eman ta zabal zazu

Universidad del País Vasco Euskal Herriko Unibertsitatea

Department of Computer Languages and Systems
Departamento de Lenguajes y Sistemas Informáticos
Lengoaia eta Sistema Informatikoak Saila

# Invariant-Free Deduction Systems for Temporal Logic

A **dissertation**
submitted to the
Department of Computer Languages and Systems
of the University of the Basque Country
in partial fulfillment of the requirements for the
degree of

**Doctor of Philosophy**
with "International PhD" mention

JOSE GAINTZARAIN IBARMIA

ADVISOR: DR. PAQUI LUCIO CARRASCO

San Sebastián, Spain, May 2012

# Invariant-Free Deduction Systems for Temporal Logic

JOSE GAINTZARAIN IBARMIA

ADVISOR: DR. PAQUI LUCIO CARRASCO

A **dissertation**
submitted to the
Department of Computer Languages and Systems
of the University of the Basque Country
in partial fulfillment of the requirements for the
degree of

**Doctor of Philosophy**
with "International PhD" mention

University of the Basque Country
Universidad del País Vasco / Euskal Herriko Unibertsitatea

Department of Computer Languages and Systems
Departamento de Lenguajes y Sistemas Informáticos
Lengoaia eta Sistema Informatikoak Saila

San Sebastián, Spain, May 2012

You cannot stop the wind but you may build windmills.
— Dutch proverb

No puedes detener el viento pero puedes construir molinos de viento.
— Proverbio neerlandés

Haizea ezin dezakezu gelditu baina haize-errotak eraiki ditzakezu.
— Herbeheretar esaera

## ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Paqui Lucio. This thesis was completed thanks to her wise guidance, her strong determination, her encouragement, her hard work and her patience. I would also like to thank Montserrat Hermo, Marisa Navarro and Fernando Orejas for their huge contribution to this thesis. I also acknowledge my gratitude to Jon Ander Hernández and Iñigo Sola for their contribution to implementation issues. I am also grateful to Alexander Bolotov for making possible a four month stay at the University of Westminster and for his useful comments and suggestions that helped to improve the final version of this thesis. I would also like to thank Michael Fisher for providing feedback on the work presented in this thesis by organizing a research visit to his group in Liverpool and for his valuable comments on the thesis.

Finally, I thank everyone that in one way or another contributed to the completion of this thesis. Among them, my father (Luis) and my mother (Miren Edurne) deserve special mention.

I dedicate this thesis to my parents.

## ABSTRACT

In this thesis we propose a new approach to deduction methods for temporal logic. Our proposal is based on an inductive definition of eventualities that is different from the usual one. On the basis of this non-customary inductive definition for eventualities, we first provide dual systems of tableaux and sequents for Propositional Linear-time Temporal Logic (PLTL). Then, we adapt the deductive approach introduced by means of these dual tableau and sequent systems to the resolution framework and we present a clausal temporal resolution method for PLTL. Finally, we make use of this new clausal temporal resolution method for establishing logical foundations for declarative temporal logic programming languages.

The key element in the deduction systems for temporal logic is to deal with eventualities and "hidden" invariants that may prevent the fulfillment of eventualities. Different ways of addressing this issue can be found in the works on deduction systems for temporal logic.

Traditional tableau systems for temporal logic generate an auxiliary graph in a first pass. Then, in a second pass, unsatisfiable nodes are pruned. In particular, the second pass must check whether the eventualities are fulfilled. The one-pass tableau calculus introduced by S. Schwendimann requires an additional handling of information in order to detect cyclic branches that contain unfulfilled eventualities. Regarding traditional sequent calculi for temporal logic, the issue of eventualities and hidden invariants is tackled by making use of a kind of inference rules (mainly, invariant-based rules or infinitary rules) that complicates their automation. A remarkable consequence of using either a two-pass approach based on auxiliary graphs or a one-pass approach that requires an additional handling of information in the tableau framework, and either invariant-based rules or infinitary rules in the sequent framework, is that temporal logic fails to carry out the classical correspondence between tableaux and sequents. In this thesis, we first provide a one-pass tableau method TTM that instead of a graph obtains a cyclic tree to decide whether a set of PLTL-formulas is satisfiable. In TTM tableaux are classical-like. For unsatisfiable sets of formulas, TTM produces tableaux whose leaves contain a formula and its negation. In the case of satisfiable sets of formulas, TTM builds tableaux where each fully expanded open branch characterizes a collection of models for the set of formulas in the root. The tableau method TTM is complete and yields a decision procedure for PLTL. This tableau method is directly associated to a one-sided sequent calculus called TTC. Since TTM is free from all the structural rules that hinder the mechanization of deduction, e.g. weakening and contraction, then the resulting sequent calculus TTC is also free from this kind of structural rules. In particular, TTC is free of any kind of cut, including invariant-based cut. From the deduction system TTC, we obtain a two-sided sequent calculus GTC that preserves all these good freeness properties and is finitary, sound and complete for PLTL. Therefore, we show that the classical correspondence between tableaux and sequent calculi can be extended to temporal logic.

The most fruitful approach in the literature on resolution methods for temporal logic, which was started with the seminal paper of M. Fisher, deals with PLTL and requires to generate invariants for performing resolution on eventualities. In this thesis, we present a new approach

to resolution for PLTL. The main novelty of our approach is that we do not generate invariants for performing resolution on eventualities. Our method is based on the dual methods of tableaux and sequents for PLTL mentioned above. Our resolution method involves translation into a clausal normal form that is a direct extension of classical CNF. We first show that any PLTL-formula can be transformed into this clausal normal form. Then, we present our temporal resolution method, called TRS-resolution, that extends classical propositional resolution. Finally, we prove that TRS-resolution is sound and complete. In fact, it finishes for any input formula deciding its satisfiability, hence it gives rise to a new decision procedure for PLTL.
In the field of temporal logic programming, the declarative proposals that provide a completeness result do not allow eventualities, whereas the proposals that follow the imperative future approach either restrict the use of eventualities or deal with them by calculating an upper bound based on the small model property for PLTL. In the latter, when the length of a derivation reaches the upper bound, the derivation is given up and backtracking is used to try another possible derivation. In this thesis we present a declarative propositional temporal logic programming language, called TeDiLog, that is a combination of the temporal and disjunctive paradigms in Logic Programming. We establish the logical foundations of our proposal by formally defining operational and logical semantics for TeDiLog and by proving their equivalence. Since TeDiLog is, syntactically, a sublanguage of PLTL, the logical semantics of TeDiLog is supported by PLTL logical consequence. The operational semantics of TeDiLog is based on TRS-resolution. TeDiLog allows both eventualities and always-formulas to occur in clause heads and also in clause bodies. To the best of our knowledge, TeDiLog is the first declarative temporal logic programming language that achieves this high degree of expressiveness.

Since the tableau method presented in this thesis is able to detect that the fulfillment of an eventuality is prevented by a hidden invariant without checking for it by means of an extra process, since our finitary sequent calculi do not include invariant-based rules and since our resolution method dispenses with invariant generation, we say that our deduction methods are invariant-free.

# CONTENTS

# LIST OF FIGURES

# 1. INTRODUCTION

Temporal logic plays a significant role in computer science, since it is an appropriate tool for specifying object behaviour, cooperative protocols, reactive systems, digital circuits, concurrent programs and, in general, for reasoning about dynamic systems whose states change over time (see e.g. [46, 56, 57, 86, 90, 91]). In particular, several concepts which are useful for the specification of properties of dynamic systems –such as fairness, non-starvation, liveness, safety, mutual exclusion, etc– can be formally stated in temporal logic using very concise and readable formulas. Several different temporal logics have been devised –as formalisms for representing dynamic systems– that mainly differ in their underlying model of time and in their expressiveness. Regarding time modeling there are linear vs. branching, discrete vs. dense, future vs. past-and-future, finite vs infinite, etc. Regarding expressiveness, they involve different temporal connectives and logical constructions (such as, quantifiers, variables, fixpoint operators). For a recent survey on temporal logics we refer the reader to [85].

Propositional Linear-time Temporal Logic (PLTL) is one of the most widely used temporal logics [1]. This logic has, as the intended model for time, the standard model of natural numbers. Different contributions in the literature on temporal logic show its usefulness in computer science and other related areas. For a recent and extensive monograph on PLTL techniques and tools, we refer the reader to [44], where sample applications along with references to specific works that use this temporal formalism to represent dynamic entities in a wide variety of fields –such as Program Specification, System Verification, Robotics, Reactive Systems, Databases, Control Systems, Agent-based Systems, etc– can be found. The minimal language for PLTL adds to classical propositional connectives two basic temporal connectives $\circ$ ("next") and $\mathcal{U}$ ("until") such that $\circ\psi$ is interpreted as "the next state makes $\psi$ true" and $\varphi\,\mathcal{U}\,\psi$ is interpreted as "$\varphi$ is true from now until $\psi$ eventually becomes true". Many other useful temporal connectives can be defined as derived connectives, e.g. $\diamond$ ("eventually"), $\Box$ ("always") and $\mathcal{R}$ ("release").

Automated reasoning for temporal logic is a quite recent trend. In temporal logics, as well as in the more general framework of modal logic, different proof methods are starting to be designed, implemented, compared, and improved. Specification and verification methods for PLTL –and also for other temporal logics– are mainly based on three kinds of proposals: automata, tableaux and resolution. Automata are related to model checking whereas tableaux and resolution are the main methods for proof theory. Other proof-theoretic approaches for PLTL include its first axiomatization à la Hilbert presented in [53]. See [110] for a good survey about theorem-proving in PLTL and its extensions. The most developed approach is model checking ([30, 31]), which is automata-based. In fact, model checking of temporal formulas is traditionally carried out by a conversion to Büchi automata (see e.g. [120]). In model checking, temporal logic is used for specification purposes, whereas the system is often implemented in a different language, hence verification requires to manage different semantic domains. Model checking

[1] Probably, the most used temporal logic is Computation Tree Logic (CTL), especially for model checking purposes.

focuses on the problem of deciding whether a concrete model (or run) of a system satisfies a logical formula or not. This approach is reasonably efficient for finite state systems and there is a large body of research in this area. The interested reader is referred to [44] (Section 4.4.7 and Chapter 5) for a recent work that describes model checking techniques. However, the automata approach is not well suited for automated deduction, in the sense that it is not able to generate proofs or deductions of a conclusion from a set of premises. A brief and clarifying discussion about model checking versus deductive temporal verification can be found in [35].

Automated deduction for PLTL, and related logics, is mainly based on tableaux and resolution. Indeed, there are recently published works comparing implementations of the different tableau and resolution procedures for PLTL and similar logics (see e.g. [69, 77, 78]).

In this thesis we propose new deduction methods for PLTL. In particular, we introduce a tableau method, two sequent calculi and a resolution procedure for PLTL. On the basis of the resolution procedure, we also present a declarative temporal logic programming language.

## *Eventualities and Invariants*

In every deduction method for temporal logic, the central topic is how to deal with eventualities and “hidden” invariants that can prevent the fulfillment of eventualities. Eventualities directly state that a property will eventually hold whereas invariants state, often in an intricate way, that a property holds at every time instant from some moment onwards.

The use of the customary inductive definitions of the temporal connectives as the only mechanism for detecting the existence of an invariant that prevents the fulfillment of an eventuality, leads to incomplete deduction systems. The reason is that such customary inductive definitions make possible to indefinitely postpone the fulfillment of an eventuality and, consequently, they make possible to indefinitely postpone the contradiction between an eventuality that states that a property $\psi$ will eventually hold and an invariant that states that $\psi$ will never hold. Therefore, more elaborated mechanisms are needed.

Next, we review how this issue is tackled by the main approaches in the tableau, sequent, resolution and temporal logic programming frameworks. Additionally we describe our contribution to each of these frameworks.

## *Tableau systems*

Traditional tableau systems for temporal logic, in particular PLTL, are based on the usual inductive definition of eventualities (see e.g. [128, 73, 8, 87, 79, 81]). In order to obtain completeness, they first build an auxiliary finite graph by using tableau rules. Since in these systems, the number of different sets of formulas that can be produced from the initial set is finite, the graph is always finite. Once the graph is completed, it is checked to detect the existence of unfulfilled eventualities. Nodes that do not belong to infinite paths that give rise to models, are pruned. These tableau methods are known as two-pass methods. The one-pass approach proposed in [117] is also based on the usual inductive definition of eventualities. The method yields cyclic trees. The second pass is avoided by associating additional information to nodes. Part of the information is generated in a top-down manner, while the branches are being built. But there are also information that is obtained in a bottom-up manner, once the branch has been completed. The information obtained in a bottom-up manner is necessary to deal with cyclic branches that are not fulfilling on their own but yield a fulfilling cycle if combined with other accessible branches. From a theoretical point of view, one of the drawbacks of the two-pass approach and the above mentioned one-pass approach is that a classical-like tableau is not obtained. We mean that, unsatisfiable sets of formulas do not always produce closed branches whose last nodes contain a formula and its negation, instead cycles that do not lead to models must be detected by using an extra process. Our proposal is based on a non-customary inductive definition of eventualities. The rule obtained from this alternative inductive definition of eventualities, together with a specific strategy for applying the tableau rules, gives rise to a tableau method, namely TTM, where tableaux are cyclic trees and unsatisfiability is exclusively detected –like in classical tableau methods– by means of closed branches that contain a formula and its negation in its last node. Additionally, by controlling cycles that only belong to a single branch, a decision procedure is obtained. Our approach was first presented in [60] and then extended in [61]. A preliminary prototype is accessible in `http://www.sc.ehu.es/jiwlucap/TTM.html`. A report about this prototype is presented in [62].

## *Sequent systems*

Traditional sequent systems for temporal logic (see e.g. [104, 105, 121]) are also based on the usual inductive definition of eventualities. In order to deal with eventualities and invariants, they either include an infinitary rule or a rule that requires to previously find an adequate invariant. On one hand infinitary rules are not effective. On the other hand, invariant-based rules are specialized cut rules that prevent from obtaining classical-like cut-free proofs. As a consequence of using either two-pass methods based on auxiliary graphs or a one-pass system that requires an additional handling of information in the tableau framework, and either infinitary rules or invariant-based rules in the sequent framework, is that the classical duality between tableau and sequent proofs does not hold. The finitary sequent system presented in [20] does not require an invariant-based rule but annotated formulas are used. These formulas do not properly belong to the logical language, so that an extra-logical feature is used. By following our approach based on a non-customary inductive definition of eventualities, we propose a finitary sequent calculus that does not include either invariant-based rules or rules that contain extra-logical features such as annotated formulas. Moreover, our tableau and sequent systems are dual in the sense that from every tableau construction a sequent derivation can be straightforwardly obtained. Our proposal was first materialized by means of the sequent system $\mathcal{FC}$ presented in [58], which is the first finitary sequent system for PLTL that is free from cut- and invariant-based rules. Later on, the sequent systems TTC and GTC were directly obtained from the tableau method TTM ([61]). Although $\mathcal{FC}$ and GTC are basically identical, the completeness proof for GTC is based on its duality with respect to TTM and –unlike in the completeness proof of $\mathcal{FC}$– structural rules such as weakening and contraction are not used.

## *Clausal Resolution*

The clausal resolution methods for PLTL presented in [126] and [40] (see also [45]) require invariant generation in order to deal with eventualities. The former does not tackle the invariant generation issue whereas the latter provides an algorithm. The resolution system in [40] gives rise to a decision procedure for PLTL, but it contains an extra-logical feature to resolve eventualities. The clausal resolution method introduced in [29] is not intended for full PLTL and the approach is based on the exhaustive analysis of all the possible transformations (in a finite scope) of eventualities into formulas that only contain the $\circ$ connective. The non-clausal resolution system presented in [1] is based on a non-customary inductive definition of eventualities that is different from the one we consider in the above mentioned tableau and sequent frameworks. However, the problem of satisfiable input sets is not addressed in [1] and therefore a decision procedure for PLTL is not provided. Our clausal resolution method, namely TRS, is defined by adapting the TTM approach for tableaux to the clausal resolution framework. Consequently, the keys of our approach to temporal resolution are a rule that deals with eventualities and a strategy formalized by means of a systematic resolution algorithm that gives rise to a resolution-based decision procedure for PLTL. This resolution method is also described in detail in [62]. A prototype for TRS-resolution can be found in `http://www.sc.ehu.es/jiwlucap/TRS.html`.

## ***Temporal Logic Programming***

The idea of directly executing logical formulas and, therefore, using logic as a programming language –already proven successful in classical Logic Programming– has also been tackled in the case of temporal logic. Temporal Logic Programming provides a single framework in which dynamic systems can be specified, developed, validated and verified by means of executable specifications that make possible to prototype, debug and improve systems before their final use. In classical Logic Programming, the underlying execution procedure is based on (classical) clausal resolution ([88, 89]). The extension of this approach to Temporal Logic Programming faces three main challenges: the undecidability of first-order temporal logic [92, 122, 121], the difficulty for dealing with eventualities and invariants and the complexity (even for the propositional fragment [119]).

Consequently, different proposals that can be classified into two groups have arisen. One of the groups is formed by the languages that are based on the imperative future approach (e.g. [94, 9, 93]). In these languages programs are formulas –written in temporal logic– that state which literals must be true in the next state. So the execution consists in explicitly building the model for the program, state by state. The other group is formed by the languages that are based on the declarative approach. The declarative languages extend classical Logic Programming for reasoning about time. However, some of the declarative languages are not purely based on temporal logic (e.g. [83, 74, 21, 50, 114]). The declarative languages that are purely based on temporal logic extend classical Logic Programming by including temporal connectives in the atoms and by also extending classical resolution ([2, 12, 127, 99, 55]). Here we only analyze the languages that belong to the imperative future approach and the declarative languages that are purely based on temporal logic. The languages that belong to the imperative future approach either restrict the use of eventualities (e.g. [94, 93]) or use the finite-model property[2] for fixing an upper bound that indicates that an eventuality cannot be fulfilled (e.g. [9]). The declarative languages that are purely based on temporal logic either directly avoid eventualities ([2, 12, 127, 99]) or do not provide completeness result ([55]). If the clausal temporal resolution method presented in [40] were considered as a basis for a declarative temporal logic programming language, its execution would require invariant generation. In the same way, the sequent-based logical foundation for declarative temporal logic programming provided in [106] includes an invariant-based rule. We propose a (propositional) declarative temporal logic programming language, named TeDiLog, whose execution mechanism is based on TRS-resolution. Consequently there are no restrictions regarding the use of eventualities. Moreover, we deal

[2] Also known as small model property.

with eventualities without requiring invariant generation. A preliminary version of this proposal was presented in [64].

### *Invariant-Freeness*

In order to sum up and highlight the distinctive feature of our approach to temporal deduction we can say that:

- Our tableau method is classical-like in the sense that it does not require an extra process (a second pass or an additional handling of information) for detecting the unsatisfiability of a set of formulas where an invariant prevents the fulfillment of an eventuality.
- Our finitary sequent systems do not include invariant-based rules.
- Our resolution method dispenses with invariant generation.
- The resolution procedure underlying our temporal logic programming language does not require either invariant generation or invariant detection by means of upper bounds.

Consequently, we say that our approach is invariant-free.

## *Outline of the thesis*

This thesis is organized in six chapters (including this one) as follows:

- In **Chapter 2**, we provide the preliminaries of the thesis, that is, the basic notions about PLTL that are used in the remaining chapters.
- In **Chapter 3**, we first introduce our one-pass tableau method TTM. This tableau method includes a new tableau rule for dealing with eventualities. The completeness result of TTM is based on this rule and the strategy formalized by means of the systematic tableau algorithm that we also present in this chapter. Such rule together with the mentioned strategy are the core of our proposal, which leads to a new approach to temporal deduction and gives rise to a new decision procedure for PLTL. From TTM, we obtain the one-sided finitary sequent calculus TTC that is cut-free and invariant-free. On the basis of TTC, we finally define the two-sided sequent calculus GTC, which is also finitary, cut-free and invariant-free. Moreover, both TTC and GTC are weakening-free and contraction-free. By means of these tableau and sequent systems we prove that the classical duality between tableau and sequent systems extends to PLTL. At the beginning of Chapter 3 we review related work to motivate our research. At the end of the chapter, we compare our approach with related work with the aim of remarking the novelties of our contribution. The contents of this chapter are strongly based on [58, 60, 61, 63].
- **Chapter 4** is devoted to our clausal temporal resolution method TRS. First, we briefly review previously existing approaches to motivate our work. Then, we introduce our clausal normal form and the steps required to transform any PLTL formula into this clausal form. Next, we provide the rule system and the notion of derivation. The crucial rule for eventualities and the systematic resolution algorithm that lead to the completeness result and to the new resolution-based decision procedure, are obtained by adapting the key

rule and the systematic tableau algorithm from the previous chapter to the clausal setting. The major novelty of the resolution method TRS is that, unlike the main approach in the literature ([40, 45]), it dispenses with invariant generation. The last section of this chapter is used to compare our contribution with previously existing approaches. The content of this chapter is based on [62].

- In **Chapter 5**, we present a declarative propositional temporal logic programming language called TeDiLog. First we introduce the syntax of TeDiLog which is an adaptation of the clausal form introduced in the previous chapter to the logic programming style. Then we provide operational and logical semantics for TeDiLog and prove their equivalence. The operational semantics is based on the TRS-resolution presented in Chapter 4 and the logical semantics is founded on PLTL logical consequence. At the beginning of the chapter previous approaches are reviewed to motivate our work and at the end of the chapter our contribution is compared with such approaches. The content of this chapter extends [64].

- In **Chapter 6**, we expose the main contributions and results of this thesis together with the publications and remarkable research activity carried out during the preparation of this thesis. Besides, we also discuss future work.

# 2. PRELIMINARIES

In this chapter we provide the basic notions related to PLTL and we also introduce some notation that is used in this thesis. Sections 2.1 and 2.2 are devoted to the syntax and the semantics of PLTL. Section 2.3 introduces the notions of soundness, refutational completeness and completenes for deduction systems. Finally, Section 2.4 introduces the notion of invariant formula for PLTL.

## *2.1 Syntax of* PLTL

The syntax of PLTL extends the syntax of classical propositional logic by allowing the use of temporal connectives. Different temporal connectives can be considered in order to obtain the full expressiveness of PLTL. In this thesis we choose the temporal connectives $\circ$ ("next") and $\mathcal{U}$ ("until") as primitive temporal connectives. Therefore we say that PLTL-formulas are built by using the nullary connective (i.e. the constant) $\textbf{F}$, propositional variables (denoted by lowercase letters $p$, $q$, ...) from a set Prop, the classical connectives $\neg$ and $\wedge$, and the temporal connectives $\circ$ and $\mathcal{U}$. In the sequel, *formula* means PLTL-formula. A lowercase Greek letter ($\varphi$, $\psi$, $\chi$, $\gamma$, ...) denotes a formula and an uppercase one ($\Phi$, $\Delta$, $\Gamma$, $\Psi$, $\Omega$, ...) denotes a finite set of formulas. As usual other connectives can be defined in terms of the previous ones: $\textbf{T} \equiv \neg\textbf{F}$, $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \mathcal{R} \psi \equiv \neg(\neg\varphi \mathcal{U} \neg\psi)$, $\diamond\varphi \equiv \textbf{T}\mathcal{U}\varphi$, $\Box\varphi \equiv \neg\diamond\neg\varphi$. Note that $\Box\varphi \equiv \textbf{F}\mathcal{R}\varphi$. As can be observed in the above definitions, the linear-time connectives $\mathcal{R}$ ("release"), $\diamond$ ("eventually") and $\Box$ ("always") can be defined in terms of the connective $\mathcal{U}$.

The connectives $\textbf{T}$, $\vee$, $\mathcal{R}$ and $\Box$ are the dual connectives of the connectives $\textbf{F}$, $\wedge$, $\mathcal{U}$ and $\diamond$ respectively. The connective $\circ$ is its own dual.

The above defined connectives will be used as abbreviations for readability in the tableau method and the sequent calculi but dual connectives are necessary in the clausal resolution method. For technical convenience, we use the nullary connective $\textbf{F}$ as part of the minimal language for PLTL. However, its use can be avoided by considering that $\textbf{F}$ can be expressed as $\psi \wedge \neg\psi$, where $\psi \in$ Prop. In fact, in the clausal resolution method we dispense with the constants $\textbf{F}$ and $\textbf{T}$ and we consider that $\diamond\varphi \equiv \neg\varphi\mathcal{U}\varphi$ and $\Box\varphi \equiv \neg\varphi\mathcal{R}\varphi$. In the clausal resolution method the empty clause is denoted syntactically as $\bot$ and $\Box\bot$.

Formulas of the form $\psi$, $\neg\psi$ and $\circ\varphi$, where $\psi \in \{\textbf{F}, \textbf{T}\} \cup$ Prop, are called *elementary*. Also sets of elementary formulas are called elementary.

