axiom, r(s(b(T,F), b(T,F), b(T,F), b(T,F)),
                con(s(b(F,T), b(T,F), b(T,F), b(T,F)),
                con(s(b(F,T), b(F,T), b(T,F), b(T,F)),
                con(s(b(F,T), b(T,F), b(F,T), b(T,F)),
                con(s(b(F,T), b(T,F), b(T,F), b(F,T)), nil))))))).

%Negation of the specification
cnf(check, negated_conjecture,
    pi0(ar(or(and(pw,and(pg, not(pm))),or(and(not(pw),and(not(pg),pm)),
           or(and(pc,and(pg,not(pm))),and(not(pc),and(not(pg),pm))))),
           or(not(pc),or(not(pg),or(not(pw),not(pm))))),
        s(b(ff,tt),b(ff,tt),b(ff,tt),b(ff,tt)))).
```

Bibliography

[Amj04] Hasan Amjad. *Combining Model Checking and Theorem Proving*. PhD thesis, Citeseer, 2004.

[BAPM83] Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. The Temporal Logic of Branching Time. *Acta Informatica*, 20(3):207–226, 1983.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded Model Checking. *Advances in computers*, 58:117–148, 2003.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In W.Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer Berlin Heidelberg, 1999.

[BCM+90] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic Model Checking: 10^{20} States and Bbeyond. In *Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 428–439. IEEE, 1990.

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97: Concurrency Theory*, pages 135–150. Springer, 1997.

[Bri13] Jill Britton. *The Wolf, the Goat and the Cabbage*, 2013.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[BS01] Bernhard Beckert and Steffen Schlager. A Sequent Calculus for First-order Dynamic Logic with Trace Modalities. In *Proceedings, International Joint Conference on Automated Reasoning*, pages 626–641. Springer, 2001.

[Bur10] Guillaume Burel. Embedding Deduction Modulo into a Prover. In Anuj Dawar and Helmut Veith, editors, *CSL 2010*, volume 6247 of *LNCS*, pages 155–169. Springer Berlin Heidelberg, 2010.

[Bur11] Guillaume Burel. Experimenting with Deduction Modulo. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE-23*, volume 6803 of *LNCS*, pages 162–176. Springer Berlin Heidelberg, 2011.

[CBRZ01] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[CCGR99] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A New Symbolic Model Verifier. In *Proceedings of the 11th International Conference on Computer Aided Verification*, CAV '99, pages 495–499, London, UK, UK, 1999. Springer-Verlag.

[CE12] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-order Logic: a Language-Theoretic Approach*, volume 138. Cambridge University Press, 2012.

[CGP99] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[CL97] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., Orlando, FL, USA, 1st edition, 1997.

[Cou90] Bruno Courcelle. The Monadic Second-order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Information and computation*, 85(1):12–75, 1990.

[DDG+13] David Delahaye, Damien Doligez, Frédŕic Gilbert, Pierre Halmagrand, and Olivier Hermant. Zenon Modulo: When Achilles Outruns the Tortoise Using Deduction Modulo. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR-19*, volume 8312 of *LNCS*, pages 274–290. Springer Berlin Heidelberg, 2013.

[DHK03] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning*, 31:33–72, 2003.

[DJ13a] Gilles Dowek and Ying Jiang. A Logical Approach to CTL. manuscript, 2013.

[DJ13b] Gilles Dowek and Ying Jiang. Axiomatizing Truth in a Finite Model. manuscript, 2013.

[DJ15] Gilles Dowek and Ying Jiang. Pushdown Systems in Polarized Deduction Modulo. manuscript, 2015.

[Dow02] Gilles Dowek. What Is a Theory? In Helmut Alt and Afonso Ferreira, editors, *STACS 2002*, volume 2285 of *LNCS*, pages 50–64. Springer Berlin Heidelberg, 2002.

[Dow10] Gilles Dowek. Polarized Resolution Modulo. In Cristian S. Calude and Vladimiro Sassone, editors, *TCS 2010*, volume 323 of *IFIP AICT*, pages 182–196. Springer Berlin Heidelberg, 2010.

[Dow11] Gilles Dowek. *Proofs and Algorithms: An Introduction to Logic and Computability*. Springer-Verlag London, 2011.

[EC80] E.Allen Emerson and Edmund M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer Berlin Heidelberg, 1980.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time Temporal Logic. *J. ACM*, 33(1):151–178, 1986.

[EHRS00] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient Algorithms for Model Checking Pushdown Systems. In *Computer Aided Verification*, pages 232–247. Springer, 2000.

[Eme95] E. Allen Emerson. Temporal and Modal Logic. In *HANDBOOK OF THEORETICAL COMPUTER SCIENCE*, pages 995–1072. Elsevier, 1995.