We denote by $\circ^n$, $\Box^n$ and $\diamond^n$, with $n \geq 0$, the sequences of $n$ connectives $\circ$, $\Box$ and $\diamond$, respectively. However, these kinds of superscripts are notation, hence they are not part of the syntax.

Given a set of formulas $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ we use $\neg\Phi$ to denote the formula $\neg(\varphi_1 \wedge \ldots \wedge \varphi_n)$ and $\bigwedge\Phi$ to denote the formula $\varphi_1 \wedge \ldots \wedge \varphi_n$. In particular, when $\Phi$ is empty, $\neg\Phi$ is the constant $\textbf{F}$ in the tableau and sequent systems (Chapter 3) and the empty clause in the resolution system

(Chapters 4 and 5). On the other hand, when $\Phi$ is empty, $\bigwedge \Phi$ is the constant $\mathbf{T}$ in the tableau and sequent systems (Chapter 3).

## 2.2 Semantics and Model Theory of PLTL

Formally, a PLTL-*structure* $\mathcal{M}$ is a pair $(S_{\mathcal{M}}, V_{\mathcal{M}})$ such that $S_{\mathcal{M}}$ is a denumerable sequence of states $s_0, s_1, s_2, \ldots$ and $V_{\mathcal{M}}$ is a map $V_{\mathcal{M}} : S_{\mathcal{M}} \to 2^{\mathsf{Prop}}$. Intuitively, $V_{\mathcal{M}}(s_j)$ specifies which atomic propositions are (necessarily) true in the state $s_j$.

Formulas are interpreted in the states of PLTL-structures. The formal semantics of formulas is given by the truth of a formula $\varphi$ in the state $s_j$ of a PLTL-structure $\mathcal{M}$, which is denoted by $\langle \mathcal{M}, s_j \rangle \models \varphi$. This semantics is inductively defined as follows:

- $\langle \mathcal{M}, s_j \rangle \not\models \mathbf{F}$
- $\langle \mathcal{M}, s_j \rangle \models p$ iff $p \in V_{\mathcal{M}}(s_j)$ for $p \in \mathsf{Prop}$
- $\langle \mathcal{M}, s_j \rangle \models \neg\varphi$ iff $\langle \mathcal{M}, s_j \rangle \not\models \varphi$
- $\langle \mathcal{M}, s_j \rangle \models \varphi \wedge \psi$ iff $\langle \mathcal{M}, s_j \rangle \models \varphi$ and $\langle \mathcal{M}, s_j \rangle \models \psi$
- $\langle \mathcal{M}, s_j \rangle \models \circ\varphi$ iff $\langle \mathcal{M}, s_{j+1} \rangle \models \varphi$
- $\langle \mathcal{M}, s_j \rangle \models \varphi \,\mathcal{U}\, \psi$ iff there exists $k \geq j$ such that $\langle \mathcal{M}, s_k \rangle \models \psi$ and for every $i \in \{j, \ldots, k-1\}$ it holds $\langle \mathcal{M}, s_i \rangle \models \varphi$.

The extension of the above formal semantics to the defined connectives yields:

- $\langle \mathcal{M}, s_j \rangle \models \mathbf{T}$
- $\langle \mathcal{M}, s_j \rangle \models \varphi \vee \psi$ iff $\langle \mathcal{M}, s_j \rangle \models \varphi$ or $\langle \mathcal{M}, s_j \rangle \models \psi$
- $\langle \mathcal{M}, s_j \rangle \models \varphi \,\mathcal{R}\, \psi$ iff for every $k \geq j$ it holds either $\langle \mathcal{M}, s_k \rangle \models \psi$ or $\langle \mathcal{M}, s_i \rangle \models \varphi$ for some $i \in \{j, \ldots, k-1\}$
- $\langle \mathcal{M}, s_j \rangle \models \diamond\, \varphi$ iff $\langle \mathcal{M}, s_k \rangle \models \varphi$ for some $k \geq j$
- $\langle \mathcal{M}, s_j \rangle \models \Box\, \varphi$ iff $\langle \mathcal{M}, s_k \rangle \models \varphi$ for every $k \geq j$.

If $\langle \mathcal{M}, s_h \rangle \models \varphi$ then we say that $\varphi$ is true in the sate $s_h$ of the PLTL-structure $\mathcal{M}$.

Note that the truth of $\varphi \,\mathcal{U}\, \psi$ and $\diamond\, \psi$ in a state $s_j$ of a PLTL-structure $\mathcal{M}$ requires that $\psi$ must eventually be true in some state $s_k$ of $\mathcal{M}$ with $k \geq j$, and also that the eventual truth of $\neg\psi$ is required for $\neg\Box\, \psi$ and $\neg(\varphi \,\mathcal{R}\, \psi)$ to be true. Consequently

**Definition 2.2.1.** *An* eventuality *is a formula of the form $\varphi \,\mathcal{U}\, \psi$ or $\diamond\, \psi$ or $\neg\Box\, \psi$ or $\neg(\varphi \,\mathcal{R}\, \psi)$. In particular, formulas of the form $\varphi \,\mathcal{U}\, \psi$ are also called* until-formulas.

The semantics is extended from formulas to sets of formulas in the usual way: $\langle \mathcal{M}, s_j \rangle \models \Phi$ iff $\langle \mathcal{M}, s_j \rangle \models \gamma$ for all $\gamma \in \Phi$. We say that $\mathcal{M}$ is a model of $\Phi$, in symbols $\mathcal{M} \models \Phi$, iff $\langle \mathcal{M}, s_0 \rangle \models \Phi$. A satisfiable set of formulas has at least one model, otherwise it is unsatisfiable.

**Definition 2.2.2.** *Two sets of formulas $\Phi$ and $\Psi$ are* equisatisfiable *whenever $\Phi$ is satisfiable if and only if $\Psi$ is satisfiable.*

$$\begin{tikzcd}
e_0 \arrow[r, "R"] & e_1 \arrow[r, "R"] & e_2 \arrow[r, "R"] & \dots \arrow[r, "R"] & e_j \arrow[r, "R"] & \dots \arrow[r, "R"] & e_k \arrow[ll, bend right, "R"']
\end{tikzcd}$$

*Figure 2.1:* Cyclic sequence

The *logical consequence* relation between a set of formulas $\Phi$ and a formula $\chi$, denoted as $\Phi \models \chi$, is defined in the following way:

$$\begin{array}{lll} \Phi \models \chi & \text{iff} & \text{for every PLTL-structure } \mathcal{M} \text{ and every } s_j \in S_{\mathcal{M}}\text{:} \\ & & \text{if } \langle \mathcal{M}, s_j \rangle \models \Phi \text{ then } \langle \mathcal{M}, s_j \rangle \models \chi \end{array}$$

The above notion of logical consequence is usually called *local logical consequence*. There is a weaker notion called *global logical consequence* which demands $\chi$ to be true at all states in $\mathcal{M}$ if $\Phi$ is true at all states in $\mathcal{M}$. This latter notion is also interesting for many applications [48].

In order to construct models for satisfiable sets of formulas we use *cyclic* (also called *ultimately periodic*) PLTL-structures that we define in terms of either infinite paths over cycling sequences (Chapters 3 and 4) or infinite sequences (Chapter 5). Each element of such sequences is associated with a set of formulas. An infinite path (or infinite sequence) becomes the sequence of states of a PLTL-structure. The propositional variables that belong to the sets associated with the states define the map $V$. Finally we ensure that a PLTL-structure built in this way makes true, at each state $s_j$, the formulas associated with the state $s_j$.

Any infinite sequence $e_0, e_1, \dots, e_k, \dots$ involves an implicit successor relation, namely $R$, such that $(e_i, e_{i+1}) \in R$ for all $i \in I\!N$. When convenient, we write $e\,R\,e'$ to denote $(e, e') \in R$. A finite sequence gives also a corresponding implicit successor relation with a pair for each element except for the last one. A finite sequence $S = e_0, e_1, \dots, e_k$ is said to be *cyclic* iff its successor relation extends the implicit $R$ with a pair $(e_k, e_j)$ for some $j \in \{0, \dots, k\}$ (see Figure 2.1). Then, $e_j, \dots, e_k$ is called the *loop* of $S$, $e_j$ is called the *cycling element* of $S$, and the *path* over $S$ is the infinite sequence

$$\mathsf{path}(S) = e_0, e_1, \dots, e_{j-1} \cdot \langle e_j, e_{j+1}, \dots, e_k \rangle^{\omega}$$

where $\_ \cdot \_$ is the infix operator of concatenation of sequences and $U^{\omega}$ denotes the infinite sequence that results by concatenation of the sequence $U$ infinitely many times. Naturally, for any non-cyclic finite sequence $S$ we consider that $\mathsf{path}(S) = S$.

A PLTL-structure $\mathcal{M}$ is *cyclic* or *ultimately periodic* if its (infinite) sequence of states $S_{\mathcal{M}}$ is a path over a cyclic sequence of states.

Ensuring that a PLTL-structure constructed from an infinite sequence $S = e_0, e_1, \dots, e_k, \dots$ makes true the eventualities that appear in the sets associated to each $e_i$ in $S$ is the key step of the model construction process. In order to carry out this step, we define the notion of *fulfillment* of eventualities. We say that $e_i$ in $S$ fulfills an eventuality $\varphi\,\mathcal{U}\,\psi$ that belongs to the set associated with $e_i$, whenever there exists $e_h$ with $h \geq i$ such that $\psi$ belongs to the set associated with

$e_h$ and $\varphi$ belongs to the set associated with $e_g$ for every $g \in \{i, \ldots, h-1\}$. We particularize and precisely define the notion of fulfillment for every deduction system in the corresponding chapter.

## 2.3 Decidability of PLTL: Sound, Refutationally Complete and Complete Deduction Systems

It is well known that PLTL is a decidable logic (see e.g. [87]). Therefore, given a PLTL-formula $\psi$, there exists a procedure that is able to decide, in a finite amount of time, whether $\psi$ is satisfiable or unsatisfiable.

Whenever a new deduction system is proposed for a decidable logic, it must be assessed whether the deduction system yields a decision procedure. A finitary deduction system gives rise to a decision procedure whenever it is *complete*. A deduction system is complete if it is able to decide both satisfiability and unsatisfiability. The completeness of a deduction system is established by proving *soundness*, *termination* and *refutational completeness*.

Soundness means that a deduction system is correct in the sense that if a formula $\psi$ is classified as unsatisfiable by such deduction system, then $\psi$ is unsatisfiable. A deduction system is refutationally complete if whenever a formula $\psi$ is unsatisfiable, then the system classifies $\psi$ as unsatisfiable. However, soundness and refutational completeness do not guarantee that the satisfiability of a formula is decidable. That is, given a formula $\psi$, if $\psi$ is unsatisfiable, then a sound and refutationally complete deduction system will be able to classify $\psi$ as unsatisfiable, but if $\psi$ is satisfiable, then the deduction system may not terminate the derivation process, i.e. the deduction system may not give any answer. For that reason, termination is additionally required in order to have completeness, i.e, in order to decide both satisfiability and unsatisfiability. However, it is customary to use the term completeness to refer to refutational completeness in refutational systems where termination is not addressed. In each chapter of this thesis we precisely define the meaning of the term completeness for each deduction system.

The notion of deciding the satisfiability of a formula $\psi$ extends to a finite set of formulas $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ in a straightforward manner, since $\Phi$ is understood as $\varphi_1 \wedge \ldots \wedge \varphi_n$.

A logic is said to be *compact* when it verifies that, given any set of formulas $\Phi$, if every finite subset of $\Phi$ is satisfiable then $\Phi$ is satisfiable. It is well known that PLTL is a non-compact logic. For example, the infinite set of formulas $\Psi = \{\circ^i p \mid i \in \mathbb{N}\} \cup \{\diamond\neg p\}$ is not satisfiable but every finite subset of $\Psi$ is satisfiable. As a consequence, any complete deduction system that is able to deal with infinite sets of formulas should be infinitary. However infinitary systems do not yield decision procedures that are able to decide the satisfiability or unsatisfiability of a formula in a finite amount of time. Since we are interested in finitary deduction systems, we only deal with finite sets of formulas.

## 2.4 Invariant Formulas in PLTL

One of the features of PLTL (and temporal logic in general) is the ability to express eventuality properties and invariant properties. Eventuality properties state that a formula will eventually become true. Eventuality properties are directly expressed by means of specific connectives (e.g. $\mathcal{U}$ and $\diamond$) that give rise to the so-called eventualities (see Definition 2.2.1), which are trivially detectable ($\varphi \mathcal{U} \psi$, $\diamond \psi$, etc). Invariant properties state that a formula is always true (from some moment onwards). Invariant properties are expressed by sets of formulas that, often, are not trivially detectable. If the set of formulas $\Phi$ expresses and invariant property, then we say that the formula $\bigwedge \Phi$ is an *invariant formula* (*invariant*, in short). Moreover, $\Phi$ could also be a subset of another set of formulas, hence we say that, usually, invariants are "hidden" in a set of formulas. Formally, a formula $\chi$ is an invariant if and only if the formula $\neg\chi \vee \circ\chi$ is true at every state of every PLTL-structure. Since "hidden" invariants can prevent the fulfillment of eventualities, the key issue in the finitary deduction systems for PLTL (and temporal logic in general) is to deal with eventualities and invariants.

In order to illustrate the concept of invariant, let us consider the following three sets of formulas

$$\begin{aligned}
\Delta_1 &= \{\Box(\neg\varphi_0 \vee \circ\psi_0), \ldots, \Box(\neg\varphi_n \vee \circ\psi_n)\} \\
\Delta_2 &= \{\Box(\neg\psi_0 \vee \neg\gamma), \ldots, \Box(\neg\psi_n \vee \neg\gamma)\} \\
\Delta_3 &= \{\Box(\neg\psi_0 \vee \varphi_0 \vee \ldots \vee \varphi_n), \ldots, \Box(\neg\psi_n \vee \varphi_0 \vee \ldots \vee \varphi_n)\}
\end{aligned}$$

The formulas $\circ\Box\neg\gamma$ and $\Box(\varphi_0 \vee \ldots \vee \varphi_n)$ are logical consequences of the set

$$\Sigma = \{\varphi_0 \vee \ldots \vee \varphi_n\} \cup \Delta_1 \cup \Delta_2 \cup \Delta_3$$

Additionally, for the formula $\chi = \bigwedge \Sigma$, it holds that $\neg\chi \vee \circ\chi$ is true in every state of every PLTL-structure. Therefore $\chi$ is an invariant that states, in an intricate way, that the eventuality $\Diamond\gamma$ cannot be true from the next state onwards. Note also that the formula $\Box(\neg(\varphi_0 \vee \ldots \vee \varphi_n) \vee \circ(\varphi_0 \vee \ldots \vee \varphi_n))$ is a logical consequence of $\Delta_1 \cup \Delta_2 \cup \Delta_3$. So that, if we restrict ourselves to the set of models of $\Delta_1 \cup \Delta_2 \cup \Delta_3$, we could say that the formula $\varphi_0 \vee \ldots \vee \varphi_n$ is an invariant with respect to such models.

Since the set $\Sigma$ can be formed by an arbitrary number of formulas, the invariant $\chi$ (unlike eventualities) cannot be trivially detected. Additionally, $\Sigma$ could just be a subset of another set of formulas.

More details about invariants can be found in e.g. [45, 104, 105, 106].

Given a set of formulas $\Psi$ and an eventuality $\Diamond\gamma$, the crucial element for every refutationally complete finitary deduction system for PLTL is to detect whether $\Psi$ contains an invariant that prevents the fulfillment of $\gamma$.

# 3. DUAL SYSTEMS OF TABLEAUX AND SEQUENTS FOR PLTL

## *3.1 Introduction*

Tableau systems are refutational proof methods that play a prominent role in the development of automated reasoning for temporal logic (and many other logics). In addition, in the case of decidable logics, such as PLTL, tableau methods serve as decision procedures for the satisfiability of (sets of) formulas. The first tableau method for PLTL was introduced by P. Wolper in [128] and it is a *two-pass method*. In the first pass, it generates an auxiliary graph by applying the tableau rules. This graph is checked and possibly pruned in a second pass that analyzes whether the eventualities are fulfilled. As stated in Definition 2.2.1, an eventuality is a formula that asserts that something does eventually hold. For example, for a path in the graph to fulfill $\varphi \mathcal{U} \psi$, the formula $\psi$ must eventually appear in the path. Hence, any maximal strongly connected component in the graph that contains $\varphi \mathcal{U} \psi$ in the label of one of its nodes, but does not contain $\psi$ in the label of any of its nodes and from which no other maximal strongly connected component can be reached, is pruned. At the end, an empty graph means unsatisfiability. Since Wolper's seminal paper [128], several authors (e.g. [73, 8, 87, 79, 81]) have proposed and studied tableau methods for different temporal and modal logics inspired by Wolper's tableau (see [71] for a good survey). In addition, Wolper's two-pass tableau has been used in the development of decision procedures or proof techniques for logics that extend PLTL to some decidable fragment of the first-order temporal logic (e.g.[84]), or to the branching case or with other features, such as agents, knowledge, etc (e.g. [70]). In the case of two-pass tableau methods the auxiliary graph and the second pass prevent the association of a sequent calculus proof to each tableau refutation.

Sequent calculi provide a general deductive setting that uniformly embeds refutational methods and other deduction techniques such as goal-directed proofs or natural deduction. Traditional sequent calculi for temporal logic (e.g. [104, 105, 121]) usually include some inference rules that complicate the automation of temporal deduction. In particular, temporal sequent calculi either need some form of cut (classical cut or invariant-based cut) or they include infinitary rules. Cut rules imply the "invention" of lemmata, called cut formulas, for their application. Invariant formulas are particular cut formulas for proving temporal eventualities. In [104] and [121], two sequent calculi for temporal logic with invariant-based rules are presented. In fact, in both approaches, a system that includes also a cut rule is presented and then a cut elimination proof is provided. However, invariant-based rules for temporal connectives cannot be avoided. In [105] various sequent calculi are presented for temporal logic without the until connective $\mathcal{U}$ (this means that the considered logic has a limited temporal expressive power). In [105] completeness and cut-elimination proofs, together with various interesting reductions among various calculi are provided. However, every calculus includes either some infinitary rule or some invariant-based rule.

A remarkable consequence of using auxiliary graphs that require a second pass in the tableau framework and either invariant-based rules or infinitary rules in the sequent framework is that temporal logic fails to carry out the classical correspondence between tableaux and sequents. In classical logic, and even in some non-classical logics (e.g. many-valued logics), each step in a tableau construction corresponds to an inference in the sequent calculus. Therefore, there is an easy, useful and well-known correspondence that associates with each closed tableau a sequent proof, which is a refutation.

In this chapter, we present a tableau system together with a dual cut-free and invariant-free finitary sequent calculus for PLTL. We first provide a Temporal Tableau Method, called TTM, which does not require auxiliary graphs to decide if a set of formulas is satisfiable. Instead, there is a tableau rule that prevents from indefinitely delaying the fulfillment of eventualities. The tableau method TTM is sound, refutationally complete and also complete. Therefore, it gives rise to a decision procedure for PLTL. The tableau method TTM is directly associated with a one-sided (or Tait style) sequent calculus that we call TTC (from Tait-style Temporal Calculus). Since TTM is free from all the structural rules that hinder the mechanization of deduction, e.g. weakening and contraction, then the resulting sequent calculus TTC is also free from this kind of structural rules. In particular, TTC is free from any kind of cut, including invariant-based cut. From the deduction system TTC, we obtain the two-sided sequent calculus GTC (from Gentzen-style Temporal Calculus) that preserves all these good freeness properties and is finitary, sound and complete for PLTL. Therefore, we show that the classical correspondence between tableaux and sequent calculi can be extended to temporal logic. Such correspondence is mainly enabled by a new style of inference rule for eventualities which introduces a new kind of temporal deduction. This new kind of temporal deduction is based on the fact that if a set of formulas $\Delta \cup \{\varphi \mathcal{U} \psi\}$ is satisfiable, then it must exist a model $\mathcal{M}$ (with states $s_0, s_1, \ldots$) that is minimal in the following sense:

$$\mathcal{M} \text{ satisfies either } \Delta \cup \{\psi\} \text{ or } \Delta \cup \{\varphi, \circ((\varphi \wedge \neg\Delta)\, \mathcal{U}\, \psi)\}$$

where $\Delta = \{\varphi_1, \ldots, \varphi_n\}$ and $\neg\Delta = \neg(\varphi_1 \wedge \ldots \wedge \varphi_n)$. In other words, in a minimal model $\mathcal{M}$ of $\Delta \cup \{\varphi \mathcal{U} \psi\}$, if $\psi$ is not true in $s_0$ then the so-called *context* $\Delta$ cannot be true from $s_1$ until the first state where $\psi$ is true. In order to clarify this fact, let us consider a model $\mathcal{M}'$ with states $s'_0, s'_1, \ldots$ such that $s'_j$ (with $j \geq 2$) is the first state in which $\psi$ is true and there is at least one state in the sequence $s'_1, \ldots, s'_j$ in which $\Delta$ is true. Now, let $k$ be the greatest $z \in \{1, \ldots, j\}$ such that $\Delta$ is true in $s'_z$. Then, the structure given by $s'_k, s'_{k+1}, \ldots$ is also a model of $\Delta \cup \{\varphi \mathcal{U} \psi\}$ that is minimal in the above sense.

The tableau method TTM and the sequent calculi TTC and GTC (first presented in [61]) extend and improve the work introduced in [60, 58].

In addition to the traditional approaches to tableau and sequent systems for temporal logic mentioned above, there are two approaches whose results are closely related to ours. On one hand, in [117] a one-pass tableau calculus that produces cyclic trees is introduced by Schwendimann. This tableau calculus avoids the second pass by adding extra information to the nodes in the tableau. Some of this information must be synthesized bottom-up and it is needed because tableau branches are not independent from each other. In particular, a cyclic branch may contain an unfulfilled eventuality that can be fulfilled if other accessible cyclic branches are considered for generating a wider cycle. Hence, it carries out an on-the-fly checking of the fulfillment of every eventuality in every branch. Our method is not based on an on-the-fly checking of eventualities. As mentioned above, in our tableau method TTM, there is a tableau rule that prevents from indefinitely delaying the fulfillment of eventualities. In TTM branches are independent from each other and the fulfillment of an eventuality that appears in a branch does not depend on other branches. Additionally, TTM tableaux are classical-like in the sense that unsatisfiable sets of formulas give rise to closed tableaux where every leaf contains either a formula and its negation or the constant F. By contrast, Schwendimann's approach does not yield classical-like tableaux in the sense that unsatisfiable sets of formulas may produce non-fulfilling cyclic branches whose last nodes do not contain an explicit inconsistency (i.e. a formula and its negation). Consequently, such approach requires an extra process for deciding whether a cyclic branch is fulfilling or not. Schwendimann's approach has also been applied to other logics such as e.g. CTL ([5]) and PDL ([72]).

On the other hand, at the time of the publication of [60], to our knowledge the first published invariant-free finitary sequent calculus for PLTL, we learned about the work of K. Brünnler and M. Lange (see [20]), which provides an interesting alternative approach to the proof theory of PLTL. The calculus presented in [20] has the analytic superformula property. Actually, in [20], the strategy that leads to prove the completeness of the sequent system –which lies in fairly selecting exactly one eventuality and sticking to it until it is fulfilled– is incorporated in the sequent system by means of the so-called annotated formulas (which do not belong to the logical language). The completeness proof of our system is also based on the mentioned strategy but such a strategy is not incorporated in the system. In this way different strategies can be used. We differentiate between the systematic derivation (which guarantees completeness) and the many other derivations that usually are feasible. In Section 3.7 we compare, in a more detailed way, our approach with the above mentioned approaches.

*Outline of the chapter.* In Section 3.2 we introduce the notions of sequent and sequent system and we point out the relationship between tableau systems and sequent systems. In Section 3.3 we present the tableau system TTM. Subsection 3.3.1 introduces the basic tableau structure. Subsection 3.3.2 provides the rule system. Subsection 3.3.3 contains the definitions of inconsistent node and open and closed branches. In Subsection 3.3.4 we establish the notion of TTM tableau which includes the key concepts of expanded branch and expanded tableau. Finally, in Subsection 3.3.5 we show some examples of tableaux. Section 3.4 is devoted to the soundness and completeness results. The soundness of TTM is proved in Subsection 3.4.1. In Subsection 3.4.2 we propose an algorithm for systematically obtaining, for any set of formulas $\Phi$, a finite tableau that proves that $\Phi$ is either satisfiable or unsatisfiable. In particular we provide the termination result and the worst case complexity for the algorithm. Examples that illustrate the application of the systematic tableau algorithm are showed in Subsection 3.4.3. In Subsection 3.4.4 we prove the completeness of TTM. In Subsection 3.4.5 we suggest some improvements. In Sections 3.5 and 3.6, we introduce, respectively, the one-sided sequent system TTC and the two-sided sequent system GTC. The rule system, the soundness and completeness results and some illustrative examples are provided for each of these two sequent systems. Finally, in Section 3.7 we deal with related work and we compare some features of our approach with other approaches.

## 3.2 Sequent-based Deduction Systems and Tableaux

Sequent calculus, first introduced by Gentzen ([65]), is the most elegant and flexible system for writing proofs. Each line of a sequent calculus proof is a sequent. A *sequent* was (originally) formed by two sequences of formulas separated by some kind of arrow (for instance, $\vdash$). The intended meaning of a sequent $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi_1, \psi_2, \ldots, \psi_m$ is the formula

$$\bigwedge_{i=1}^{n} \varphi_i \rightarrow \bigvee_{i=1}^{m} \psi_i$$

where $\rightarrow$ is the classical connective of implication (i.e. $\chi \rightarrow \gamma \equiv \neg\chi \vee \gamma$). The sequence $\varphi_1, \varphi_2, \ldots, \varphi_n$ is called the *antecedent* of the above sequent and the sequence $\psi_1, \psi_2, \ldots, \psi_m$ is called its *consequent* (or succedent). Since the seminal work of Gentzen, many variations of the notion of sequent have been explored to provide different sequent-based deduction systems. A sequent calculus is a proof system given by a set of rules such that each rule indicates that a sequent may be inferred from a set of sequents. That is, a (finitary) rule consists of a *numerator* formed by a (finite) set of sequents $S_1, \ldots, S_n$ and a *denominator* $S$ separated by a horizontal line, next to which is the name of the rule[1]:

$$(r)\ \frac{S_1, \ldots, S_n}{S}$$

In a rule $(r)$ as above, each sequent $S_i$ is called a *premise* and $S$ is the *conclusion*. Traditionally, a sequent calculus consists of structural rules and connective rules (rules for the connectives). The conclusion of a connective rule has a *principal formula* that is affected by the inference. For example

$$(\wedge L)\ \frac{\Delta, \varphi, \psi \vdash \chi}{\Delta, \varphi \wedge \psi \vdash \chi}$$

is a rule for conjunction ($\wedge$) whose principal formula is $\varphi \wedge \psi$. However, in structural rules, the inference is guided by the whole conclusion. An example of structural rule is classical weakeaning

$$(Wk)\ \frac{\Delta \vdash \chi}{\Delta, \Delta' \vdash \chi}$$

There are many variations of sequents. The simplest one is obtained by allowing the antecedent and consequent to be a (multi)set instead of a sequence. This choice (of sequences, multisets or sets) is directly related to the classical structural rules of exchange and contraction. In particular, the exchange rule only makes sense in sequence-based sequent calculi, whereas the contraction rule, which is well-founded for sequences and multisets, leads to some confusion when sets are considered. More precisely, the classical contraction rule (on the left):

$$\frac{\Delta, \varphi, \varphi \vdash \chi}{\Delta, \varphi \vdash \chi}$$

makes no sense when the antecedent is a set, however some legal application of connective rules could hide a contraction. For example, the inference

$$\frac{\varphi \wedge \psi, \varphi, \psi \vdash \chi}{\varphi \wedge \psi \vdash \chi}$$

[1] Sometimes, due to space reasons, the rule is formatted as follows:

$$\begin{array}{c} S_1 \\ \vdots \\ S_n \\ \hline S \end{array}$$

could result from a legal application of the above rule $(\wedge L)$ for $\Delta = \{\varphi \wedge \psi\}$. In classical logic this kind of hidden use of the contraction does not harm, however in temporal logic[2] we must be more careful on this matter. The sequent systems we are going to introduce are based on sets. The notation $\Delta, \varphi$ stands for $\Delta \cup \{\varphi\}$ where $\varphi \notin \Delta$. This convention clearly disallows hidden contraction. In particular, it disallows the above inference that uses the rule $(\wedge L)$ for $\Delta = \{\varphi \wedge \psi\}$.

Another simple variation of sequent is related to the cardinality of the consequent. That is, sequents can be either multiple-conclusioned or single-conclusioned, or even one-sided, respectively depending on whether the consequent is a set, a singleton or empty.[3] One-sided sequents were first used by Schütte [116] with multisets and by Tait [123] with sets, hence when a new system is presented it is usual to point out whether it is a Gentzen-Schütte style calculus or whether it is a Tait style calculus. There are really two kinds of one-sided sequents: left-handed (empty consequent) and right-handed (empty antecedent). In this thesis, we use left-handed sequents because they are very close to tableau systems. In fact, we present the tableau system TTM that is directly related to the left-handed sequent calculus TTC. Besides, the established results for the calculus TTC can be easily extended to the two-sided sequent calculus GTC. We have preferred to formulate the calculus GTC by means of single-conclusioned sequents, instead of multiple-conclusioned sequents, because in our opinion single-conclusioned sequents are closer to natural deduction and capture better our intuition in logical reasoning. A multiple-conclusioned system can be easily obtained from GTC.

## 3.3 The Tableau Method TTM

In this section we present a tableau system, called TTM, for PLTL. In TTM, tableaux are essentially trees but branches can end in a leave that represents a loop into another node in its branch. Our tableaux are one-pass in the sense of [117], that is, they do not require a second pass to check an auxiliary graph of states in order to determine if every eventuality is fulfilled. As a consequence, temporal stages are represented inside the branches of the tableaux instead of in an auxiliary graph. The contents of this section are divided into five subsections. In Subsection 3.3.1 we introduce preliminary concepts related to the tableau structure. In Subsections 3.3.2, 3.3.3 and 3.3.4 we present the rules for constructing tableaux, the notion of inconsistency in nodes and the notion of tableau itself, respectively. Finally, in 3.3.5 we provide some detailed examples of tableaux.

### 3.3.1 Pre-tableaux

A tableau $\mathcal{T}_\Phi$ for a finite set of formulas $\Phi$ is a tree-like structure where each node $n$ is labelled with a set of formulas $L(n)$. The root is labelled with the set $\Phi$ whose satisfiability we wish to check. The children of a node $n$ are obtained by applying one of the rules to one of the formulas in $L(n)$. Nodes are organized in branches, so that the rules serve to either enlarge the branch (with one new child) or split the branch with two new children. In order to formalize the notion of branch we recall the concept of strongly generated set.

[2] In general, in modal logic.

[3] There are more sophisticated variants of sequents that are obtained, for example, by adding structure or labels into sequents, but they are out of the scope of this thesis.

**Definition 3.3.1.** *Let* $\mathsf{Nodes}$ *be a finite non-empty set of nodes,* $n$ *a node in* $\mathsf{Nodes}$ *and* $\mathsf{Nodes}^+$ *the set of all non-empty sequences of elements in* $\mathsf{Nodes}$*. A non-empty set* $B \subseteq \mathsf{Nodes}^+$ *is* strongly generated *with respect to* $\mathsf{Nodes}$ *and* $n$ *iff it verifies the following conditions:*

1. *If* $n_0, n_1, \ldots, n_k \in B$*, then* $n_i \neq n_j$ *for all* $i$ *and* $j$ *such that* $0 \leq i < j \leq k$
2. *If* $n_0, n_1, \ldots, n_k \in B$*, then* $n_0 = n$
3. *If* $n_0, n_1, \ldots, n_k \in B$*, then* $n_0, n_1, \ldots, n_i \in B$ *for all* $i \in \{0, \ldots, k-1\}$
4. *For every node* $m \in \mathsf{Nodes}$ *there is a unique sequence* $n_0, n_1, \ldots, n_k \in B$ *such that* $n_k = m$*.*

We denote by $\mathsf{trees}(\mathsf{Nodes}, n)$ the collection of all subsets of $\mathsf{Nodes}^+$ that are strongly generated with respect to $\mathsf{Nodes}$ and $n$. Let $B \in \mathsf{trees}(\mathsf{Nodes}, n)$, each sequence $b \in B$ is called a branch. A branch $b' = n_0, n_1, \ldots, n_i$ is a *prefix* of another branch $b = n_0, n_1, \ldots, n_k$ if $0 \leq i \leq k$. If, besides, $i \neq k$, we say that $b'$ is a *proper prefix* of $b$. A branch $b \in B$ is *maximal* whenever $b$ is not proper prefix of any other branch in $B$.

Note that, in the above Definition 3.3.1, condition 1 means that a node cannot appear more than once in a branch, condition 2 means that the first element in every branch is the node $n$, condition 3 means that a strongly generated set is closed with respect to non-empty prefixes and condition 4 states that every node must be the last node of exactly one branch, which may not be maximal. Note also that $\mathsf{trees}(\mathsf{Nodes}, n)$ is finite and every sequence $b \in B$ is finite for any $B \in \mathsf{trees}(\mathsf{Nodes}, n)$.

Now we define the concept of pre-tableau for a set of formulas.

**Definition 3.3.2. (Pre-tableau)** *A* pre-tableau *for a finite set of formulas* $\Phi$ *is a tuple* $\mathcal{T}_\Phi = (\mathsf{Nodes}, n_\Phi, L, B, R)$ *such that:*

1. $\mathsf{Nodes}$ *is a finite non-empty set of nodes*
2. $n_\Phi$ *is a node in* $\mathsf{Nodes}$*, called initial node*
3. $L : \mathsf{Nodes} \to 2^\Gamma$ *is the labelling function where* $\Gamma$ *is a set of formulas that contains* $\Phi$ *such that the initial node is labelled by* $\Phi$*, that is* $L(n_\Phi) = \Phi$
4. $B$ *is a strongly generated set in* $\mathsf{trees}(\mathsf{Nodes}, n_\Phi)$*, called the set of branches*
5. $R$ *is the successor relation over* $\mathsf{Nodes}$*.* $R$ *should be coherent with* $B$ *in the sense that for all* $n, n' \in \mathsf{Nodes}$*,* $(n, n') \in R$ *iff there exists a sequence* $n_0, n_1, \ldots, n_k \in B$ *such that* $n = n_i$ *and* $n' = n_{i+1}$ *for some* $i \in \{0, \ldots, k-1\}$*.*

As usual, $R^+$ and $R^*$ respectively denote the transitive closure and the reflexive-transitive closure of any binary relation $R$.

### 3.3.2 Tableau Rules

A tableau rule is applied to a set of formulas $L(n)$ labelling a node $n$ (which is the last node of a branch). Each rule application requires a previous choice of a formula from $L(n)$. We call the set $L(n) \setminus \{\varphi\}$, where $\varphi$ is the chosen formula, the *context* and it is denoted by $\Delta$.

| Rule | $\alpha$ | $A(\alpha)$ |
|---|---|---|
| $(\neg\neg)$ | $\neg\neg\varphi$ | $\{\varphi\}$ |
| $(\wedge)$ | $\varphi \wedge \psi$ | $\{\varphi, \psi\}$ |
| $(\neg\circ)$ | $\neg\circ\varphi$ | $\{\circ\neg\varphi\}$ |

| Rule | $\beta$ | $B_1(\beta)$ | $B_2(\beta)$ |
|---|---|---|---|
| $(\neg\wedge)$ | $\neg(\varphi \wedge \psi)$ | $\{\neg\varphi\}$ | $\{\neg\psi\}$ |
| $(\neg\mathcal{U})$ | $\neg(\varphi\,\mathcal{U}\,\psi)$ | $\{\neg\varphi, \neg\psi\}$ | $\{\varphi, \neg\psi, \neg\circ(\varphi\,\mathcal{U}\,\psi)\}$ |
| $(\mathcal{U})_1$ | $\varphi\,\mathcal{U}\,\psi$ | $\{\psi\}$ | $\{\varphi, \neg\psi, \circ(\varphi\,\mathcal{U}\,\psi)\}$ |

| Rule | $\beta$ | $B_1(\beta)$ | $B_2(\beta, \Delta)$ |
|---|---|---|---|
| $(\mathcal{U})_2$ | $\varphi\,\mathcal{U}\,\psi$ | $\{\psi\}$ | $\{\varphi, \neg\psi, \circ((\varphi \wedge \neg\Delta)\,\mathcal{U}\,\psi)\}$ |

where $\Delta$ stands for the context

*Figure 3.1:* Primitive TTM-Rules

As usual, the TTM-rules are based on a classification of the formulas into conjunctive and disjunctive, which are respectively named as $\alpha$-formulas and $\beta$-formulas. In Figure 3.1, any $\alpha$-formula $\alpha$ is decomposed in a unique set, called $A(\alpha)$, and any $\beta$-formula $\beta$ is decomposed into two constituent sets $B_1$ and $B_2$. The set $B_1$ depends on the considered formula $\beta$, whereas the set $B_2$ can also depend on the context $\Delta$. [4]

This classification gives raise to the tableau rules whose names are also given in Figure 3.1. Every rule, except $(\mathcal{U})_2$, is well known in the literature. It is worth noting that $(\mathcal{U})_1$ and $(\mathcal{U})_2$ affect the same $\beta$-formula, but not in the same way. The rule $(\mathcal{U})_2$ can be considered quite peculiar, since $B_2(\beta, \Delta)$ includes a formula which depends on the whole set of formulas in the node. Moreover, $(\mathcal{U})_2$ leads to a new tableau construction style that allows us to dispense with the auxiliary graph. This rule is based on the fact that if a formula $\varphi\,\mathcal{U}\,\psi$ is satisfiable in a given context $\Delta$, it is because there exists a model for $\Delta \cup \{\varphi\,\mathcal{U}\,\psi\}$, with sates $s_0, s_1, \ldots$, that is minimal in the sense that if $s_j$ (with $j \geq 0$) is the first state in which $\psi$ is true then $\Delta$ is not true in the states that belong to the sequence $s_1, s_2, \ldots, s_{j-1}$. More precisely, the crucial idea behind the rule $(\mathcal{U})_2$ is based on the following equisatisfiability result that relates two eventualities.

**Proposition 3.3.3.** *Let $\Delta$ be a set of formulas. $\Sigma_1 = \Delta \cup \{\varphi\,\mathcal{U}\,\psi\}$ and $\Sigma_2 = \Delta \cup \{\psi \vee (\varphi \wedge (\neg\psi) \wedge \circ((\varphi \wedge \neg\Delta)\,\mathcal{U}\,\psi))\}$, where $\varphi\,\mathcal{U}\,\psi \notin \Delta$, are equisatisfiable.*

*Proof.* In order to show that $\Sigma_1$ and $\Sigma_2$ are equisatisfiable, let us suppose that $\mathcal{M}$ is a model of $\Sigma_1$. If $\langle \mathcal{M}, s_0 \rangle \models \Delta \cup \{\psi\}$, then $\mathcal{M}$ is also a model of $\Sigma_2$. Otherwise, $\langle \mathcal{M}, s_0 \rangle \models \Delta \cup \{\varphi, \neg\psi, \circ(\varphi\,\mathcal{U}\,\psi)\}$ and there exists a state $s_j$ with $j \geq 1$ such that $\langle \mathcal{M}, s_j \rangle \models \psi$ and $\langle \mathcal{M}, s_i \rangle \models \varphi$ for every $i \in \{0, \ldots, j-1\}$. Let $k$ be the greatest $h$ such that $0 \leq h < j$ and $\langle \mathcal{M}, s_h \rangle \models \Delta \cup \{\varphi\,\mathcal{U}\,\psi\}$. We can ensure the existence of $k$ because at least $\langle \mathcal{M}, s_0 \rangle \models \Delta \cup \{\varphi\,\mathcal{U}\,\psi\}$. As a consequence of the choice of $k$, it holds that $\langle \mathcal{M}, s_k \rangle \models \{\varphi, \neg\psi, \circ((\varphi \wedge \neg\Delta)\,\mathcal{U}\,\psi)\}$. Then, the PLTL-structure $\mathcal{M}' = (S_{\mathcal{M}'}, V_{\mathcal{M}'})$ such that $S_{\mathcal{M}'} = s'_0, s'_1, \ldots$ and $s'_g =$

[4] Remember that $\Delta$ is always assumed to be a finite set and that $\neg\Delta$ is F whenever $\Delta$ is empty.

| Rule | $\alpha$ | $A(\alpha)$ |
|---|---|---|
| $(\Box)$ | $\Box\varphi$ | $\{\varphi, \circ\Box\varphi\}$ |
| $(\neg\Diamond)$ | $\neg\Diamond\varphi$ | $\{\neg\varphi, \circ\neg\Diamond\varphi\}$ |

| Rule | $\beta$ | $B_1(\beta)$ | $B_2(\beta)$ |
|---|---|---|---|
| $(\mathcal{R})$ | $\varphi\mathcal{R}\psi$ | $\{\varphi,\psi\}$ | $\{\neg\varphi,\psi,\circ(\varphi\mathcal{R}\psi)\}$ |
| $(\Diamond)_1$ | $\Diamond\varphi$ | $\{\varphi\}$ | $\{\neg\varphi, \circ\Diamond\varphi\}$ |
| $(\neg\Box)_1$ | $\neg\Box\varphi$ | $\{\neg\varphi\}$ | $\{\varphi, \circ\neg\Box\varphi\}$ |
| $(\neg\mathcal{R})_1$ | $\neg(\varphi\mathcal{R}\psi)$ | $\{\neg\psi\}$ | $\{\neg\varphi,\psi,\circ\neg(\varphi\mathcal{R}\psi)\}$ |

| Rule | $\beta$ | $B_1(\beta)$ | $B_2(\beta,\Delta)$ |
|---|---|---|---|
| $(\Diamond)_2$ | $\Diamond\varphi$ | $\{\varphi\}$ | $\{\neg\varphi, \circ((\neg\Delta)\mathcal{U}\varphi)\}$ |
| $(\neg\Box)_2$ | $\neg\Box\varphi$ | $\{\neg\varphi\}$ | $\{\varphi, \circ((\neg\Delta)\mathcal{U}\neg\varphi)\}$ |
| $(\neg\mathcal{R})_2$ | $\neg(\varphi\mathcal{R}\psi)$ | $\{\neg\psi\}$ | $\{\neg\varphi,\psi,\circ(((\neg\varphi)\wedge\neg\Delta)\mathcal{U}\neg\psi)\}$ |

where $\Delta$ stands for the context

*Figure 3.2:* Some Derived TTM-Rules

$s_{k+g}$ and $V_{\mathcal{M}'}(s'_g) = V_{\mathcal{M}}(s_{k+g})$ for every $g \geq 0$ is a model of $\Delta \cup \{\varphi, \neg\psi, \circ((\varphi\wedge\neg\Delta)\mathcal{U}\psi)\}$. Hence, $\mathcal{M}' \models \Sigma_2$. In the converse direction, any model of $\Sigma_2$ is itself a model of $\Sigma_1$. ■

The above property is applied to the tableau construction by means of the rule $(\mathcal{U})_2$. The proof of Proposition 3.3.3 reflects the intuition behind the rule $(\mathcal{U})_2$. In fact, Proposition 3.3.3 is used in Lemma 3.4.1 to prove the correctness of the rule $(\mathcal{U})_2$. The use of the rule $(\mathcal{U})_2$ makes possible to prevent the repetition of contexts (i.e. sets of formulas) from the node $n$ in which $(\mathcal{U})_2$ is applied to an eventuality $\varphi\mathcal{U}\psi \in L(n)$ until the first node $n'$ for which $\psi \in L(n')$, provided that the number of possible contexts is finite. Consequently, the rule $(\mathcal{U})_2$ makes possible not to allow the indefinite postponement of the presence of $\psi$ (i.e. the fulfillment of $\varphi\mathcal{U}\psi$) in the sequence of nodes obtained from $n$, provided that the number of possible contexts is finite.

One may wonder whether the rule $(\mathcal{U})_1$ is essential for completeness. Our completeness proof uses it, but it is an open problem whether there exists an alternative proof disregarding the rule $(\mathcal{U})_1$. However, we conjecture that $(\mathcal{U})_1$ is essential for completeness. Anyway, from a practical point of view it is better that the system includes the rule $(\mathcal{U})_1$, since $(\mathcal{U})_2$ is costly to use.

Besides the above primitive TTM-rules, the method TTM also uses the operator unnext to convert the labelling set $L(n)$ of a node $n$ into another set $\mathsf{unnext}(L(n))$ that labels a new node and that intuitively represents the jump from one time instant to the next one.

**Definition 3.3.4.** *For any set of formulas $\Psi$:*

$$\mathsf{unnext}(\Psi) = \{\gamma \mid \circ\gamma \in \Psi\}$$

Note that, $\mathsf{unnext}(\Psi)$ could be the empty set, which we denote by $\emptyset$.

From the primitive TTM-rules we can derive rules for the defined connectives like the ones in Figure 3.2. However, along the chapter, most technical details are given only for the primitive rules, in particular for the rule $(\mathcal{U})_2$.

### *3.3.3 Consistent and Inconsistent Nodes and Closed and Open Branches*

Tableaux are constructed with the aim of refuting the initial set of formulas. The search for a refutation is carried out by decomposing formulas into their constituent sets of formulas in order to find out whether an inconsistent set of formulas can be obtained.

**Definition 3.3.5.** *A node $n$ is consistent iff* $\textsc{f} \notin L(n)$ *and there is no* $\varphi$ *such that* $\{\varphi, \neg\varphi\} \subseteq L(n)$. *Otherwise,* $n$ *is* inconsistent.

Note that, in Definition 3.3.5, the formula $\varphi$ is not required to be an atom. Indeed, by demanding $\varphi$ to be atomic the completeness of TTM would be lost. For example, the set of formulas $\Psi = \{p\,\mathcal{U}\,q, \neg(p\,\mathcal{U}\,q)\}$ would not be refutable, if the label $L(n)$ of an inconsistent node $n$ should satisfy $\textsc{f} \in L(n)$ or $\{\gamma, \neg\gamma\} \subseteq L(n)$ such that $\gamma \in \mathsf{Prop}$. In fact, using the tableau rules there is no way to achieve such atomic inconsistency. However, $\Psi$ must be inconsistent in order to achieve completeness. It is also worth noting that a node labelled by $\Sigma = \{p\,\mathcal{U}\,q, (\neg p)\,\mathcal{R}\,(\neg q)\}$ (which is equivalent to $\Psi$) is not inconsistent (in the sense of Definition 3.3.5). The set of formulas $\Sigma$ can be refuted by our tableau method, but using the (non-atomic) inconsistency of $\{\circ((\neg p)\,\mathcal{R}\,(\neg q)), \neg\circ((\neg p)\,\mathcal{R}\,(\neg q))\}$.

When a branch $b$ contains an inconsistent node we say that $b$ is *closed*. Any closed branch is trivially unsatisfiable. Branches that are not closed are said to be *open*. However, open branches are not necessarily satisfiable. In particular, an open branch could be a prefix of a closed one.

### *3.3.4 Semantic Tableaux*

The tableau rules given in Subsection 3.3.2, together with the notion of consistent node (Definition 3.3.5), allow us to determine when a pre-tableau is a tableau. Along this subsection $\mathcal{T}_\Phi$ stands for a pre-tableau for $\Phi$ given by a tuple $(\mathsf{Nodes}, n_\Phi, L, B, R)$.

**Definition 3.3.6. (Coherent pre-tableau)** *A pre-tableau $\mathcal{T}_\Phi$ is* coherent *if and only if every node $n$ in a non-maximal branch in $B$ is consistent and exactly one of the following items holds for every branch $b = n_0, n_1, \ldots, n_i, n_{i+1}, \ldots, n_k \in B$ and every $i \in \{0, \ldots, k-1\}$:*

*(1)* $L(n_{i+1}) = A(\alpha) \cup L(n_i) \setminus \{\alpha\}$ *for some* $\alpha \in L(n_i)$

*(2) There exist exactly one node $n' \in N \setminus \{n_{i+1}\}$ and one branch $b' = n_0, n_1, \ldots, n_i, n' \in B$ such that for some $\beta \in L(n_i)$ either*

- $L(n_{i+1}) = B_1(\beta) \cup L(n_i) \setminus \{\beta\}$ *and* $L(n') = C(\beta, L(n_i)) \cup L(n_i) \setminus \{\beta\}$ *or*
- $L(n_{i+1}) = C(\beta, L(n_i)) \cup L(n_i) \setminus \{\beta\}$ *and* $L(n') = B_1(\beta) \cup L(n_i) \setminus \{\beta\}$

*where $C(\beta, L(n_i))$ is $B_2(\beta)$ or $B_2(\beta, L(n_i) \setminus \{\beta\})$.*

*(3)* $L(n_{i+1}) = \mathsf{unnext}(L(n_i))$.