[ES04] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer Berlin Heidelberg, 2004.

[Fis91] Michael Fisher. A Resolution Method for Temporal Logic. In *IJCAI*, volume 91, pages 99–104. Citeseer, 1991.

[GK06] Harald Ganzinger and Konstantin Korovin. Theory Instantiation. In Miki Hermann and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4246 of *Lecture Notes in Computer Science*, pages 497–511. Springer Berlin Heidelberg, 2006.

[GlGl02] J. Goubault-larrecq and Jean Goubault-larrecq. A Note on the Completeness of Certain Refinements of Resolution, 2002.

[Gor14] Rajeev Goré. And-Or Tableaux for Fixpoint Logics with Converse: LTL, CTL, PDL and CPDL. In *Automated Reasoning*, pages 26–45. Springer, 2014.

[G.S83] G.S.Tseitin. On the Complexity of Derivation in Propositional Calculus. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning*, Symbolic Computation, pages 466–483. Springer Berlin Heidelberg, 1983.

[HC68] George Edward Hughes and M. J. Cresswell. *An Introduction to Modal Logic*. Methuen, 1968.

[HJB10] Marijn Herle, Matti Järvisalo, and Armin Biere. Clause Elimination Procedures for CNF Formulas. In Christian G. Fermüller and Andrei Voronkov, editors, *LPAR-17*, volume 6397 of *LNCS*, pages 357–371. Springer Berlin Heidelberg, 2010.

[Ji15a] Kailiang Ji. CTL Model Checking in Deduction Modulo. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 295–310. Springer International Publishing, 2015.

[Ji15b] Kailiang Ji. Resolution in Solving Graph Problems. manuscript, 2015.

[Kor08] Konstantin Korovin. iProver– An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, volume 5195 of *LNCS*, pages 292–298. Springer Berlin Heidelberg, 2008.

[Koz82] Dexter Kozen. Results on the Propositional Mu-calculus. In Mogens Nielsen and ErikMeineche Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 348–359. Springer Berlin Heidelberg, 1982.

[KSU13] Daniel Kühlwein, Stephan Schulz, and Josef Urban. E-males 1.1. In MariaPaola Bonacina, editor, *Automated Deduction CADE-24*, volume 7898 of *Lecture Notes in Computer Science*, pages 407–413. Springer Berlin Heidelberg, 2013.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *Computer Aided Verification*, pages 1–35. Springer, 2013.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[McM02] Ken L McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In *Computer Aided Verification*, pages 250–264. Springer, 2002.

[Pet81] G. L. Peterson. Myths About the Mutual Exclusion Problem. *Information Processing Letters*, 12:115–116, 1981.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[PWZ02] Wojciech Penczek, Bozena Wozna, and Andrzej Zbrzezny. Bounded Model Checking for the Universal Fragment of CTL. *Fundam. Inf.*, 51:135–156, 2002.

[Rob65a] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12:23–41, 1965.

[Rob65b] J.A. Robinson. Automatic deduction with hyper-resolution. *Intl. J. Computer Mathematics*, 1(3):227–234, 1965.

[RSS95] S. Rajan, N. Shankar, and M.K. Srivas. An Integration of Model-Checking with Automated Proof Checking. In Pierre Wolper, editor, *Computer-Aided Verification, CAV '95*, volume 939 of *Lecture Notes in Computer Science*, pages 84–97, Liege, Belgium, 1995. Springer-Verlag.

[SNW96] Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for Concurrency: Towards a Classification. *Theoretical Computer Science*, 170:297–348, 1996.

[Sut09] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[TS96] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, New York, NY, USA, 1996.

[ZA95] Manna Zohar and Pnueli Amir. *Temporal Verification of Reactive Systems—Safety*. Springer-Verlag, New York, 1995.

[Zha09] Wenhui Zhang. Bounded Semantics of CTL and SAT-Based Verification. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM 2009*, volume 5885 of *LNCS*, pages 286–305. Springer Berlin Heidelberg, 2009.

[Zha12] Wenhui Zhang. *VERDS Modeling Language*, 2012. http://lcs.ios.ac.cn/~zwh/verds/index.html.

[Zha13] Wenhui Zhang. Ternary Boolean Diagrams and Their Application to Model Checking. manuscript, 2013.

[Zha14] Wenhui Zhang. QBF Encoding of Temporal Properties and QBF-based Verification. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR 2014*, volume 8562 of *LNCS*, pages 224–239. Springer International Publishing, 2014.

[ZHD14] Lan Zhang, Ullrich Hustadt, and Clare Dixon. A Resolution Calculus for the Branching-time Temporal Logic CTL. *ACM Transactions on Computational Logic (TOCL)*, 15(1):10, 2014.