*In items (1) and (3), every branch in $B$ with proper prefix $n_0, n_1, \ldots, n_i$ must also have prefix $n_0, n_1, \ldots, n_i, n_{i+1}$, whereas in (2) every branch in $B$ with proper prefix $n_0, n_1, \ldots, n_i$ has also prefix $n_0, n_1, \ldots, n_i, n_{i+1}$ or prefix $n_0, n_1, \ldots, n_i, n'$.*

In a coherent pre-tableau branches whose last node is inconsistent do not accept more enlargements or splittings. Every enlargement or splitting of a branch corresponds to the application of a TTM-rule or the unnext operator to its last node. The application of an $\alpha$-rule enlarges a branch $n_0, \ldots, n_i$ with a new node $n_{i+1}$ that includes, in the label, the constituents of the treated formula $\alpha$, but not $\alpha$ itself. So that, the application scheme for the $\alpha$-rules is

$$\begin{tikzcd}
\Delta, \alpha \arrow[d, no head] \\
\Delta, A(\alpha)
\end{tikzcd}$$

where $\alpha \notin \Delta$. The application of a $\beta$-rule splits a branch $n_0, \ldots, n_i$ with two new nodes $n_{i+1}$ and $n'$ in such a way that the label of one of the new nodes includes the constituents in $B_1(\beta)$ and the label of the other new node includes the constituents in $C(\beta, \Delta)$, where $C(\beta, \Delta)$ is either $B_2(\beta)$ or $B_2(\beta, \Delta)$, but the treated formula $\beta$ is not included in the labels of the two new nodes. So that, the application scheme for the $\beta$-rules is

$$\begin{tikzcd}
 & \Delta, \beta \arrow[dl, no head] \arrow[dr, no head] & \\
\Delta, B_1(\beta) & & \Delta, C(\beta, \Delta)
\end{tikzcd}$$

where $\beta \notin \Delta$. The application of the unnext operator enlarges a branch $n_0, \ldots, n_i$ with a new node $n_{i+1}$ whose label is $\mathsf{unnext}(L(n_i))$. The application scheme for the unnext operator is

$$\begin{tikzcd}
\Delta \arrow[d, no head] \\
\mathsf{unnext}(\Delta)
\end{tikzcd}$$

In order to ensure when an open branch describes a model, we deal with the notions of stage, cyclic branch, saturated set and fulfilling branch.

If we can ensure that the number of different labels used in the construction of a coherent pre-tableau $\mathcal{T}_\Phi$ is finite, then any infinite branch must contain infinitely many different nodes with the same label. In particular, when a repetition arises in an open branch

$$n_0, n_1, \ldots, n_{j-1}, n_j, \ldots, n_k$$

i.e. when $L(n_k) = L(n_{j-1})$ for some $j \in \{1, \ldots, k\}$, then an infinite branch of the form

$$n_0, n_1, \ldots, n_{j-1}, n_j, \ldots, n_k, n_j, \ldots, n_k, \ldots$$

can be obtained. In fact, this will be a cyclic branch that will be finitely represented.

**Definition 3.3.7.** *If $b = n_0, n_1, \ldots, n_k$ is an open branch such that $L(n_k) = L(n_{j-1})$ for some $j \in \{1, \ldots, k\}$, then $b$ is cyclic and we define*

$$\mathsf{cycle}(b) = n_j, n_{j+1}, \ldots, n_k$$

$$\mathsf{path}(b) = n_0, n_1, \ldots, n_{j-1} \cdot \langle n_j, n_{j+1}, \ldots, n_k \rangle^\omega$$

*In other words, we consider that the implicit successor relation on $b$ is extended with $n_k R n_j$. If a (closed or open) branch $b'$ is not cyclic then $\mathsf{path}(b') = b'$.*

The last node $n_k$ whose label appears previously in the branch is intentionally added to the branch because this repetition is what we will use in the systematic tableau for detecting the loop (see Subsection 3.4.2).

Every branch (cyclic or not) of a coherent pre-tableau can be seen as divided into *stages* according to the applications of the operator unnext. In other words, a stage is a sequence of consecutive nodes between two consecutive applications of the operator unnext.

**Definition 3.3.8.** *Given a branch $b$, every maximal subsequence $n_g, n_{g+1}, \ldots, n_h$ of* $\mathsf{path}(b)$ *such that $L(n_\ell) \neq \mathsf{unnext}(L(n_{\ell-1}))$ for every $\ell \in \{g+1, \ldots, h\}$, is called a* stage. *The functions* first *and* last *respectively return the first and the last node of a given stage. We denote by* $\mathsf{stages}(b)$ *the* sequence of all stages *of a branch $b$. The successor relation on* $\mathsf{stages}(b)$ *is induced by the successor relation on* $\mathsf{path}(b)$. *That is, if $s$ and $s'$ are respectively stages $n_0, \ldots, n_i$ and $n'_0, \ldots, n'_r$ in* $\mathsf{path}(b)$ *then $sRs'$ whenever $n_i R n'_0$. Hence, if $b = n_0, n_1, \ldots, n_j, \ldots, n_k$ is a cyclic branch such that* $\mathsf{cycle}(b) = n_j, n_{j+1}, \ldots, n_k$ *(and $j \geq 1$), then* $\mathsf{stages}(b)$ *is a non-empty finite sequence of stages $s_0, s_1, \ldots, s_m$ such that* $\mathsf{last}(s_m) R \mathsf{first}(s_z)$ *for some $z \in \{1, \ldots, m\}$ and $n$ belongs to* $\mathsf{cycle}(b)$ *for every $y \in \{z, \ldots, m\}$ and every node $n$ in $s_y$. For such a cyclic branch $b$, we respectively denote by* $\mathsf{stages}(\mathsf{cycle}(b))$ *and* $\mathsf{path}(\mathsf{stages}(b))$ *the sequences $s_z, \ldots, s_m$ and $s_0, \ldots, s_{z-1} \cdot \langle s_z, \ldots, s_m\rangle^\omega$.*

The following example serves to illustrate the notions of stages and path by means of a sample branch.

**Example 3.3.9.** *Consider a cyclic branch $b = n_0, n_1, n_2, n_3, n_4$ such that $L(n_4) = L(n_2)$. Then,* $\mathsf{path}(b) = n_0, n_1, n_2 \cdot \langle n_3, n_4\rangle^\omega$. *Let us suppose that $L(n_1) = \mathsf{unnext}(L(n_0))$ and $L(n_4) = \mathsf{unnext}(L(n_3))$. Then,* $\mathsf{stages}(b)$ *is formed by three stages: $s_0 = \langle n_0\rangle$, $s_1 = \langle n_1, n_2, n_3\rangle$ and $s_2 = \langle n_4, n_3\rangle$. Therefore, the induced relation $R$ on* $\mathsf{stages}(b)$ *is given by $s_0 R s_1$, $s_1 R s_2$ and $s_2 R s_2$. Hence,* $\mathsf{path}(\mathsf{stages}(b)) = s_0, s_1 \cdot \langle s_2\rangle^\omega$.

With a slight abuse of notation, the labelling function $L$ is extended from nodes to stages in the natural way. That is, for any stage $s$:

$$L(s) = \bigcup_{n \in s} L(n).$$

The general notion of fulfillment is introduced at the end of Section 2.2. Now we adapt such notion to our tableau system.

**Definition 3.3.10.** *Let $S$ be a sequence of stages, $s \in S$ and $\varphi \mathcal{U} \psi \in L(s)$, we say that $\varphi \mathcal{U} \psi$ is fulfilled in $S$ iff there exists $s'$ such that $sR^*s'$ and $\psi \in L(s')$. A sequence $S$ of stages is* fulfilling *iff for all $s \in S$ every $\varphi \mathcal{U} \psi \in L(s)$ is fulfilled in $S$. A branch $b$ is fulfilling iff the sequence* $\mathsf{path}(\mathsf{stages}(b))$ *is fulfilling.*

The concept of fulfilling branch together with the following concept of $\alpha\beta$-saturated stage is crucial for determining when branches are able to describe a model.

**Definition 3.3.11.** *A stage $s$ is $\alpha\beta$-*saturated *if and only if for every $\varphi \in L(s)$:*

1. *If $\varphi$ is an $\alpha$-formula then $A(\varphi) \subseteq L(s)$*
2. *If $\varphi$ is a $\beta$-formula then $B_1(\varphi) \subseteq L(s)$ or $B_2(\varphi) \subseteq L(s)$ or $B_2(\varphi, \Delta) \subseteq L(s)$, where $\Delta = L(n_i) \setminus \{\varphi\}$ for some $n_i \in s$ such that $\varphi = \chi \mathcal{U} \gamma \in L(n_i)$.*

Now, we give a sufficient condition to consider that an open branch is (sufficiently) expanded. That is, it is able to describe a collection of models. This condition can be syntactically checked. For the construction of systematic tableaux (see Subsection 3.4.2), we will refine this sufficient condition to a simpler one (see Remark 3.4.8).

**Definition 3.3.12.** *An open branch $b$ is* expanded *if and only if $b$ is fulfilling, cyclic and each stage $s \in \mathsf{stages}(b)$ is $\alpha\beta$-saturated.*

For example, an expanded branch of a coherent pre-tableau for $\{r\,\mathcal{U}\,p\}$ can be formed by the sequence of stages $s_0, s_1, s_2$ where $L(s_0) = \{r\,\mathcal{U}\,p, p\}$ and $L(s_1) = L(s_2) = \emptyset$. Actually that branch is fulfilling, cyclic and $\alpha\beta$-saturated, hence it is expanded. Also the sequence of stages $x_0, x_1, x_2, x_3$ where $L(x_0) = \{r\,\mathcal{U}\,p, r, \neg p, \circ(r\,\mathcal{U}\,p)\}$, $L(x_1) = \{r\,\mathcal{U}\,p, p\}$ and $L(x_3) = L(x_4) = \emptyset$ is an expanded branch. It is worth noting that expanded branches can be enlarged. For instance an expanded branch of a coherent pre-tableau for $\{\Box\,(r\,\mathcal{U}\,p)\}$ is given by the sequence of stages $z_0, z_1$ where $L(z_0) = L(z_1) = \{\Box\,(r\,\mathcal{U}\,p), \circ\Box\,(r\,\mathcal{U}\,p), r\,\mathcal{U}\,p, p\}$. But the sequence of stages $z_0, z_1, z_2, z_3, z_4$ where $L(z_0)$ and $L(z_1)$ are as above, $L(z_2) = \{\Box\,(r\,\mathcal{U}\,p), \circ\Box\,(r\,\mathcal{U}\,p), r\,\mathcal{U}\,p, r, \neg p, \circ(r\,\mathcal{U}\,p)\}$, $L(z_3) = L(z_0)$ and $L(z_4) = L(z_2)$ is an expanded branch too.

**Remark 3.3.13.** *Enlargement of expanded branches is used in Subsection 3.4.2 for the systematic construction of tableaux in order to ensure the construction of fulfilling branches without checking directly whether a branch is fulfilling (see Remark 3.4.8).*

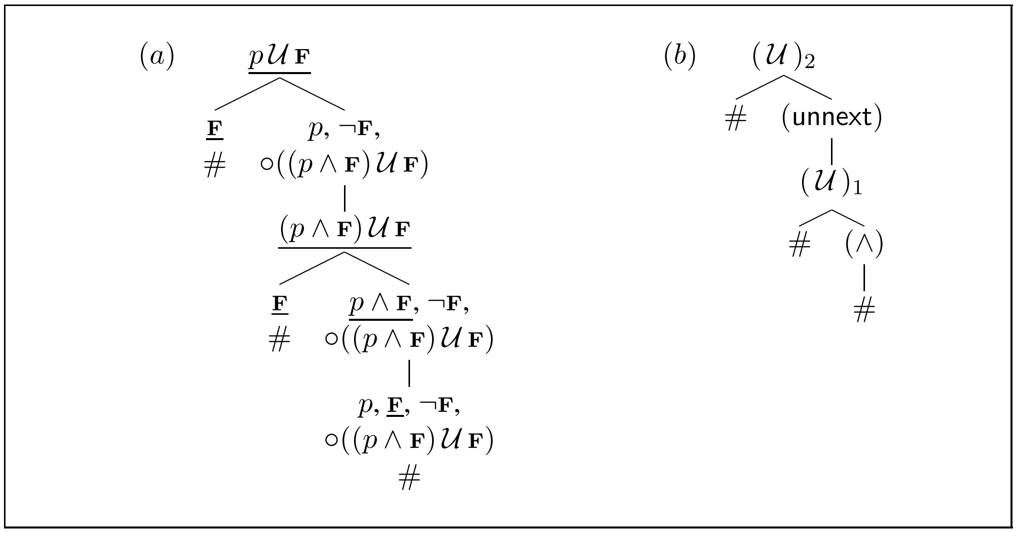
When constructing a tableau, only open branches (expanded or non-expanded) can be enlarged. A completely expanded tableau is constructed for deciding if the original set of formulas is satisfiable or not, respectively depending on whether there is at least one expanded open branch or all its branches are closed.

**Definition 3.3.14. (Tableau)** *A* tableau *for a set of formulas $\Phi$ is a coherent pre-tableau for $\Phi$. An* expanded tableau *is a tableau where every maximal branch is either expanded or closed. An expanded tableau is open if it has at least one open maximal branch*[5]*, otherwise it is closed.*

### *3.3.5 Examples of Tableaux*

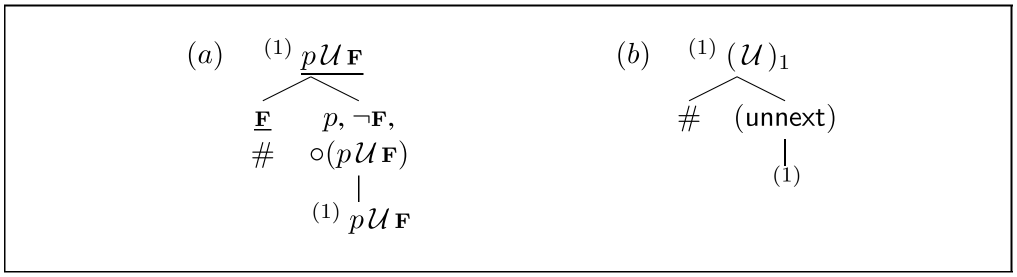
In this subsection, we give some examples of tableaux. Each tableau is showed by means of a figure formed by a part $(a)$ and a part $(b)$. The $(a)$ part of the figure is a tree that contains the sets of formulas that label each node of the tableau and the $(b)$ part of the figure is a tree that shows the rules applied at each step. For space reasons, the $(a)$ and $(b)$ part may appear in the same figure or in different figures. For readability, we also underline the formula which the TTM-rule is applied to. When the unnext operator is applied, we do not underline any formula. In the nodes in which we apply the rule $(\,\mathcal{U}\,)_2$ or the rule $(\diamond)_2$, we only underline the eventuality to which the rule is applied. Branches with the mark $\#$ are closed branches. In the last nodes of closed branches, we underline the formulas that cause inconsistency. However, when a node is inconsistent for more than one reason, we only point out one of them. Note that, when a formula is treated at a node $n$ of a stage $s$, this formula does not appear in the label of any successor of $n$ that belongs to the stage $s$, although it remains belonging to the label of $s$. Hence, already treated formulas cannot be used to expand a branch again (at the same stage). Additionally,

[5] which is expanded.



(a)
$p\mathcal{U}\mathbf{F}$
$\underline{\mathbf{F}}$
#
$p, \neg\mathbf{F}, \circ((p \wedge \mathbf{F})\mathcal{U}\mathbf{F})$
$(p \wedge \mathbf{F})\mathcal{U}\mathbf{F}$
$\underline{\mathbf{F}}$
#
$p \wedge \mathbf{F}, \neg\mathbf{F}, \circ((p \wedge \mathbf{F})\mathcal{U}\mathbf{F})$
$p, \underline{\mathbf{F}}, \neg\mathbf{F}, \circ((p \wedge \mathbf{F})\mathcal{U}\mathbf{F})$
#
(b)
$(\mathcal{U})_2$
#
(unnext)
$(\mathcal{U})_1$
#
$(\wedge)$
#

*Figure 3.3:* Closed tableau for the set of formulas $\{p\mathcal{U}\mathbf{F}\}$



(a)
$^{(1)}\, p\mathcal{U}\mathbf{F}$
$\underline{\mathbf{F}}$
#
$p, \neg\mathbf{F}, \circ(p\mathcal{U}\mathbf{F})$
$^{(1)}\, p\mathcal{U}\mathbf{F}$
(b)
$^{(1)}\, (\mathcal{U})_1$
#
(unnext)
(1)

*Figure 3.4:* Non-expanded tableau for the set of formulas $\{p\mathcal{U}\mathbf{F}\}$

since open expanded branches are cyclic, we mark the leaf and the internal repeated node with the same superscript of the form $^{(i)}$ where $i \geq 1$.

**Example 3.3.15.** *In Figure 3.3 a closed expanded tableau for the unsatisfiable set of formulas $\{p\mathcal{U}\mathbf{F}\}$ is showed.*

*Note that the rightmost branch consists of two stages, the first one is formed by the two higher nodes. The remaining three nodes form the second stage of the branch.*

*The set of formulas $\{p\mathcal{U}\mathbf{F}\}$ also serves to show that by using only the rule $(\mathcal{U})_1$ the fulfillment of an eventuality can be indefinitely delayed. In particular, the set of formulas $\{p\mathcal{U}\mathbf{F}\}$ cannot be* TTM*-refuted without using the rule $(\mathcal{U})_2$ (see Figure 3.4). The rightmost branch, namely $b$, of the tableau in Figure 3.4 is cyclic and is made up of two stages, $x_0$ and $x_1$. The first two nodes form the stage $x_0$ and the third and second nodes form the stage $x_1$. Therefore* $\mathsf{path}(\mathsf{stages}(b)) = x_0 \cdot \langle x_1 \rangle^\omega$. *Although the branch $b$ is open and cyclic and each stage is $\alpha\beta$-saturated, $b$ is not an expanded branch because it is not fulfilling.*

**Example 3.3.16.** *In Figure 3.5 an open expanded tableau for the satisfiable set of formulas $\{p, \circ\neg p, \neg\mathbf{F}\mathcal{U}\neg p\}$ is showed. The tableau has two closed branches and one expanded open branch, which is the central one. This open branch, which here we refer to as $b$, describes a*

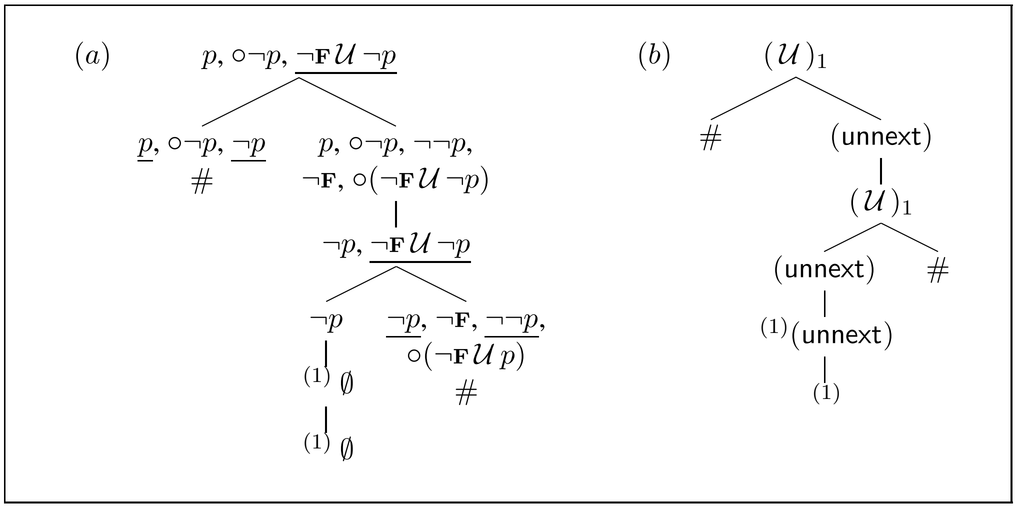

(a)
$p, \circ\neg p, \underline{\neg\mathbf{F}\,\mathcal{U}\,\neg p}$
$\underline{p}, \circ\neg p, \underline{\neg p}$
#
$p, \circ\neg p, \neg\neg p,$
$\neg\mathbf{F}, \circ(\neg\mathbf{F}\,\mathcal{U}\,\neg p)$
$\neg p, \underline{\neg\mathbf{F}\,\mathcal{U}\,\neg p}$
$\neg p$
$^{(1)}\,\emptyset$
$^{(1)}\,\emptyset$
$\underline{\neg p}, \neg\mathbf{F}, \underline{\neg\neg p},$
$\circ(\neg\mathbf{F}\,\mathcal{U}\,p)$
#
(b)
$(\mathcal{U})_1$
#
(unnext)
$(\mathcal{U})_1$
(unnext)
#
$^{(1)}$(unnext)
(1)

*Figure 3.5:* Open expanded tableau for the set of formulas $\{p, \circ\neg p, \neg\mathbf{F}\,\mathcal{U}\,\neg p\}$

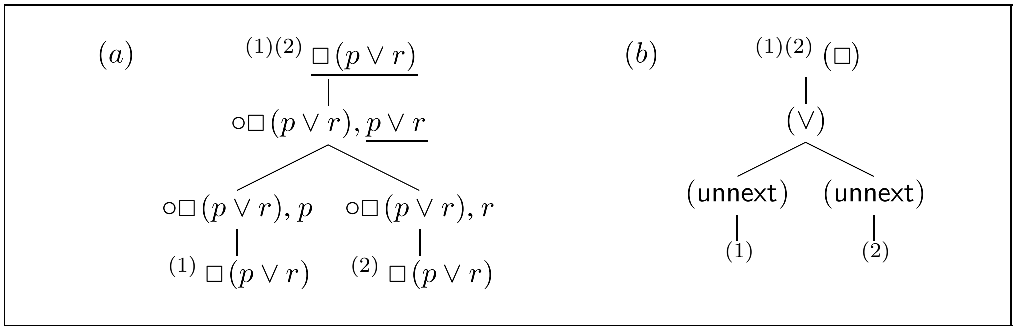

(a)
$^{(1)(2)}\,\underline{\Box\,(p \vee r)}$
$\circ\Box\,(p \vee r), \underline{p \vee r}$
$\circ\Box\,(p \vee r), p$
$\circ\Box\,(p \vee r), r$
$^{(1)}\,\Box\,(p \vee r)$
$^{(2)}\,\Box\,(p \vee r)$
(b)
$^{(1)(2)}\,(\Box)$
$(\vee)$
(unnext)
(unnext)
(1)
(2)

*Figure 3.6:* Open expanded tableau for the set of formulas $\{\Box\,(p \vee r)\}$

*collection of models. The first state $s_0$ of those models should make true the formulas labelling the first stage (let us say $x_0$) of the branch which is formed by the first two nodes. In particular, $p$ should be true at the first state $s_0$. The second stage, $x_1$, is given by the third and fourth nodes of the branch, in particular $\neg p$ should be true in the second state ($s_1$) of such collection of models. In fact, any infinite sequence of states prefixed by these two states, $s_0$ and $s_1$, is a model of the root of the tableau since the third and fourth stages of the branch, namely $x_2$ and $x_3$, are given by the fifth and sixth nodes that are labelled by the empty set. Note that* $\mathsf{path}(\mathsf{stages}(b)) = x_0, x_1, x_2 \cdot \langle x_3\rangle^\omega$.

**Example 3.3.17.** *In Figure 3.6 we show an open expanded tableau for the satisfiable set of formulas $\{\Box\,(p \vee q)\}$. This tableau has two expanded (open) branches $b_1$ (on the left) and $b_2$ (on the right). Regarding the branch $b_1$, the first three nodes form a stage $x_0$ and the fourth node together with the second and third nodes form another stage $x_1$ and* $\mathsf{path}(\mathsf{stages}(b_1)) = x_0 \cdot \langle x_1\rangle^\omega$. *This open branch describes a collection of models. The first state of those models should make true the formulas labelling the first stage and all the other states should make true the formulas labelling the second stage. In particular $p$ should be true in all the states of those*

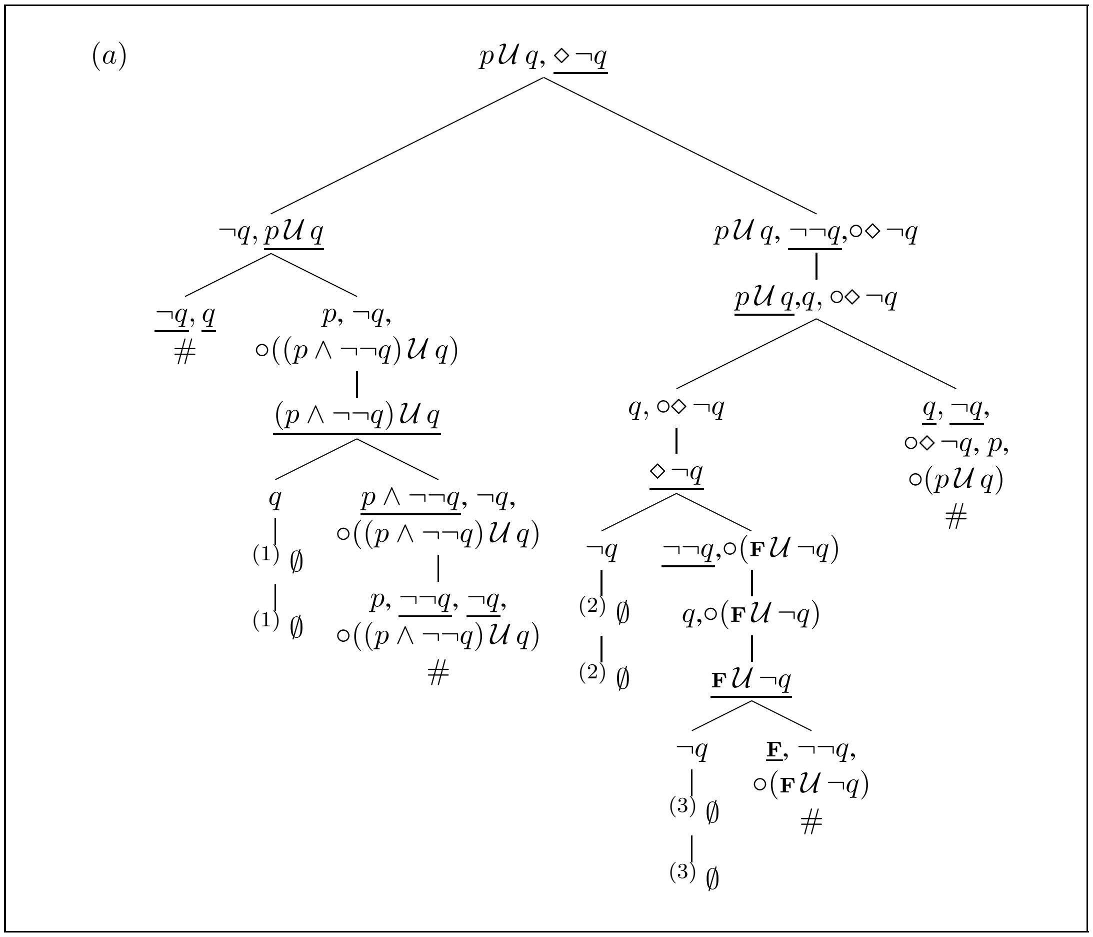


*Figure 3.7:* Open expanded tableau for the set of formulas $\{p\mathcal{U}\,q, \diamond\neg q\}$ (Part 1 of 2)

*models. The case of the branch $b_2$ is symmetric with the difference that $r$ should be true in all the states of the models described by $b_2$.*

**Example 3.3.18.** *The tableau in Figure 3.7 is an open expanded tableau for the satisfiable set of formulas $\{p\mathcal{U}\,q, \diamond\neg q\}$. Due to space reasons, the $(b)$ part of the tableau is in Figure 3.8. Note that the derived rules $(\diamond)_1$ and $(\diamond)_2$ –which are shown in Figure 3.2– are used. This tableau has three expanded open branches describing three different collections of models. The leftmost open branch, that here we refer to as $b_1$, represents the class of models with a first state where $p$ and $\neg q$ are true and a second state where $q$ is true. In $b_1$ the first three nodes form a stage, let us say $x_0$, the fourth and fifth nodes form a stage $x_1$, and the sixth and seventh nodes form, respectively, stages $x_2$ and $x_3$. Since the cycle of $b_1$ is formed by the seventh node,* $\mathsf{path}(\mathsf{stages}(b_1)) = x_0, x_1, x_2 \cdot \langle x_3\rangle^\omega$. *In the first state of the models represented by the central open branch $b_2$, the propositional variable $q$ is true, whereas in the second one $\neg q$ holds. As in the branch $b_1$, in the branch $b_2$ we can differentiate four stages, namelly $y_0, \ldots, y_3$, and the cycle is formed by the last node of the branch. The stage $y_3$ is formed by the last node of the branch and* $\mathsf{path}(\mathsf{stages}(b_2)) = y_0, y_1, y_2 \cdot \langle y_3\rangle^\omega$. *Finally, the rightmost open branch, $b_3$, represents models whose first three states respectively make true the literals $q$, $q$*

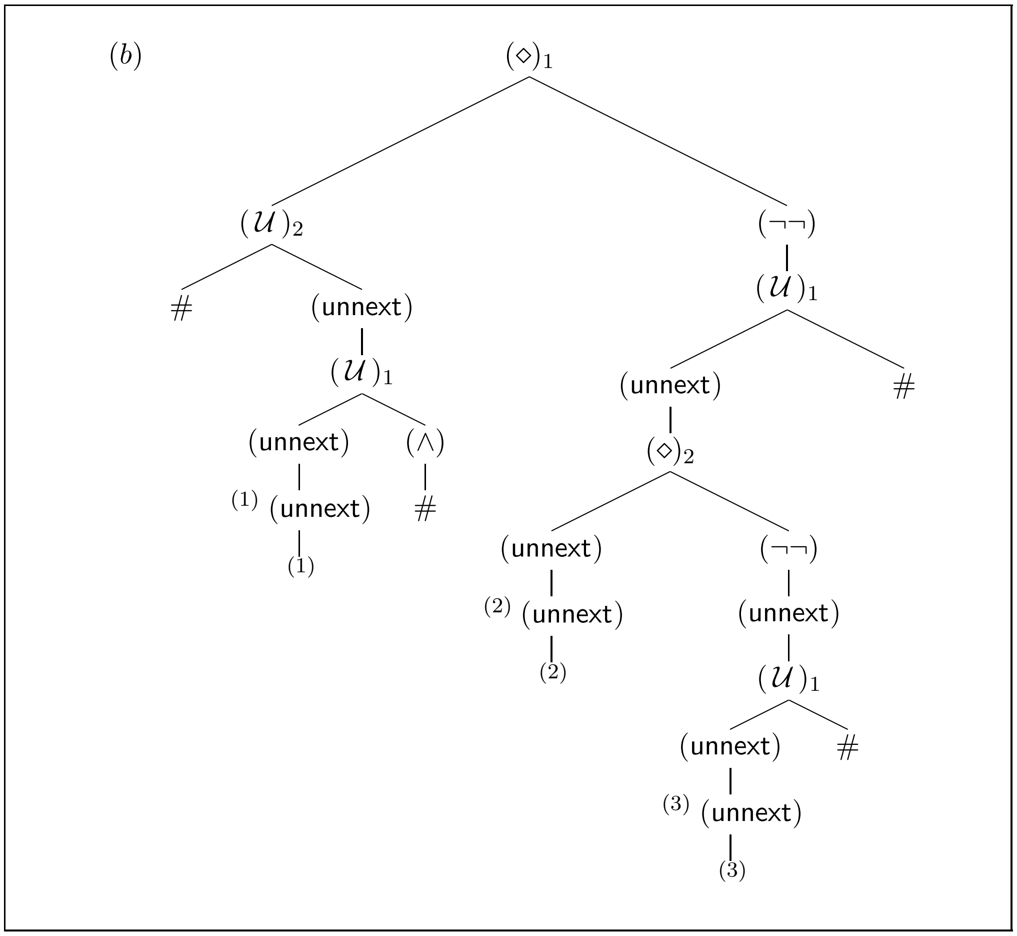


*Figure 3.8:* Open expanded tableau for the set of formulas $\{p\mathcal{U}\,q, \diamond\neg q\}$ (Part 2 of 2)

*and $\neg q$. In the branch $b_3$, the four applications of the* unnext *operator give rise to five stages, $z_0, \ldots, z_4$. The cycle of $b_3$, as well as the stage $z_4$, are formed by the last node of the branch and* $\mathsf{path}(\mathsf{stages}(b_3)) = z_0, z_1, z_2, z_3 \cdot \langle z_4 \rangle^\omega$.

## 3.4 Soundness and Completeness of TTM

In this section we first adapt, to the field of tableau methods, the notions of soundness, refutational completeness and completeness introduced in Section 2.3. Then, we prove that the tableau system TTM is sound, refutationally complete and also complete.

A tableau method is *sound* if, whenever a closed tableau exists for $\Phi$, then $\Phi$ is unsatisfiable. A tableau method is *refutationally complete* if, whenever $\Phi$ is unsatisfiable, a closed tableau for $\Phi$ can be constructed. Therefore, a sound and refutationally complete tableau method guarantees that, given a set of formulas $\Phi$, a refutation (i.e. a closed tableau) is obtained if and only if the set $\Phi$ is unsatisfiable. A tableau method is *complete* if both satisfiability and unsatisfiability are decidable. However, soundness and refutational completeness do not guarantee that for satisfiable sets of formulas such satisfiability is decidable. A termination proof is additionally required in order to prove completeness.

Subsection 3.4.1 is devoted to soundness. In Subsection 3.4.2 we introduce an algorithm for the construction of systematic tableaux together with the concepts and results that the algorithm and its correctness give rise to. In particular, we discuss about the analytic superformula property and present our notion of closure, which serves to proof that the algorithm terminates for any finite set of formulas. The worst case complexity is also established. In Subsection 3.4.3 we give some examples of systematic tableaux. In Subsection 3.4.4 we prove the completeness of TTM, by proving, as a first step, its refutational completeness. In Subsection 3.4.5 we provide a practical improvement of the rule $(\mathcal{U})_2$.

### *3.4.1 Soundness*

In this section we first show that the TTM-rules preserve equisatisfiability (Definition 2.2.2) and that the unnext operator preserves satisfiability. Then, soundness is proved in Theorem 3.4.2.

The soundness of a system can be guaranteed rule by rule, where a rule is sound whenever it preserves the satisfiability.

**Lemma 3.4.1.** *For every set of formulas $\Phi$, any $\alpha$-formula $\gamma$ and any $\beta$-formula $\chi$:*

1. $\Phi \cup \{\gamma\}$ *is satisfiable iff* $\Phi \cup A(\gamma)$ *is satisfiable*
2. $\Phi \cup \{\chi\}$ *is satisfiable iff* $\Phi \cup B_1(\chi)$ *or* $\Phi \cup B_2(\chi)$ *or* $\Phi \cup B_2(\chi, \Phi)$ *is satisfiable.*
3. *If* $\Phi$ *is satisfiable then* $\mathsf{unnext}(\Phi)$ *is satisfiable.*

*Proof.* The case of the rule $(\mathcal{U})_2$ is proved by using Proposition 3.3.3. The remaining cases are straightforwardly proved by using the semantics of the connectives and the operator unnext, presented in Section 2.2, and Definition 3.3.4. ■

Hence, soundness can be proved.

**Theorem 3.4.2.** *If there exists a closed expanded tableau for $\Phi$ then $\Phi$ is unsatisfiable.*

*Proof.* Let $\mathcal{T}_\Phi$ be a closed expanded tableau for $\Phi$. The set of formulas labelling each leaf is inconsistent and therefore unsatisfiable. Then, by Lemma 3.4.1, each node in $\mathcal{T}_\Phi$ is labelled with an unsatisfiable set of formulas, in particular the root. Therefore $\Phi$ is unsatisfiable. ■

### *3.4.2 Systematic Tableaux*

In this subsection we provide an algorithm for systematically building an expanded tableau. We also study the main properties that our systematic tableau satisfies and we proof that the algorithm terminates for any set of formulas given as input.

Unlike in complete tableau methods for propositional classical logic, the nondeterministic application of the TTM-rules and the unnext operator does guarantee neither refutational completeness nor completeness. In order to guarantee refutational completeness and completeness we provide an algorithm that, given a set of formulas $\Phi$, constructs an expanded tableau for $\Phi$ that we denote by $\mathcal{T}_\Phi$. The tableau $\mathcal{T}_\Phi$ will be closed if $\Phi$ is unsatisfiable and open otherwise.

The systematic tableau algorithm is depicted by a while-program in Figure 3.9. The systematic tableau construction provides a proof search procedure for automated deduction.

```
Input: A finite set of formulas $\Phi$
Output: An expanded tableau $\mathcal{T}_\Phi = (\mathsf{Nodes}, n_\Phi, L, B, R)$ for $\Phi$

Nodes := $\{n_\Phi\}$; $L := \{(n_\Phi, \Phi)\}$; $B := \{n_\Phi\}$; $R := \emptyset$; selfun := $\{(n_\Phi, \emptyset)\}$
while unmarked_branches$(B) \neq \emptyset$ loop
  choose $b \in$ unmarked_branches$(B)$
  $n_k :=$ last_node$(b)$;
  if selfun$(n_k) = \emptyset$ then fair_select$(b, \mathcal{T}_\Phi,$ selfun$)$ end if
  if $L(n_k) \setminus$ selfun$(n_k)$ is not elementary
    then choose $\gamma \in L(n_k) \setminus$ selfun$(n_k)$
         non-select_expand$(\gamma, b, \mathcal{T}_\Phi,$ selfun$)$
    else if selfun$(n_k)$ is neither empty nor elementary
         then select_expand$(b, \mathcal{T}_\Phi,$ selfun$)$
         else $\{L(n_k)$ is elementary$\}$ unnext_expand$(b, \mathcal{T}_\Phi,$ selfun$)$
  end if
end loop
```

*Figure 3.9:* Systematic Tableau Algorithm

The construction of $\mathcal{T}_\Phi$ consists in a systematic extension of branches using TTM-rules for decomposing $\alpha$- and $\beta$-formulas into their constituents. When the current stage (Definition 3.3.8) becomes $\alpha\beta$-saturated (Definition 3.3.11), and consequently, the $\alpha$- and $\beta$-rules cannot be applied, the operator unnext (Definition 3.3.4) is used to jump to a new stage. Regarding the use of the rules $(\mathcal{U})_1$ and $(\mathcal{U})_2$, a specific strategy is followed. During the construction of each stage, one eventuality –if there is any– is fixed as selected (Figure 3.9, line 5, fair_select). Then the TTM-rules (except $(\mathcal{U})_2$) are nondeterministically applied until we obtain a set of formulas where every formula, except the selected eventuality, is elementary (see Subsection 2.1). However, at each iteration step only one formula is chosen (Figure 3.9, line 7) for applying the corresponding rule (Figure 3.9, line 8, non-select_expand) and, consequently, in general several iteration steps are needed to obtain a set where only the selected eventuality is non-elementary. At that point, the rule $(\mathcal{U})_2$ is applied to the selected eventuality (Figure 3.9, line 10, select_expand), if there is any (what is checked in line 9). When $(\mathcal{U})_2$ is applied, new non-elementary formulas may appear. Consequently, the TTM-rules (except $(\mathcal{U})_2$) are nondeterministically applied again until we obtain an elementary set of formulas. Note that again, in general, several iteration steps will be needed to obtain an elementary set. The construction of a stage stops if an inconsistent node (Definition 3.3.5) is obtained because the corresponding branch is marked as closed and only unmarked branches are considered for further enlargements or splittings (Figure 3.9, line 2, unmarked_branches). If all the nodes of the stage we are constructing are consistent and the label of the last node of the stage is elementary, then the operator unnext is applied (line 11, unnext_expand) and the construction of the next stage begins.

When, during the construction of a stage, the rule $(\mathcal{U})_2$ is applied to the selected eventuality $\varphi\,\mathcal{U}\,\psi$ with context $\Delta_0$, the branch is split into two branches, let us say $b_1$ and $b_2$. The label of the last node in the branch $b_1$ is $\Delta_0 \cup \{\psi\}$ and the eventuality $\varphi\,\mathcal{U}\,\psi$ is fulfilled in this branch. Therefore it represents an attempt to make $\psi$ true in this state. However, $b_1$ could still be the prefix of a closed branch. If following the enlargement of $b_1$ the next stage is created, i.e., if the branch does not close before applying the operator unnext, another eventuality must be selected, if there is any available eventuality. The label of the last node in the branch $b_2$ is $\Delta_0 \cup \{\varphi, \neg\psi, \circ((\varphi \wedge \neg\Delta_0)\,\mathcal{U}\,\psi)\}$. Therefore, in the branch $b_2$, the fulfillment of $\varphi\,\mathcal{U}\,\psi$ is postponed, but in the next stage, if the next stage is created, the eventuality $(\varphi \wedge \neg\Delta_0)\,\mathcal{U}\,\psi$ will be necessarily the selected one. In other words, the idea is to select an eventuality $\varphi\,\mathcal{U}\,\psi$ and to apply $(\mathcal{U})_2$ only to $\varphi\,\mathcal{U}\,\psi$ and to the eventualities generated from it in the branch where the inclusion of $\psi$, i.e., the fulfillment of $\varphi\,\mathcal{U}\,\psi$ is postponed. The eventualities generated from $\varphi\,\mathcal{U}\,\psi$ can be described as follows:

$$(\varphi \wedge \neg\Delta_0)\,\mathcal{U}\,\psi \qquad (\varphi \wedge \neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\psi \qquad \ldots \qquad (\varphi \wedge \neg\Delta_0 \wedge \neg\Delta_1 \wedge \ldots \wedge \neg\Delta_k)\,\mathcal{U}\,\psi$$

where $\Delta_i$ is the context at the moment of applying $(\mathcal{U})_2$ to $(\varphi \wedge \neg\Delta_0 \wedge \neg\Delta_1 \wedge \ldots \wedge \neg\Delta_{i-1})\,\mathcal{U}\,\psi$. Those eventualities are generated in different (consecutive) stages.

Since new eventualities are built up during the process, we must guarantee termination. Classical propositional tableaux satisfy the *subformula property* (SP):

> For every formula $\psi$ used in the construction of any tableau for $\Phi$, there exists some formula $\gamma \in \Phi$ such that $\psi$ is a (possibly negated) subformula of $\gamma$.

This property ensures the termination of the construction of any tableau for a (finite) set of formulas. Most tableau systems for modal and temporal logics, fail to satisfy the SP, since some of their rules introduce formulas that are not subformulas of the principal formula of the rule. Hence, termination of modal/temporal tableaux is not obvious. However, most tableau systems for modal and temporal logics, satisfy the *analytic superformula property* (ASP):

> For every finite set of formulas $\Phi$, there exists a finite set that contains all the formulas that may occur in any tableau for $\Phi$.

Such set is usually called the closure of $\Phi$. The ASP also ensures the non-existence of infinite branches where all the nodes have different labels. Hence, by controlling loops, the finiteness of proof search can be ensured. In our case, as a consequence of using the rule $(\mathcal{U})_2$, the tableau system TTM fails to satisfy the ASP. However, TTM satisfies a slightly weaker variant that is enough for ensuring completeness and that we call the *weak analytic superformula property* (WASP):

> For every finite set of formulas $\Phi$, there exists a finite set that contains all the formulas that may occur in any *systematic tableau* for $\Phi$.

Our algorithm (Figure 3.9) constructs a systematic tableau $\mathcal{T}_\Phi$ for any $\Phi$ such that TTM satisfies the WASP with respect to the set $\mathsf{clo}(\Phi)$ (closure of $\Phi$) (see Definition 3.4.9).

In order to satisfy the WASP, the algorithm keeps at most one selected formula to which the rule $(\mathcal{U})_2$ can be applied and when a new eventuality is generated in one stage, by using $(\mathcal{U})_2$, that new eventuality is the selected eventuality in the next stage. In this way, when the rule $(\mathcal{U})_2$ is applied with context $\Delta_h$, the eventualities previously built by using $(\mathcal{U})_2$ are never in $\Delta_h$. Consequently there are only a finite number of different contexts and this leads to the fact that after a finite process it must happen that when the rule $(\mathcal{U})_2$ is applied to the set

$$\Delta_i \cup \{(\varphi \wedge \neg\Delta_0 \wedge \neg\Delta_1 \wedge \ldots \wedge \neg\Delta_{i-1})\,\mathcal{U}\,\psi\}$$

the context $\Delta_i$ is equal to some $\Delta_j$ with $j \in \{0, \ldots, i-1\}$. In such a case, the new set of formulas that corresponds to the branch that postpones the fulfillment of $\varphi\,\mathcal{U}\,\psi$ is

$\Delta_i \cup \{\varphi \wedge \neg\Delta_0 \wedge \neg\Delta_1 \wedge \ldots \wedge \neg\Delta_{i-1}, \neg\psi, \circ((\varphi \wedge \neg\Delta_0 \wedge \neg\Delta_1 \wedge \ldots \wedge \neg\Delta_{i-1} \wedge \neg\Delta_i)\,\mathcal{U}\,\psi)\}$

and a contradiction is generated from $\Delta_i$ and $\neg\Delta_0 \wedge \neg\Delta_1 \wedge \ldots \wedge \neg\Delta_{i-1}$. This ensures that the branch where the fulfillment of $\varphi\,\mathcal{U}\,\psi$ is always postponed will eventually close. Regarding the branch that does not postpone the fulfillment of $\varphi\,\mathcal{U}\,\psi$, the new set is $\Delta_i \cup \{\psi\}$ and it does not contain any eventuality generated by the rule $(\,\mathcal{U}\,)_2$ and there are only finite different sets of this kind, so repeated labels, that give rise to cycles, must necessarily appear after a finite number of tableau expansion steps. This strategy guarantees that a finite amount of steps is sufficient to decide whether the selected eventuality can be fulfilled or not. If the eventuality cannot be fulfilled, the corresponding branches close. If the eventuality can be fulfilled, when the eventuality is fulfilled another eventuality is selected and the process goes on. This selection must be done in a fair manner, i.e., an eventuality that from some stage onwards appears as an eligible eventuality whenever an eventuality must be selected, cannot remain indefinitely unselected. For handling selected formulas the algorithm uses a selection function selfun. Along the construction of the systematic tableau, the function selfun associates to every node $n$ one of the following three possible sets of formulas:

1. the empty set
2. a non-elementary singleton of the form $\{\varphi\,\mathcal{U}\,\psi\}$
3. an elementary singleton of the form $\{\circ(\varphi\,\mathcal{U}\,\psi)\}$.

The case 1 means that no until-formula is selected. In 2, selfun yields the set containing the selected until-formula to which $(\,\mathcal{U}\,)_2$ will be applied in the current stage. The case 3 happens for every node $n$ of a stage $s$ that has been created after the application of $(\,\mathcal{U}\,)_2$ in a node $n' \in s$ and in which the fulfillment of the eventuality in selfun$(n')$ has been postponed. Therefore, case 3 means that $\circ(\varphi\,\mathcal{U}\,\psi)$ is the formula that has been obtained by applying the rule $(\,\mathcal{U}\,)_2$ to a formula $\chi\,\mathcal{U}\,\psi \in$ selfun$(n')$ and that in the next stage $\varphi\,\mathcal{U}\,\psi$ will be the selected eventuality. At the beginning, selfun associates the empty set to the initial node.

In order to construct $\mathcal{T}_\Phi$, our algorithm nondeterministically chooses, at each step, a maximal branch to be extended. The algorithm ends when every maximal branch is marked either as closed or as expanded.

The procedure unmarked_branches yields the maximal branches that are not marked yet. For extending the chosen branch, the algorithm uses three procedures. First, a procedure non-select_expand that applies the corresponding TTM-rule, excepting $(\,\mathcal{U}\,)_2$, to a formula that has been nondeterministically chosen from the set of non-selected formulas in the last node $n$ of the branch, i.e., from the set $L(n) \setminus$ selfun$(n)$. Second, when the TTM-rules other than $(\,\mathcal{U}\,)_2$ cannot be further applied, the procedure select_expand applies the rule $(\,\mathcal{U}\,)_2$ to the until-formula that is *selected* by the function selfun, if there is some. The procedure fair_select updates the function selfun using a fair strategy. Third, when the node is labelled by an elementary set, then the operator unnext is applied using the procedure unnext_expand. Let us give a more detailed explanation of all the procedures used by the algorithm.

last_node$(b)$ gives the last node added to a given branch $b$.

non-select_expand$(\gamma, b, \mathcal{T}_\Phi,$ selfun$)$ applies to the branch $b$ the $\alpha$- or $\beta$-rule (excepting $(\,\mathcal{U}\,)_2$) that corresponds to the formula $\gamma$. In both cases, the formula selected by the function selfun is preserved. That is, for $n_k =$ last_node$(b)$:

- If $\gamma$ is an $\alpha$-formula, create a new node $n$ and a new branch $b' = b \cdot n$ according to the corresponding $\alpha$-rule such that $L(n) = (L(n_k) \setminus \{\gamma\}) \cup A(\gamma)$ and extend selfun and $R$ to be $\mathsf{selfun}(n) = \mathsf{selfun}(n_k)$ and $n_k R n$.
- If $\gamma$ is a (non-selected) $\beta$-formula, create two new nodes $n'$ and $n''$ and two new branches $b' = b \cdot n'$ and $b'' = b \cdot n''$ according to the corresponding $\beta$-rule such that $L(n') = (L(n_k)\setminus\{\gamma\}) \cup B_1(\gamma)$ and $L(n'') = (L(n_k)\setminus\{\gamma\}) \cup B_2(\gamma)$. Extend selfun and $R$ to be $\mathsf{selfun}(n') = \mathsf{selfun}(n_k)$, $\mathsf{selfun}(n'') = \mathsf{selfun}(n_k)$ and $n_k R n', n_k R n''$.

$\mathsf{select\_expand}(b, \mathcal{T}_\Phi, \mathsf{selfun})$ applies the rule $(\mathcal{U})_2$ to an until-formula $\varphi \mathcal{U} \psi$ that is selected by the function selfun. The function selfun yields the empty set for the new node that contains $\psi$ since the until-formula has been fulfilled. In the other branch, the new selected formula is $\circ((\varphi \wedge \neg\Delta) \mathcal{U} \psi)$. That is, for $n_k = \mathsf{last\_node}(b)$:

Let $\mathsf{selfun}(n_k) = \{\varphi \mathcal{U} \psi\}$. Create two new nodes $n'$ and $n''$ and two new branches $b' = b \cdot n'$ and $b'' = b \cdot n''$ such that $L(n') = (L(n_k) \setminus \{\varphi \mathcal{U} \psi\}) \cup \{\psi\}$ and $L(n'') = (L(n_k) \setminus \{\varphi \mathcal{U} \psi\}) \cup \{\varphi, \neg\psi, \circ((\varphi \wedge \neg\Delta) \mathcal{U} \psi)\}$ where $\Delta = L(n_k) \setminus \{\varphi \mathcal{U} \psi\}$. Extend selfun and $R$ to be $\mathsf{selfun}(n') = \emptyset$, $\mathsf{selfun}(n'') = \{\circ((\varphi \wedge \neg\Delta) \mathcal{U} \psi)\}$ and $n_k R n', n_k R n''$.

$\mathsf{unnext\_expand}(b, \mathcal{T}_\Phi, \mathsf{selfun})$ creates a new node $n$ and a new branch $b' = b \cdot n$ such that $L(n) = \mathsf{unnext}(L(n_k))$ and extends selfun and $R$ to be $\mathsf{selfun}(n) = \mathsf{unnext}(\mathsf{selfun}(n_k))$ and $n_k R n$ where $n_k = \mathsf{last\_node}(b)$.

$\mathsf{unmarked\_branches}(B)$ returns the set of unmarked maximal branches in a given set of branches $B$.

$\mathsf{fair\_select}(b, \mathcal{T}_\Phi, \mathsf{selfun})$ selects an until-formula, if there is some in the last node of $b$. That is, for $n_k = \mathsf{last\_node}(b)$, whenever $\mathsf{selfun}(n_k) = \emptyset$ and $L(n_k)$ contains at least one until-formula, it updates $\mathsf{selfun}(n_k)$ with a singleton $\{\varphi \mathcal{U} \psi\}$ such that $\varphi \mathcal{U} \psi \in L(n_k)$. Otherwise, $\mathsf{selfun}(n_k)$ remains the empty set. If the node contains more than one until-formula, the selection performed by fair_select on $L(n_k)$ should be *fair*, in the sense that no until-formula that from some stage onwards appears as an eligible eventuality whenever an eventuality must be selected, could remain non-selected indefinitely.

Let us give some useful results about the systematic tableau $\mathcal{T}_\Phi$ that this algorithm constructs for any set of formulas $\Phi$.

**Proposition 3.4.3.** *If $\{\varphi, \neg\varphi\} \subseteq L(s)$ for some stage $s$ in a branch $b$ of $\mathcal{T}_\Phi$, then every maximal branch of $\mathcal{T}_\Phi$ prefixed by $b$ is closed.*

*Proof.* By structural induction on $\varphi$. It is easy to see that the application of TTM-rules to two complementary formulas that belong to the same stage, but not necessarily to the same node, should generate complementary constituents until they occur in the same node or, at most, they become elementary. ∎

In the next proposition we show that non-satisfied unselected eventualities are kept in branches at least until they are fulfilled or they become selected.

**Proposition 3.4.4.** *Let $b$ be a branch*[6] *of $\mathcal{T}_\Phi$, and $s_0, s_1, s_2, \ldots, s_k$ be any initial subsequence of* $\mathsf{path}(\mathsf{stages}(b))$. *If the set $L(\mathsf{last}(s_k))$ is elementary, and $\varphi\,\mathcal{U}\,\psi \in L(s_i)$ for some $i \in \{0, \ldots, k\}$, and $\varphi\,\mathcal{U}\,\psi$ is not selected in the sequence $s_i, \ldots, s_k$, and $\psi \notin L(s_i) \cup \ldots \cup L(s_k)$, then $\{\varphi, \neg\psi, \circ(\varphi\,\mathcal{U}\,\psi)\} \subseteq L(s_j)$ for all $j \in \{i, \ldots, k\}$.*

*Proof.* By the construction of $\mathcal{T}_\Phi$, since non-selected eventualities are handled by procedure non-select_expand using the rule $(\,\mathcal{U}\,)_1$ . ■

It is worth noting that in the above Proposition 3.4.4, the requirement of $L(\mathsf{last}(s_k))$ being elementary is necessary. In order to illustrate this fact, let us consider the set of formulas $\{p\,\mathcal{U}\,q, \Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \Box(r\,\mathcal{U}\,v)\}$ and the branch $b = n_0, n_1, n_2$ such that

$$\begin{aligned} L(n_0) &= \{p\,\mathcal{U}\,q, \Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \Box(r\,\mathcal{U}\,v)\} \\ L(n_1) &= \{p\,\mathcal{U}\,q, \circ\Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \Box(r\,\mathcal{U}\,v)\} \\ L(n_2) &= \{p\,\mathcal{U}\,q, \circ\Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \circ\Box(r\,\mathcal{U}\,v)\} \end{aligned}$$

where, additionally, $\mathsf{selfun}(n_0) = \mathsf{selfun}(n_1) = \mathsf{selfun}(n_2) = \{p\,\mathcal{U}\,q\}$. The branch $b$ contains only one stage $s_0 = n_0, n_1, n_2$ and $L(n_2)$ is non-elementary. For the sequence of stages $s_0$ it holds that $r\,\mathcal{U}\,v \in L(s_0)$, $r\,\mathcal{U}\,v$ is not selected in $s_0$ and $v \notin L(s_0)$. However, $\{r, \neg v, \circ(r\,\mathcal{U}\,v)\}$ is not a subset of $L(s_0)$.

Next, we give a more detailed description of the syntactic form of the formulas appearing in sequences of stages where a selected eventuality remains unfulfilled. Under that proviso, at each stage, there is exactly one selected eventuality and exactly one node to which the procedure select_expand is applied. We also call this node the *selected node* of that stage. The fact that, at each stage, there is exactly one selected eventuality and exactly one node to which the procedure select_expand is applied, is crucial for defining the notion of closure with respect to which TTM satisfies the WASP. We first define some auxiliary sets of sub- and super-formulas of a given set of formulas $\Phi$. Let $\mathsf{sf}(\Phi)$ denote the set of all the subformulas of the formulas in $\Phi$ and their negations. Then, the preclosure of $\Phi$, $\mathsf{preclo}(\Phi)$, is the set of formulas that extends $\mathsf{sf}(\Phi)$ with all the superformulas that are generated from $\mathsf{sf}(\Phi)$ by means of all the TTM-rules with the exception of the rule $(\,\mathcal{U}\,)_2$. That is

$$\begin{aligned} \mathsf{preclo}(\Phi) = \mathsf{sf}(\Phi) \;\; &\cup \;\; \{\circ(\varphi\,\mathcal{U}\,\psi), \neg\circ(\varphi\,\mathcal{U}\,\psi), \circ\neg(\varphi\,\mathcal{U}\,\psi) \mid \varphi\,\mathcal{U}\,\psi \in \mathsf{sf}(\Phi)\} \\ &\cup \;\; \{\circ\neg\varphi \mid \neg\circ\varphi \in \mathsf{sf}(\Phi)\}. \end{aligned}$$

Note that $\mathsf{preclo}(\Phi)$ cannot be used as closure only because it does not capture the superformulas generated by the application of the rule $(\,\mathcal{U}\,)_2$. In order to capture these superformulas, we define the following set of conjunctions of negated contexts:

$$\begin{aligned} \mathsf{conj}(\Phi) \;\; &= \;\; \{\textstyle\bigwedge\Gamma \mid \Gamma \subseteq \{\varphi \mid \varphi\,\mathcal{U}\,\psi \in \mathsf{sf}(\Phi)\} \cup \mathsf{negctx}(\Phi)\} \\ &\qquad \text{where } \mathsf{negctx}(\Phi) = \{\neg\Delta \mid \Delta \subseteq \mathsf{preclo}(\Phi)\} \end{aligned}$$

That is, $\mathsf{negctx}(\Phi)$ is the set of all possible *negated contexts* and $\mathsf{conj}(\Phi)$ is formed by all the possible conjunctions of formulas in $\mathsf{negctx}(\Phi)$ and the left-hand side subformulas of all the until-formulas in $\mathsf{sf}(\Phi)$. In particular, $\mathrm{F} \in \mathsf{negctx}(\Phi)$ and $\mathrm{F}, \neg\mathrm{F} \in \mathsf{conj}(\Phi)$, since $\mathrm{F}$ and $\neg\mathrm{F}$ are respectively the disjunction and the conjunction of the empty set of formulas. Note also that, by definition, in the conjunctions of $\mathsf{conj}(\Phi)$ every element of $\mathsf{negctx}(\Phi)$ occurs at most once.

[6] The branch $b$ could be cyclic or not, so that $\mathsf{path}(\mathsf{stages}(b))$ could respectively be infinite or finite.

**Proposition 3.4.5.** *Let $b$ be a branch*[6] *of $\mathcal{T}_\Phi$, let $s_0, s_1, s_2, \ldots, s_k$ be any initial subsequence of* $\mathsf{path}(\mathsf{stages}(b))$ *and $\varphi \,\mathcal{U}\, \psi \in \mathsf{sf}(\Phi)$ such that $i$ is the least natural number such that* $\mathsf{selfun}(n) = \{\varphi \,\mathcal{U}\, \psi\}$ *for some $n \in s_i$. If the label of the last node of $b$ is an elementary set and $\psi \notin L(s_i) \cup \ldots \cup L(s_k)$, then for all $\ell \in \{0, \ldots, k-i\}$:*

$$\{\delta_\ell, \neg\psi, \circ(\delta_{\ell+1} \,\mathcal{U}\, \psi)\} \subseteq L(s_{i+\ell})$$

*where $\delta_0 = \varphi$ and $\delta_{\ell+1} = \delta_\ell \wedge \chi$ for some $\chi \in$* $\mathsf{negctx}(\Phi)$. *Moreover, if $\delta_\ell = \bigwedge \Gamma$ for some $\Gamma$ such that $\chi \in \Gamma$ then every maximal branch of $\mathcal{T}_\Phi$ prefixed by $s_0, \ldots, s_{i+\ell}$ is closed.*

*Proof.* Since the label of the last node of $b$ is elementary, we can ensure, by Definition 3.3.8, that no node of $b$ belongs to two different stages. Consequently, if a stage $s_h$, with $h \in \{i, \ldots, k\}$, contains a node $m$ such that $\mathsf{selfun}(m)$ is a singleton formed by an eventuality of the form $\psi_1 \,\mathcal{U}\, \psi_2$, then, by construction of $\mathcal{T}_\Phi$, $s_h$ also contains a node $m'$ (generated later than $m$) whose label has been obtained by application of the rule $(\,\mathcal{U}\,)_2$.

On one hand, the procedure select_expand yields two branches such that each branch either contains $\{\delta_\ell, \neg\psi, \circ(\delta_{\ell+1} \,\mathcal{U}\, \psi)\}$ or contains $\psi$. Note that, by construction of $\mathcal{T}_\Phi$, if $\mathsf{selfun}(n) = \{\circ(\delta_{\ell+1} \,\mathcal{U}\, \psi)\}$ for some $n \in s_{i+\ell}$, then $\mathsf{selfun}(n') = \{\delta_{\ell+1} \,\mathcal{U}\, \psi\}$ for the first node $n' \in s_{i+\ell+1}$, for all $\ell \in \{0, \ldots, k-i-1\}$. Therefore, $\delta_0 = \varphi$ and for all $j > 0$: $\delta_j = \delta_{j-1} \wedge \neg\Delta_{j-1}$ where $\neg\Delta_{j-1} \in \mathsf{negctx}(\Phi)$ and $\Delta_{j-1}$ is the context $L(n) \setminus \mathsf{selfun}(n)$ of the selected node $n$ of the stage $s_{i+j-1}$. Hence, by induction on $\ell$, $\delta_\ell \in \mathsf{conj}(\Phi)$ holds for all $\ell \in \{0, \ldots, k-i\}$.

On the other hand, since $\chi$ is the negation of the context of the selected node $n \in s_{i+\ell}$, if $\delta_{\ell+1} = \delta_\ell \wedge \chi$ and $\delta_\ell = \bigwedge \Gamma$ for some $\Gamma$ such that $\chi \in \Gamma$, then every branch prefixed by $s_0, \ldots, s_{i+\ell}$ contains at the same stage (possibly at different nodes) $\{\gamma, \neg\gamma\}$ for some formula $\gamma$. Hence, by Proposition 3.4.3, every maximal branch prefixed by $s_0, \ldots, s_{i+\ell}$ is closed. ■

It is worth noting that if in the above Proposition 3.4.5 the label of the last node of the branch $b$ is non-elementary, then the result is not guaranteed. In order to illustrate this fact, let us consider the set of formulas $\{p \,\mathcal{U}\, q, \circ\Box(p \,\mathcal{U}\, q), r \,\mathcal{U}\, v, \circ\Box(r \,\mathcal{U}\, v)\}$ and the branch $b = n_0, n_1, n_2, n_3, n_4, n_5$ such that

$$\begin{aligned}
L(n_0) &= \{p \,\mathcal{U}\, q, \circ\Box(p \,\mathcal{U}\, q), r \,\mathcal{U}\, v, \circ\Box(r \,\mathcal{U}\, v)\}\\
L(n_1) &= \{p \,\mathcal{U}\, q, \circ\Box(p \,\mathcal{U}\, q), r, \neg v, \circ(r \,\mathcal{U}\, v), \circ\Box(r \,\mathcal{U}\, v)\}\\
L(n_2) &= \{q, \circ\Box(p \,\mathcal{U}\, q), r, \neg v, \circ(r \,\mathcal{U}\, v), \circ\Box(r \,\mathcal{U}\, v)\}\\
L(n_3) &= \{\Box(p \,\mathcal{U}\, q), r \,\mathcal{U}\, v, \Box(r \,\mathcal{U}\, v)\}\\
L(n_4) &= \{\Box(p \,\mathcal{U}\, q), r \,\mathcal{U}\, v, \circ\Box(r \,\mathcal{U}\, v)\}\\
L(n_5) &= \{p \,\mathcal{U}\, q, \circ\Box(p \,\mathcal{U}\, q), r \,\mathcal{U}\, v, \circ\Box(r \,\mathcal{U}\, v)\}
\end{aligned}$$

where, additionally, $\mathsf{selfun}(n_0) = \mathsf{selfun}(n_1) = \{p \,\mathcal{U}\, q\}$, $\mathsf{selfun}(n_2) = \emptyset$ and $\mathsf{selfun}(n_3) = \mathsf{selfun}(n_4) = \mathsf{selfun}(n_5) = \{r \,\mathcal{U}\, v\}$. The branch $b$ gives rise to two stages $s_0 = n_0, n_1, n_2$ and $s_1 = n_3, n_4, n_5, n_1, n_2$. If we consider the sequence of stages $s_1$, it holds that $\mathsf{selfun}(n_3) = \{r \,\mathcal{U}\, v\}$ and $v \notin L(s_1)$. However, $L(s_1)$ does not contain the formula $\circ(\delta_1 \,\mathcal{U}\, v)$ mentioned in Proposition 3.4.5.

**Corollary 3.4.6.** *If $b$ is a cyclic branch of $\mathcal{T}_\Phi$ and the label of the last node of $b$ is elementary, then every selected eventuality in $b$ is fulfilled.*

*Proof.* By Proposition 3.4.5 since, whenever there is an unfulfilled selected eventuality in a branch, the presence of the formulas $\delta_\ell$ makes impossible the existence of a loop. ■

It is trivial, by construction, that every stage in a cyclic branch of $\mathcal{T}_\Phi$ is $\alpha\beta$-saturated. Hence, by Proposition 3.4.4 and Corollary 3.4.6, we can refine the sufficient conditions for being an expanded branch of $\mathcal{T}_\Phi$ (see Definition 3.3.12) as follows

**Proposition 3.4.7.** *Let $b$ be an open branch of $\mathcal{T}_\Phi$, if $b$ satisfies the following three conditions:*

*(i) $b$ is cyclic*

*(ii) for every eventuality $\gamma \in \mathsf{preclo}(\Phi)$ such that $\gamma \in L(\mathsf{first}(s))$ for all $s \in \mathsf{stages}(\mathsf{cycle}(b))$, there exists some $s' \in \mathsf{stages}(\mathsf{cycle}(b))$ such that $\mathsf{selfun}(\mathsf{first}(s')) = \{\gamma\}$*

*(iii) the label of the last node of $b$ is elementary*

*then $b$ is an expanded branch.*

*Proof.* By Proposition 3.4.4, non-selected unfulfilled eventualities are preserved from one stage to its successor. In addition, by Corollary 3.4.6, every selected eventuality in a cyclic branch whose last node is labelled by an elementary set, is fulfilled. Hence, by condition (ii), every eventuality from $\mathsf{preclo}(\Phi)$ that occurs in $L(\mathsf{first}(s))$ for every $s \in \mathsf{stages}(\mathsf{cycle}(b))$ should be selected (at least) once and, hence, should be fulfilled. ■

Consequently, we use the three conditions in Proposition 3.4.7 to refine the implementation of the procedure unmarked_branches

**Remark 3.4.8.** *Whenever a branch $b$ satisfies conditions (i), (ii) and (iii) of Proposition 3.4.7, the procedure* unmarked_branches *considers $b$ to be marked as expanded.*

Note that a branch can satisfy the conditions stated in Definition 3.3.12 without satisfying conditions (i), (ii) and (iii) of Proposition 3.4.7. This means that sometimes the systematic algorithm does not detect that a branch is already expanded and goes on extending it until conditions (i), (ii) and (iii) of Proposition 3.4.7 are satisfied. For example, an expanded branch for the set $\{p\,\mathcal{U}\,q, \circ\Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \circ\Box(r\,\mathcal{U}\,v)\}$ is given by the sequence of nodes $b = n_0, n_1, n_2, n_3, n_4, n_5$ such that

$$\begin{aligned}
L(n_0) &= \{p\,\mathcal{U}\,q, \circ\Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \circ\Box(r\,\mathcal{U}\,v)\}\\
L(n_1) &= \{p\,\mathcal{U}\,q, \circ\Box(p\,\mathcal{U}\,q), v, \circ\Box(r\,\mathcal{U}\,v)\}\\
L(n_2) &= \{q, \circ\Box(p\,\mathcal{U}\,q), v, \circ\Box(r\,\mathcal{U}\,v)\}\\
L(n_3) &= \{\Box(p\,\mathcal{U}\,q), \Box(r\,\mathcal{U}\,v)\}\\
L(n_4) &= \{\Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \circ\Box(r\,\mathcal{U}\,v)\}\\
L(n_5) &= \{p\,\mathcal{U}\,q, \circ\Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \circ\Box(r\,\mathcal{U}\,v)\}
\end{aligned}$$

where $\mathsf{selfun}(n_0) = \mathsf{selfun}(n_1) = \{p\,\mathcal{U}\,q\}$, $\mathsf{selfun}(n_2) = \mathsf{selfun}(n_3) = \emptyset$ and $\mathsf{selfun}(n_4) = \mathsf{selfun}(n_5) = \{r\,\mathcal{U}\,v\}$. The branch $b$ gives rise to two stages $s_0 = n_0, n_1, n_2$ and $s_1 = n_3, n_4, n_5, n_1, n_2$. The branch $b$ is cyclic, fulfilling and the stages are $\alpha\beta$-saturated. Consequently, $b$ satisfies the conditions in Definition 3.3.12, but $b$ does not satisfy condition (iii) in Proposition 3.4.7 and consequently the algorithm has to enlarge the branch. For instance, the systematic algorithm can build the branch $b \cdot n_6, n_7$ such that

$$L(n_6) = \{q, \circ\Box(p\,\mathcal{U}\,q), r\,\mathcal{U}\,v, \circ\Box(r\,\mathcal{U}\,v)\}$$
$$L(n_7) = \{q, \circ\Box(p\,\mathcal{U}\,q), v, \circ\Box(r\,\mathcal{U}\,v)\}$$

and $\mathsf{selfun}(n_6) = \{r\,\mathcal{U}\,v\}$ and $\mathsf{selfun}(n_7) = \emptyset$. The stages of the branch $b \cdot n_6, n_7$ are $x_0 = n_0, n_1, n_2$ and $x_1 = n_3, n_4, n_5, n_6, n_7$, additionally, $\mathsf{path}(\mathsf{stages}(b)) = x_0 \cdot \langle x_1\rangle^\omega$. The branch $b \cdot n_6, n_7$ satisfies conditions (i), (ii) and (iii) in Proposition 3.4.7.

By Corollary 3.4.6 and Remark 3.4.8, TTM satisfies the WASP with respect to the following notion of closure.

**Definition 3.4.9.** *Let $\Phi$ be a set of formulas. The closure of $\Phi$, $\mathsf{clo}(\Phi)$, is the following set of formulas:*

$$\mathsf{clo}(\Phi) = \mathsf{preclo}(\Phi) \cup \mathsf{conj}(\Phi) \cup \Omega$$
*where*
$$\Omega = \{(\gamma_1 \wedge \gamma_2)\,\mathcal{U}\,\psi, \circ((\gamma_1 \wedge \gamma_2)\,\mathcal{U}\,\psi) \mid \varphi\,\mathcal{U}\,\psi \in \mathsf{sf}(\Phi) \text{ and } \gamma_1, \gamma_2 \in \mathsf{conj}(\Phi)\}$$

Since in the systematic tableaux the formulas of the form

$$\circ((\varphi \wedge \neg\Delta_0 \wedge \neg\Delta_1 \wedge \ldots \wedge \neg\Delta_k)\,\mathcal{U}\,\psi)$$

built up by using the rule $(\mathcal{U})_2$ can only contain one repetition of a negated context, i.e., since there can only exist at most two values $g$ and $h$ such that $1 \leq g < h \leq k$ and $\neg\Delta_g = \neg\Delta_h$, $\gamma_1$ and $\gamma_2$ are enough to represent such possible repetition of a negated context. In other words, $L(n) \subseteq \mathsf{clo}(\Phi)$ holds for all node $n$ in $\mathcal{T}_\Phi$, by Corollary 3.4.6 and Remark 3.4.8. In addition, the closure set of a finite set of formulas is finite.

**Proposition 3.4.10.** *If $\Phi$ is a finite set of formulas, then $\mathsf{clo}(\Phi)$ is also finite.*

*Proof.* It is easy to see that, if $|\mathsf{preclo}(\Phi)| = n$ then $|\mathsf{negctx}(\Phi)| \in O(2^n)$. As a consequence $|\mathsf{conj}(\Phi)|, |\mathsf{clo}(\Phi)| \in O(2^{O(2^n)})$. ■

The above results jointly with the fairness of fair_select, allows us to ensure that the algorithm in Figure 3.9 finitely computes an expanded tableau $\mathcal{T}_\Phi$ for any input $\Phi$.

**Lemma 3.4.11.** *The algorithm in Figure 3.9, for any input $\Phi$, stops leaving in $\mathcal{T}_\Phi$ an expanded tableau.*

*Proof.* By König's lemma, the only possibility for infinite iteration would be the infinite expansion of (at least) one branch, namely $b$. By Propositions 3.4.5, 3.4.7 and 3.4.10, the branch $b$ should contain an eventuality that is never selected, which contradicts the fairness of the fair_select procedure. ■

Note that the use of a fair strategy for selecting the eventualities in each branch of the tableau is essential for proving that the algorithm in Figure 3.9 finishes.

We would like to remark that previous tableau methods for PLTL, with the exception of the one-pass proposal of [117], for obtaining a model of a satisfiable set of formulas (when deciding satisfiability) should generate the whole graph of possible states and all the successive tableaux required for constructing this graph. However, we can use a depth-first strategy and, as soon as a branch is marked expanded, the algorithm could stop providing a model for the original

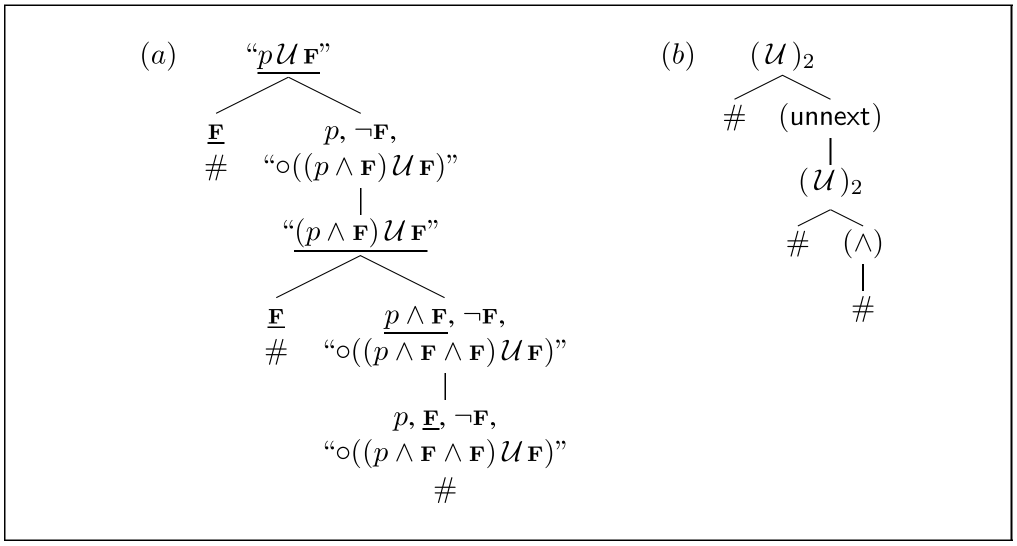

(a)
“$p\,\mathcal{U}\,\mathbf{F}$”
$\underline{\mathbf{F}}$
#
$p$, $\neg\mathbf{F}$,
“$\circ((p \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F})$”
“$(p \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F}$”
$\underline{\mathbf{F}}$
#
$p \wedge \mathbf{F}$, $\neg\mathbf{F}$,
“$\circ((p \wedge \mathbf{F} \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F})$”
$p$, $\underline{\mathbf{F}}$, $\neg\mathbf{F}$,
“$\circ((p \wedge \mathbf{F} \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F})$”
#
(b)
$(\mathcal{U})_2$
#
(unnext)
$(\mathcal{U})_2$
#
$(\wedge)$
#

*Figure 3.10:* Systematic closed tableau for the set of formulas $\{p\,\mathcal{U}\,\mathbf{F}\}$

set of formulas. It is also worth noting that in the tableau calculus introduced by Schwendimann in [117], the fulfillment of eventualities may depend on more than one cyclic branch, and consequently, unlike in TTM, a fully expanded cyclic branch may not yield a model by itself.

### 3.4.3 Examples of Systematic Tableaux

In this subsection, we give four expanded tableaux built by using the systematic tableau algorithm in Figure 3.9. In order to show each tableau we follow the same notation as in Subsection 3.3.5. The only difference is that in the systematic tableaux, we also manage the selection function selfun. So that, the formulas selected by the function selfun appear between the quotation marks “ and ”. When in a node of the $(a)$ part there is a formula $\chi$ between the quotation marks “ and ”, i.e. “$\chi$”, that means that the value of selfun for such node is $\{\chi\}$. If a node does not contain any formula between the quotation marks, then the value of selfun for such node is $\emptyset$.

In the first two examples we provide the systematic expanded tableaux that correspond to the tableaux showed in Example 3.3.15 (Figure 3.3) and Example 3.3.16 (Figure 3.5) in Section 3.3.5.

**Example 3.4.12.** *In Figure 3.10 the systematic expanded tableau for the unsatisfiable set of formulas $\{p\,\mathcal{U}\,\mathbf{F}\}$ is showed. This tableau is closed.*

*By following the algorithm for systematic tableau construction, the only available eventuality, $p\,\mathcal{U}\,\mathbf{F}$, is selected. Hence the value of the selection function* selfun *for the first node is $\{p\,\mathcal{U}\,\mathbf{F}\}$. Then the $\beta$-rule $(\mathcal{U})_2$ is applied to the formula $p\,\mathcal{U}\,\mathbf{F}$ with the empty set of formulas as context. The application of $(\mathcal{U})_2$ splits the branch into two branches. The branch on the left is closed because the label of its last node contains $\mathbf{F}$. For the branch on the right, the label of the new node contains the new formula $\circ((p \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F})$. The formula $\mathbf{F}$ that appears on the left hand-side of the formula $\circ((p \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F})$ corresponds to the negation of the empty set of formulas. The value of the selection function* selfun *for this second node in the branch on the right is $\{\circ(p\,\mathcal{U}\,\mathbf{F})\}$. Since the label is elementary, the operator* unnext *is applied in order to jump to*

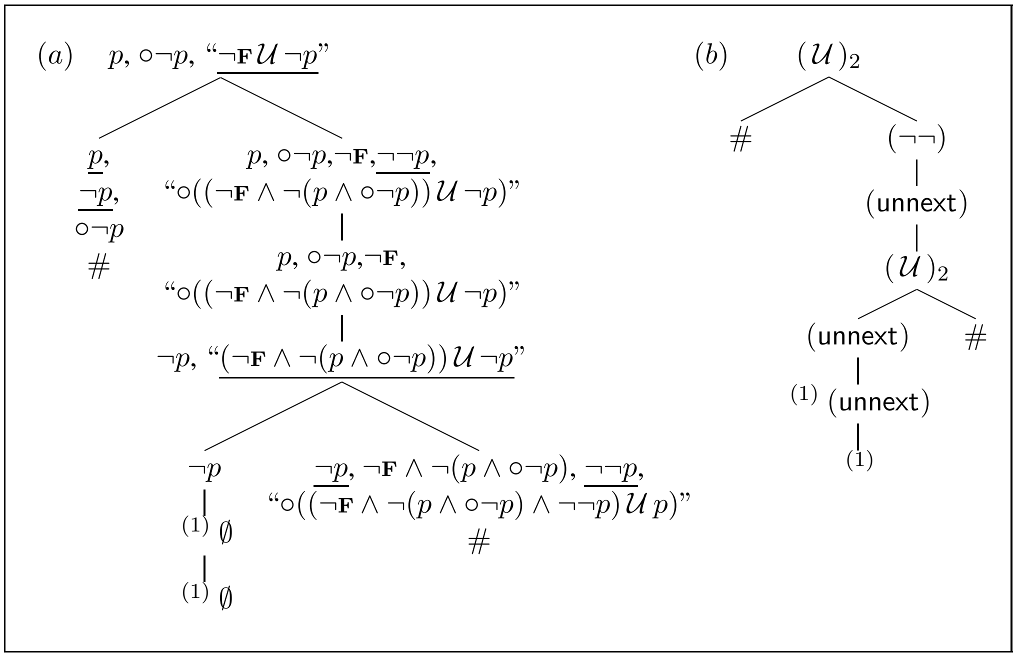

(a) p, ○¬p, "¬F 𝒰 ¬p"
p, ¬p, ○¬p #
p, ○¬p, ¬F, ¬¬p, "○((¬F ∧ ¬(p ∧ ○¬p)) 𝒰 ¬p)"
p, ○¬p, ¬F, "○((¬F ∧ ¬(p ∧ ○¬p)) 𝒰 ¬p)"
¬p, "(¬F ∧ ¬(p ∧ ○¬p)) 𝒰 ¬p"
¬p
(1) ∅
(1) ∅
¬p, ¬F ∧ ¬(p ∧ ○¬p), ¬¬p, "○((¬F ∧ ¬(p ∧ ○¬p) ∧ ¬¬p) 𝒰 p)" #
(b) (𝒰)2
#
(¬¬)
(unnext)
(𝒰)2
(unnext)
#
(1) (unnext)
(1)

*Figure 3.11:* Systematic expanded tableau for the set of formulas $\{p, \circ\neg p, \neg\mathbf{F}\,\mathcal{U}\,\neg p\}$

*the next state. The value of the selection function* selfun *for the new node (the third one in this branch) is* $\{(p \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F}\}$*. By applying the rule* $(\mathcal{U})_2$*, the branch is split into a closed branch on the left and a branch with a new node whose label contains the formula* $\circ((p \wedge \mathbf{F} \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F})$*. The second* $\mathbf{F}$ *from the left, in the formula* $\circ((p \wedge \mathbf{F} \wedge \mathbf{F})\,\mathcal{U}\,\mathbf{F})$*, corresponds to the negation of the empty set of formulas, which was the context in this second application of the rule* $(\mathcal{U})_2$*. Finally by applying the* $\alpha$*-rule* $(\wedge)$ *to the formula* $p \wedge \mathbf{F}$*, an inconsistent node is generated.*

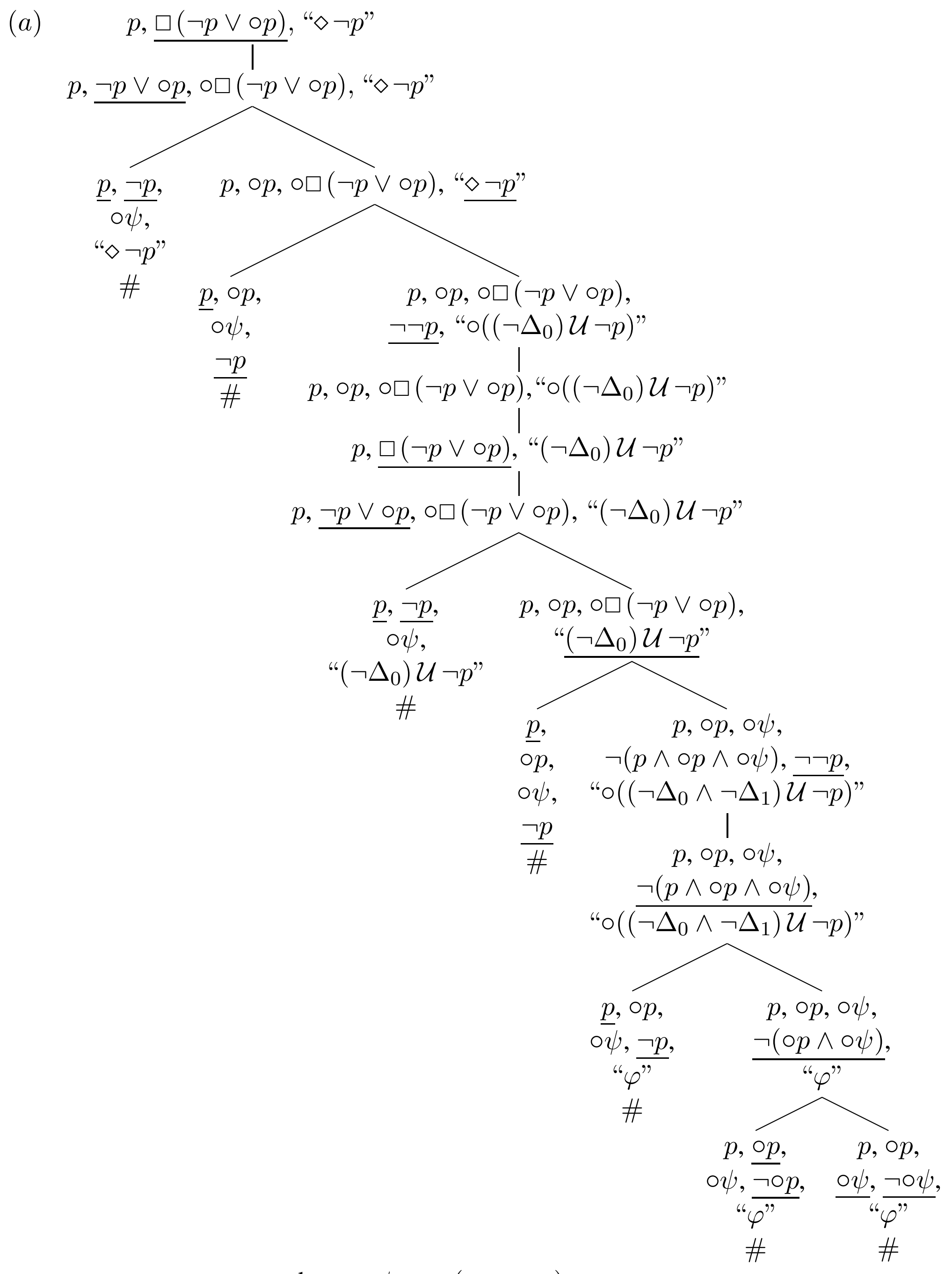
*It is worth noting that in the construction of the tableau in Figure 3.3 the rules* $(\mathcal{U})_1$ *and* $(\mathcal{U})_2$ *are used whereas the systematic tableau in Figure 3.10 does not include any application of the rule* $(\mathcal{U})_1$*.*

**Example 3.4.13.** *In Figure 3.11 we provide the systematic expanded tableau for the satisfiable set of formulas* $\{p, \circ\neg p, \neg\mathbf{F}\,\mathcal{U}\,\neg p\}$*. In the first application of the rule* $(\mathcal{U})_2$*, the context is* $\{p, \circ\neg p\}$ *and in the second application of the rule* $(\mathcal{U})_2$*, the context is* $\{\neg p\}$*. The negations of these two sets of formulas are used to generate, respectively, the formulas* $\circ((\neg\mathbf{F} \wedge \neg(p \wedge \circ\neg p))\,\mathcal{U}\,p)$ *and* $\circ((\neg\mathbf{F} \wedge \neg(p \wedge \circ\neg p) \wedge \neg\neg p)\,\mathcal{U}\,p)$ *obtained by means of the two applications of the rule* $(\mathcal{U})_2$*. The central open branch represents the collection of models explained in Example 3.3.16.*

Note that the formula $\neg\mathbf{F}\,\mathcal{U}\,\neg p$ can also be expressed as $\diamond\neg p$.

The next two examples are related to the induction on time. These examples illustrate the use of both the derived rule $(\diamond)_2$ in Figure 3.2 and the rule $(\mathcal{U})_2$ in Figure 3.1.

**Example 3.4.14.** *In Figure 3.12 and Figure 3.13 we depict a systematic closed tableau for the set* $\Phi = \{p, \Box(\neg p \vee \circ p), \diamond\neg p\}$*. The subset* $\Sigma = \{p, \Box(\neg p \vee \circ p)\}$ *states, by means of the so-called* induction on time*, that* $\Box p$ *holds. Hence,* $\Phi$ *is unsatisfiable. Note that the formula* $\bigwedge \Sigma$


(a)
p, □(¬p ∨ ○p), "◇¬p"
p, ¬p ∨ ○p, ○□(¬p ∨ ○p), "◇¬p"
p, ¬p, ○ψ, "◇¬p" #
p, ○p, ○□(¬p ∨ ○p), "◇¬p"
p, ○p, ○ψ, ¬p #
p, ○p, ○□(¬p ∨ ○p), ¬¬p, "○((¬Δ0) U ¬p)"
p, ○p, ○□(¬p ∨ ○p), "○((¬Δ0) U ¬p)"
p, □(¬p ∨ ○p), "(¬Δ0) U ¬p"
p, ¬p ∨ ○p, ○□(¬p ∨ ○p), "(¬Δ0) U ¬p"
p, ¬p, ○ψ, "(¬Δ0) U ¬p" #
p, ○p, ○□(¬p ∨ ○p), "(¬Δ0) U ¬p"
p, ○p, ○ψ, ¬p #
p, ○p, ○ψ, ¬(p ∧ ○p ∧ ○ψ), ¬¬p, "○((¬Δ0 ∧ ¬Δ1) U ¬p)"
p, ○p, ○ψ, ¬(p ∧ ○p ∧ ○ψ), "○((¬Δ0 ∧ ¬Δ1) U ¬p)"
p, ○p, ○ψ, ¬p, "φ" #
p, ○p, ○ψ, ¬(○p ∧ ○ψ), "φ"
p, ○p, ○ψ, ¬○p, "φ" #
p, ○p, ○ψ, ¬○ψ, "φ" #

where $\psi = \Box(\neg p \vee \circ p)$
$\Delta_0 = \Delta_1 = \{p, \circ p, \circ\psi\}$
$\neg\Delta_0 = \neg\Delta_1 = \neg(p \wedge \circ p \wedge \circ\psi)$
$\varphi = \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)$

*Figure 3.12:* Systematic closed tableau for $\{p, \Box(\neg p \vee \circ p), \Diamond\neg p\}$ by using $(\Diamond)_2$ and $(\mathcal{U})_2$ (Part 1 of 2)

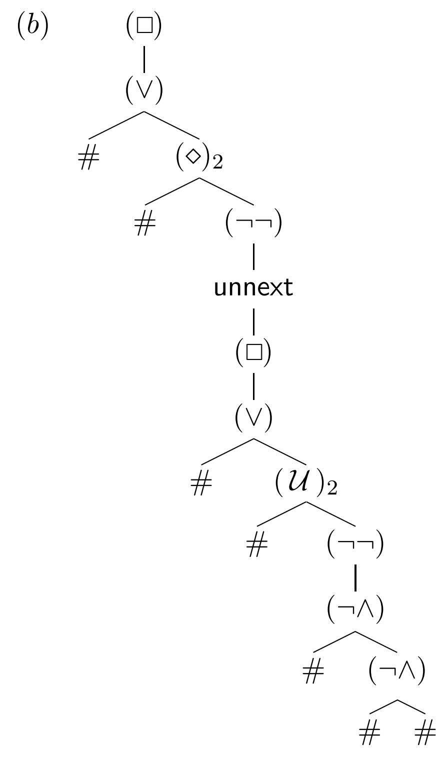


*Figure 3.13:* Systematic closed tableau for $\{p, \Box(\neg p \vee \circ p), \Diamond\neg p\}$ by using $(\Diamond)_2$ and $(\mathcal{U})_2$ (Part 2 of 2)

*is an invariant that contradicts the eventuality $\Diamond\neg p$. The sets of formulas that label the nodes appear in Figure 3.12 whereas the rules applied at each step appear in Figure 3.13.*

*The algorithm for systematic tableau construction, first selects the only available eventuality $\Diamond\neg p$. So the value of the selection function* selfun *for the first node is $\{\Diamond\neg p\}$. Then the $\alpha$-rule $(\Box)$ is applied to the formula $\Box(\neg p \vee \circ p)$ enlarging the branch with a new node (the second one). In the second node, the $\beta$-rule $(\vee)$ is applied to the formula $\neg p \vee \circ p$ and two new nodes are generated. The one on the left is inconsistent and it yields a closed branch. In the one on the right every formula, with the exception of the selected eventuality, is elementary and consequently the rule $(\mathcal{U})_2$ is applied to $\Diamond\neg p$ with context $\Delta_0 = \{p, \circ p, \circ\Box(\neg p \vee \circ p)\}$. The application of the rule $(\mathcal{U})_2$ splits the brach by creating two new nodes. The one on the left is inconsistent and gives rise to another closed branch. For the new node on the right, the new value of the selection function* selfun *is $\{\circ((\neg\Delta_0)\,\mathcal{U}\,p)\}$. Since the set that labels the node on the right is non-elementary –because of the formula $\neg\neg p$– the $\alpha$-rule $(\neg\neg)$ is applied and a new node with elementary label is obtained. Consequently, the operator* unnext *is applied and the branch is enlarged with a new node. The value of the selection function* selfun *for this node is $\{(\neg\Delta_0)\,\mathcal{U}\,\neg p\}$. The following two steps are like the two initial steps, i.e., the $\alpha$-rule $(\Box)$ enlarges the branch and the $\beta$-rule $(\vee)$ splits the enlarged branch giving rise to a closed branch and a branch where only the selected eventuality $(\neg\Delta_0)\,\mathcal{U}\,\neg p$ is non-elementary. So the latter branch is split again by applying the rule $(\mathcal{U})_2$ with context $\Delta_1 = \Delta_0$. The node in the left is*

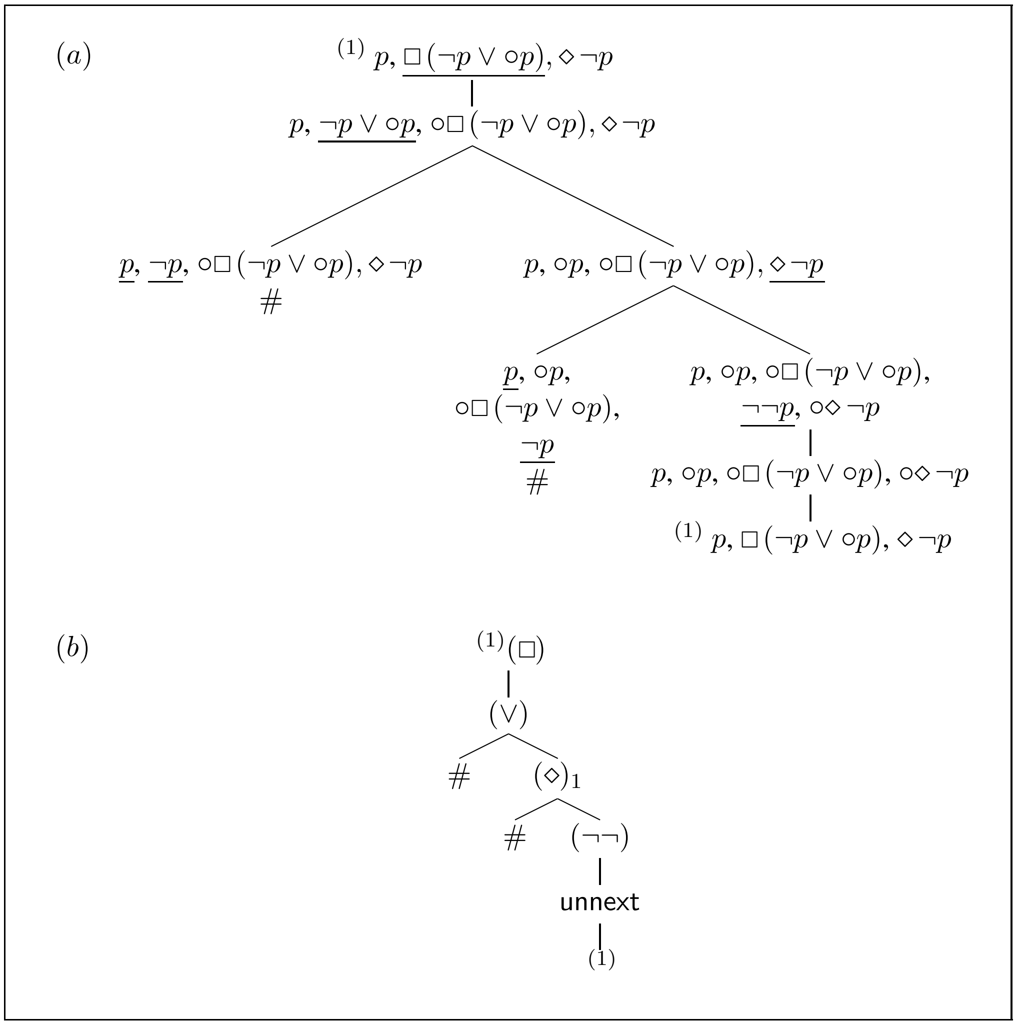

(a)
(1) $p, \underline{\Box(\neg p \vee \circ p)}, \Diamond\neg p$
$p, \underline{\neg p \vee \circ p}, \circ\Box(\neg p \vee \circ p), \Diamond\neg p$
$\underline{p}, \underline{\neg p}, \circ\Box(\neg p \vee \circ p), \Diamond\neg p$
#
$p, \circ p, \circ\Box(\neg p \vee \circ p), \underline{\Diamond\neg p}$
$\underline{p}, \circ p, \circ\Box(\neg p \vee \circ p), \underline{\neg p}$
#
$p, \circ p, \circ\Box(\neg p \vee \circ p), \underline{\neg\neg p}, \circ\Diamond\neg p$
$p, \circ p, \circ\Box(\neg p \vee \circ p), \circ\Diamond\neg p$
(1) $p, \Box(\neg p \vee \circ p), \Diamond\neg p$
(b)
(1)($\Box$)
($\vee$)
#
$(\Diamond)_1$
#
$(\neg\neg)$
unnext
(1)

*Figure 3.14:* Non-systematic and non-expanded open tableau for $\{p, \Box(\neg p \vee \circ p), \Diamond\neg p\}$

*inconsistent and the branch is closed. The label of the node on the right is non-elementary and the value of the selection function* selfun *is* $\{\circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,p)\}$. *Moreover, the repetition of the context, i.e.,* $\Delta_1 = \Delta_0$, *leads to inconsistency since the label of this node contains the formulas* $\{p, \circ p, \circ\psi, \neg(p \wedge \circ p \wedge \circ\psi)\}$. *First the branch is enlarged by means of the rule* $\neg\neg$ *and finally, two consecutive applications of the* $\beta$*-rule* $(\neg\wedge)$ *produce three closed branches.*

*It is worth noting that by using only the rule* $(\mathcal{U})_1$, *the fulfillment of an eventuality can be indefinitely delayed. As shown in Figure 3.14, the set* $\Phi = \{p, \Box(\neg p \vee \circ p), \Diamond\neg p\}$ *cannot be* TTM*-refuted without using the rules* $(\Diamond)_2$ *and* $(\mathcal{U})_2$. *In the third branch from the left, we obtain the initial set after applying the operator* unnext. *Although the branch is cyclic, it is not fulfilling, so it is not expanded. If the rules* $(\Diamond)_2$ *and* $(\mathcal{U})_2$ *are not properly used as shown in Figure 3.12 and Figure 3.13, the process will give rise to an infinite branch. Obviously, this derivation does not follow the algorithm for systematic tableau construction.*

**Example 3.4.15.** *In Figure 3.15 we depict a systematic expanded tableau for the satisfiable set*

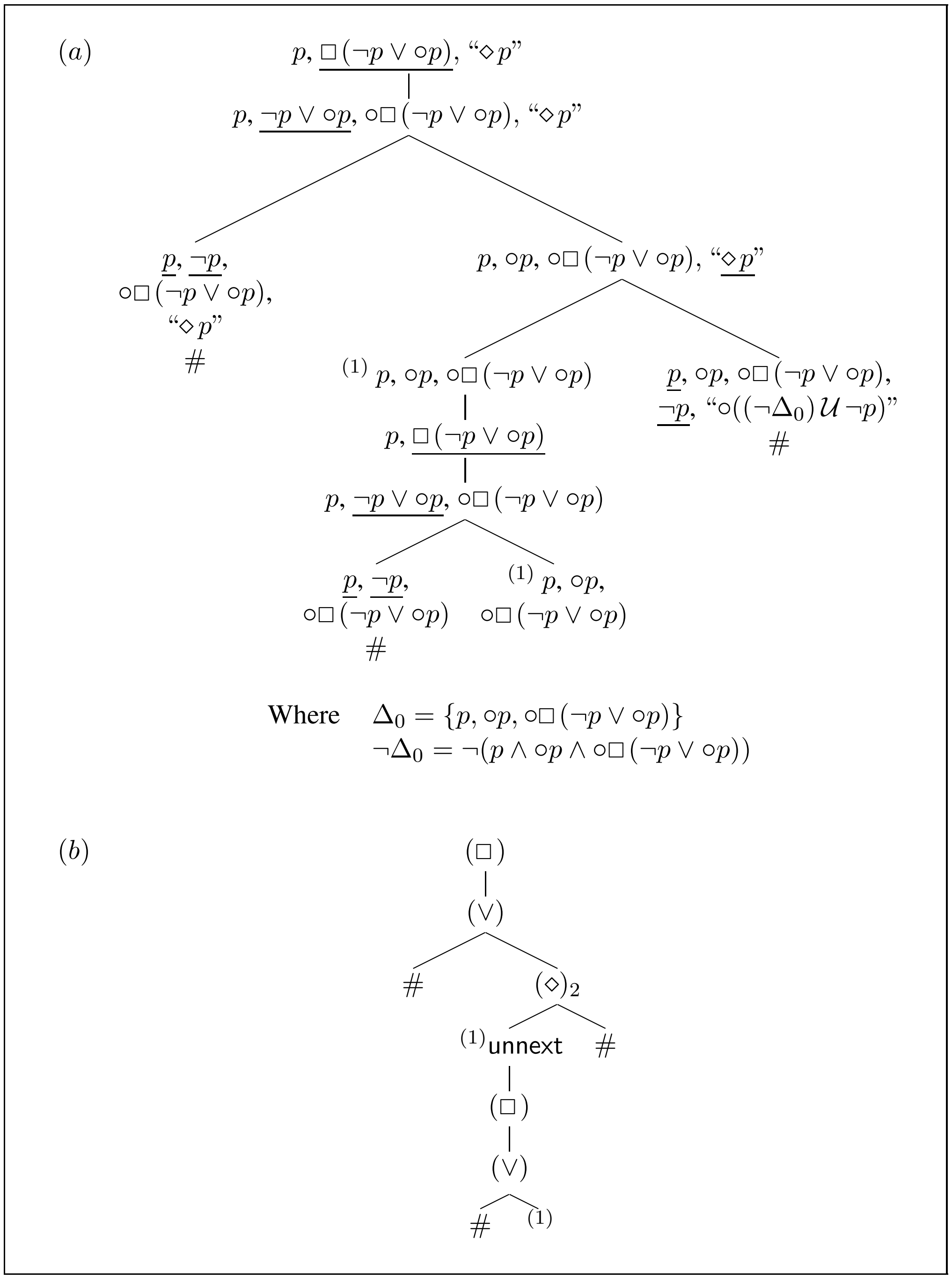


*Figure 3.15:* Systematic expanded tableau for $\{p, \Box(\neg p \vee \circ p), \Diamond p\}$ obtained by using $(\Diamond)_2$.

$\Psi = \{p, \Box(\neg p \vee \circ p), \Diamond p\}$. *The tableau is formed by four branches. Three of them are closed and one is open (the third one from the left). The open branch, let us call it b, contains seven nodes, so that, it is of the form* $n_0, \ldots, n_6$ *where* $L(n_3) = L(n_6)$. *The branch b is cyclic, the label of its last node is elementary and in the cycle there are no eventualities, hence the sufficient conditions for the algorithm to mark it as an expanded branch hold. We mark the leaf of b and the internal repeated node with the superscript* $^{(1)}$. *This open branch is formed by two stages* $x_0 = n_0, n_1, n_2, n_3$ *and* $x_1 = n_4, n_5, n_6$ *and* $\mathsf{path}(b) = x_0 \cdot \langle x_1 \rangle^\omega$ *which describes a model in which p is true in every state because* $p \in L(x_0)$ *and* $p \in L(x_1)$.

### 3.4.4 Completeness

In this subsection we prove the refutational completeness of TTM by showing that if $\Phi$ is satisfiable then we can associate to any expanded branch $b$ of the systematic tableau for $\Phi$ a cyclic PLTL-structure $\mathcal{G}_b$ that yields a model of $\Phi$.

**Definition 3.4.16.** *For any expanded branch b, we define the* PLTL*-structure* $\mathcal{G}_b = (S_{\mathcal{G}_b}, V_{\mathcal{G}_b})$ *such that* $S_{\mathcal{G}_b} = \mathsf{path}(\mathsf{stages}(b))$ *and* $V_{\mathcal{G}_b}(s) = \{p \mid p \in L(s)$ *and* $p \in \mathsf{Prop}\}$.

Note that termination of the systematic tableau construction is guaranteed by the finiteness of the closure (see Proposition 3.4.10) together with the fairness in selecting until-formulas. Consequently, since every maximal branch of $\mathcal{T}_\Phi$ is closed or expanded, then any expanded branch must have two nodes with the same label (see Remark 3.4.8) which necessarily belong to two different stages, since one stage cannot contain two identical nodes. Summarizing, any expanded branch of $\mathcal{T}_\Phi$ has at least two nodes, at least two stages, and is cyclic. In the rest of this subsection we will assume that $b = n_0, \ldots, n_k$ is an expanded branch of $\mathcal{T}_\Phi$, hence $b$ is cyclic, and that $\mathcal{G}_b$ is the cyclic PLTL-structure associated to $b$.

In the previous Subsection 3.4.2 we prove some properties about the behaviour of eventualities along the branches of $\mathcal{T}_\Phi$, that obviously can be applied to $\mathcal{G}_b$. The next proposition shows the behaviour of negated eventualities in $\mathcal{G}_b$.

**Proposition 3.4.17.** *Let* $s_j \in S_{\mathcal{G}_b}$ *such that* $\neg(\varphi \mathcal{U} \psi) \in L(s_j)$. *Then, every finite subsequence* $\pi = s_j, s_{j+1}, \ldots, s_k$ *of* $S_{\mathcal{G}_b}$ *satisfies one of the two following properties:*

*(a)* $\{\varphi, \neg\psi, \neg\circ(\varphi \mathcal{U} \psi)\} \subseteq L(s_i)$ *for all* $i \in \{j, \ldots, k\}$

*(b) There exists* $i \in \{j, \ldots, k\}$ *such that* $\{\neg\varphi, \neg\psi\} \subseteq L(s_i)$ *and* $\{\varphi, \neg\psi, \neg\circ(\varphi \mathcal{U} \psi)\} \subseteq L(s_\ell)$ *for all* $\ell \in \{j, \ldots, i-1\}$.

*Proof.* By induction on $k - j$. The case $k = j$ is trivial. For $k - j \geq 1$, the induction hypothesis guarantees that $\pi' = s_j, s_1, \ldots, s_{k-1}$ satisfies one of the properties *(a)* or *(b)*. If $\pi'$ satisfies *(b)*, so does $\pi$. If $\pi'$ satisfies *(a)* then, by $\alpha\beta$-saturation, we have $\{\varphi, \neg\psi, \neg\circ(\varphi \mathcal{U} \psi)\} \subseteq L(s_k)$ or $\{\neg\varphi, \neg\psi\} \subseteq L(s_k)$. Hence, $\pi$ verifies *(a)* or *(b)*, respectively. ■

Therefore, we can prove that each state of $\mathcal{G}_b$ satisfies its labels, that is the set of formulas labelling all nodes that constitute the concerned stage.

**Lemma 3.4.18.** *For every* $s \in S_{\mathcal{G}_b}$, *if* $\varphi \in L(s)$ *then* $\langle \mathcal{G}_b, s \rangle \models \varphi$.

*Proof.* By structural induction on $\varphi$. The case of literals is trivial by definition of $\mathcal{G}_b$.
For formulas of the form $\neg\neg\varphi$, $\varphi \wedge \psi$, $\neg(\varphi \wedge \psi)$, $\circ\varphi$ and $\neg\circ\varphi$ the property holds because every stage in $S_{\mathcal{G}_b}$ is $\alpha\beta$-saturated and the induction hypothesis on $\{\varphi\}$, $\{\varphi, \psi\}$, $\{\neg\varphi, \neg\psi\}$, $\{\varphi\}$ and $\{\neg\varphi\}$, respectively.
For $\varphi\,\mathcal{U}\,\psi$, by Propositions 3.4.4 and 3.4.5, there should exist a finite subsequence $s_0, s_1, \dots, s_n$ of $S_{\mathcal{G}_b}$ such that $s_0 = s, \psi \in s_n$ and $\varphi \in s_i$ for every $i \in \{0, \dots, n-1\}$. By the induction hypothesis, $\langle \mathcal{G}_b, s_n \rangle \models \psi$ and $\langle \mathcal{G}_b, s_i \rangle \models \varphi$ for every $i \in \{0, \dots, n-1\}$ and consequently $\langle \mathcal{G}_b, s \rangle \models \varphi\,\mathcal{U}\,\psi$.
For $\neg(\varphi\,\mathcal{U}\,\psi)$ formulas, by the above Propositions 3.4.3 and 3.4.17 and the induction hypothesis, there does not exist any finite path $s_0, s_1, \dots, s_n$ in $S_{\mathcal{G}_b}$ such that $s_0 = s$, $\langle \mathcal{G}_b, s_n \rangle \models \psi$ and $\langle \mathcal{G}_b, s_i \rangle \models \varphi$ for every $i \in \{0, \dots, n-1\}$. Consequently $\langle \mathcal{G}_b, s \rangle \not\models \varphi\,\mathcal{U}\,\psi$ and hence $\langle \mathcal{G}_b, s \rangle \models \neg(\varphi\,\mathcal{U}\,\psi)$. ■

**Corollary 3.4.19.** $\mathcal{G}_b \models \Phi$

*Proof.* Immediate consequence of Lemma 3.4.18. ■

By means of the collection of results proved in this section, we provide an alternative proof of the result that states that "every satisfiable set of PLTL-formulas has a cyclic model" (see Theorem 7.1 in [128] and Theorem 1 in [15]). Our proof is constructive in the sense that it gives a tableau-based procedure that constructs the cyclic model $\mathcal{G}_b$ for any satisfiable $\Phi$.

Now, we prove the refutational completeness of the tableau system TTM.

**Theorem 3.4.20.** *If $\Phi$ is unsatisfiable then there exists a closed tableau for $\Phi$.*

*Proof.* Suppose that it does not exist any closed TTM-tableau for $\Phi$. Then the systematic tableau $\mathcal{T}_\Phi$ would be open and there would be at least one expanded branch $b$ of $\mathcal{T}_\Phi$. By Corollary 3.4.19, $\mathcal{G}_b \models \Phi$. Consequently $\Phi$ would be satisfiable. ■

Moreover, the tableau method TTM is also complete.

**Theorem 3.4.21.** *If $\Phi$ is satisfiable then there exists a (finite) open expanded tableau for $\Phi$.*

*Proof.* The systematic tableau $\mathcal{T}_\Phi$ suffices to prove this fact. ■

Hence, the system TTM can be used as a satisfiability decision procedure for PLTL.

### *3.4.5 Improving Eventuality Handling*

The application of the rule $(\mathcal{U})_2$ builds up complex formulas that involve the whole context. Hence, for practical purposes, it is interesting to simplify these formulas as much as possible. In this subsection we are going to show some ideas for avoiding redundant formulas in the negated context produced by application of the rule $(\mathcal{U})_2$. That is, we introduce a new rule $(\mathcal{U})_3$ (see Figure 3.16) that is an improvement of $(\mathcal{U})_2$ that prevents two kinds of redundancy:

1. Disjuncts stating that the next stage fails to satisfy a formula which the context ensures forever.
2. Duplication of formulas.

The first kind of redundancy is related to the logical equivalence of the formulas $\Box\delta_1 \wedge \circ((\varphi \wedge (\neg\Box\delta_1 \vee \neg\delta_2))\,\mathcal{U}\,\psi)$ and $\Box\delta_1 \wedge \circ((\varphi \wedge \neg\delta_2)\,\mathcal{U}\,\psi)$. By means of this improvement, formulas of the form $\circ^i\Box\varphi$ and syntactical variants (which are called persistent formulas in the forthcoming Definition 3.4.24 and Proposition 3.4.23) are left out of the context. The second kind of redundancy corresponds to the equivalence of $\varphi \wedge \varphi$ and $\varphi$.

At the end of this subsection, we analyze the gain of the new rule with respect to the older one.

In order to deal with the first kind of redundancy, we introduce the following notion of persistence.

**Definition 3.4.22.** *A formula $\varphi$ is called* persistent *iff for all $\mathcal{M}$ and all $s_j \in S_{\mathcal{M}}$, if $\langle\mathcal{M}, s_j\rangle \models \varphi$ then $\langle\mathcal{M}, s_k\rangle \models \varphi$ for all $k > j$.*

When decomposing formulas in a systematic derivation process some syntactical patterns may be used to detect persistent formulas. That is the case of the formulas of the form $\Box\varphi$ and $\circ\Box\varphi$. By taking also into account that

$$\Box\varphi \equiv \neg\Diamond\varphi \equiv \neg(\mathbf{T}\,\mathcal{U}\,\varphi) \equiv \neg(\neg\mathbf{F}\,\mathcal{U}\,\varphi) \equiv \mathbf{F}\,\mathcal{R}\,\varphi \equiv \neg\mathbf{T}\,\mathcal{R}\,\varphi$$

it is easy to prove the following result which constitutes a syntactical characterization of a subset of persistent formulas.

**Proposition 3.4.23.** *Every formula that matches one of the following patterns:*

$$\circ^i\Box\varphi, \circ^i\neg\Diamond\varphi, \neg\circ^i\Diamond\varphi, \circ^i\neg(\mathbf{T}\,\mathcal{U}\,\varphi), \neg\circ^i(\mathbf{T}\,\mathcal{U}\,\varphi),$$
$$\circ^i\neg(\neg\mathbf{F}\,\mathcal{U}\,\varphi), \neg\circ^i(\neg\mathbf{F}\,\mathcal{U}\,\varphi), \circ^i(\mathbf{F}\,\mathcal{R}\,\varphi), \mathbf{T}, \neg\mathbf{F}$$

*is* persistent. ■

Note that we have characterized a proper subset of the set of all the persistent formulas. For example, $\neg((\neg(\varphi \wedge \neg\varphi))\,\mathcal{U}\,\psi)$ is a persistent formula which does not match any of the above syntactic patterns.

**Definition 3.4.24.** *For any set of formulas $\Phi$, we write* $\mathsf{persist\_ch}(\Phi)$ *to denote the set of all $\gamma \in \Phi$ such that $\gamma$ fits one of the forms considered in Proposition 3.4.23.*

On one hand, in order to avoid the inclusion of persistent formulas in the negation of the context, we define the following operator:

$$\widetilde{\Delta} = \neg(\Delta \setminus \mathsf{persist\_ch}(\Delta))$$

Therefore, to get rid of the above first kind of redundancy, the rule $(\mathcal{U})_3$ applies this new operator $\widetilde{\_}$ instead of the previous operator $\neg(\_)$ to the context.

On the other hand, we define an operator $\Cap$ in order to prevent duplication of formulas. First, we need to extract all the negative conjuncts of a formula. The set $\mathsf{cnjts}(\varphi)$ consists of all the conjuncts of $\varphi$ and is recursively defined as follows:

$$\mathsf{cnjts}(\varphi) = \begin{cases} \mathsf{cnjts}(\varphi_1) \cup \mathsf{cnjts}(\varphi_2) & \text{if } \varphi \text{ is } \varphi_1 \wedge \varphi_2 \\ \{\varphi\} & \text{otherwise} \end{cases}$$

| Rule | $\beta$ | $B_1(\beta)$ | $B_2(\beta, \Delta)$ |
|---|---|---|---|
| $(\mathcal{U})_3$ | $\varphi \mathcal{U} \psi$ | $\{\psi\}$ | $\{\varphi, \neg\psi, \circ((\varphi \Cap \widetilde{\Delta}) \mathcal{U} \psi)\}$ |
| $(\diamond)_3$ | $\diamond \varphi$ | $\{\varphi\}$ | $\{\neg\varphi, \circ(\widetilde{\Delta} \mathcal{U} \varphi)\}$ |

*Figure 3.16:* The Rules $(\mathcal{U})_3$ and $(\diamond)_3$

Then, the set of all negative conjuncts of $\varphi$ is

$$\mathsf{negcnjts}(\varphi) = \{\psi \mid \psi \in \mathsf{cnjts}(\varphi) \text{ and } \psi \text{ is } \mathbf{F} \text{ or a formula of the form } \neg\gamma\}$$

Consequently, the operator $\Cap$ is defined as follows:

$$\varphi \Cap \widetilde{\Delta} = \begin{cases} \mathbf{F} & \text{if } (\Delta \setminus \mathsf{persist\_ch}(\Delta)) = \emptyset \text{ or } \mathbf{F} \in \mathsf{negcnjts}(\varphi) \\ \mathbf{F} & \text{if } \Delta \in \{\mathsf{cnjts}(\psi) \mid \neg\psi \in \mathsf{negcnjts}(\varphi)\} \\ \mathbf{F} & \text{if } (\Delta \setminus \mathsf{persist\_ch}(\Delta)) \in \{\mathsf{cnjts}(\psi) \mid \neg\psi \in \mathsf{negcnjts}(\varphi)\} \\ \varphi \wedge \widetilde{\Delta} & \text{otherwise} \end{cases}$$

Now we give some details to clarify the four cases in the definition of $\varphi \Cap \widetilde{\Delta}$. First of all, let us consider the set

$$\Sigma_1 = \{\chi_1, \ldots, \chi_n, \gamma_1, \ldots, \gamma_m, \varphi, \neg\psi, \circ((\varphi \wedge \neg(\chi_1 \wedge \ldots \wedge \chi_n \wedge \gamma_1 \wedge \ldots \wedge \gamma_m)) \mathcal{U} \psi)\}$$

where $\mathsf{persist\_ch}(\Sigma_1) = \{\chi_1, \ldots, \chi_n\}$. The set $\Sigma_1$ is equivalent to the set

$$\Sigma_2 = \{\chi_1, \ldots, \chi_n, \gamma_1, \ldots, \gamma_m, \varphi, \neg\psi, \circ((\varphi \wedge \neg(\gamma_1 \wedge \ldots \wedge \gamma_m)) \mathcal{U} \psi)\}$$

Consequently, in the definition of $\varphi \Cap \widetilde{\Delta}$, we can exclude the persistent formulas in $\Delta$ that belong to $\mathsf{persist\_ch}(\Delta)$. In the first case, on one hand, if $\Delta \setminus \mathsf{persist\_ch}(\Delta) = \emptyset$ then, since the negation of the empty set is $\mathbf{F}$, we consider the equivalence $\varphi \wedge \mathbf{F} \equiv \mathbf{F}$. On the other hand, if $\mathbf{F} \in \mathsf{negcnjts}(\varphi)$ then we consider the equivalence $\mathbf{F} \wedge \varphi \equiv \mathbf{F}$. In the second case, if $\Delta \in \{\mathsf{cnjts}(\psi) \mid \neg\psi \in \mathsf{negcnjts}(\varphi)\}$ then $\varphi$ is of the form $\varphi_1 \wedge \ldots \wedge \varphi_k$ with $k \geq 1$ and $\varphi_j = \neg\Delta$ for some $j \in \{1, \ldots, k\}$. Therefore, we could consider the equivalence $\varphi \wedge \neg\Delta \equiv \varphi$ and state that in the second case $\varphi \Cap \widetilde{\Delta}$ is $\varphi$. However, $\{\chi, (\gamma \wedge \neg\chi) \mathcal{U} \lambda\}$ is equivalent to $\{\chi, \mathbf{F} \mathcal{U} \lambda\}$ for any formulas $\chi$, $\gamma$ and $\lambda$ and, consequently, we choose $\varphi \Cap \widetilde{\Delta}$ to be $\mathbf{F}$. The third case is like the second one, but without considering the persistent formulas. The fourth case is the general case where the only simplification consists in leaving out the persistent formulas.

By taking into account the above explanation, it is easy to see that the following two sets of formulas are logically equivalent:

$$\Delta \cup \{\circ((\varphi \Cap \widetilde{\Delta}) \mathcal{U} \psi)\} \text{ and } \Delta \cup \{\circ((\varphi \wedge \neg\Delta) \mathcal{U} \psi)\}$$

The rule $(\mathcal{U})_3$ in Figure 3.16 refines the rule $(\mathcal{U})_2$ in Figure 3.1 since the second premise $\circ((\varphi \wedge \neg\Delta) \mathcal{U} \psi)$ of the rule $(\mathcal{U})_2$ is substituted by $\circ((\varphi \Cap \widetilde{\Delta}) \mathcal{U} \psi)$ in the rule $(\mathcal{U})_3$. It is easy to derive, from the new rule $(\mathcal{U})_3$, the corresponding rule $(\diamond)_3$ for the defined connective $\diamond$.

Now, let us give two examples that make use of these two new rules $(\diamond)_3$ and $(\mathcal{U})_3$ showed in Figure 3.16. In these examples, the tableaux are built by using the systematic tableau algorithm in Figure 3.9 and the rules $(\diamond)_3$ and $(\mathcal{U})_3$ instead of the rules $(\diamond)_2$ and $(\mathcal{U})_2$. In order to show each tableau we follow the same notation as in Subsection 3.4.3 (and Subsection 3.3.5). As in the applications of the rules $(\diamond)_2$ and $(\mathcal{U})_2$, in the applications of the rules $(\diamond)_3$ and $(\mathcal{U})_3$ we only underline the eventuality to which the rule is applied.

**Example 3.4.25.** *In Figures 3.17 and 3.18 we depict a systematic tableau for $\{p, \Box\diamond p\}$ obtained by using the rules $(\diamond)_3$ and $(\mathcal{U})_3$. As expected from the satisfiability of the root set, the tableau is open. Concretely, there are two cyclic (expanded) branches with a common repeated node. Both rules $(\diamond)_3$ and $(\mathcal{U})_3$ are used twice. In the first application (from the top) of the rule $(\diamond)_3$, the persistent formula $\circ\Box\diamond p$ is left out of the negation of the context. Consequently, only the negation of p is considered in the formula $\circ((\neg p)\,\mathcal{U}\,p)$, which belongs to the label of the child on the right. In the second application of the rule $(\diamond)_3$, again the persistent formula $\circ\Box\diamond p$ is left out of the negation of the context. Since there are no more formulas, the set of non-persistent formulas is empty and the formula $\circ(\mathbf{F}\,\mathcal{U}\,p)$ is in the label of the child on the right. In both applications of the rule $(\mathcal{U})_3$, the selected eventuality is $\mathbf{F}\,\mathcal{U}\,p$. In both cases, the corresponding context contains at least one formula that is not persistent but, by definition of the operator $\Cap$, the formula $\circ(\mathbf{F}\,\mathcal{U}\,p)$ is produced in both cases because of the formula $\mathbf{F}$ in $\mathbf{F}\,\mathcal{U}\,p$.*

*The left-most open brach, $b_1$, is formed by six nodes $n_0, \ldots, n_5$ where $L(n_2) = L(n_5)$, and yields two stages, $x_0 = n_0, n_1, n_2$ and $x_1 = n_3, n_4, n_5$. Consequently $\mathsf{path}(b_1) = x_0 \cdot \langle x_1 \rangle^\omega$. The right-most open brach, $b_2$, is formed by ten nodes $n'_0, \ldots, n'_9$ where $L(n'_2) = L(n'_9)$ and gives rise to three stages, $y_0 = n'_0, n'_1, n'_2$, $y_1 = n'_3, n'_4, n'_5$ and $y_2 = n'_6, \ldots, n'_9$. Therefore $\mathsf{path}(b_2) = y_0 \cdot \langle y_1, y_2 \rangle^\omega$. In the models described by $b_1$, p is true in all the states. In the models described by $b_2$, p is true in the states $s_0, s_2, s_4, \ldots$ whereas $\neg p$ is true in the remaining states $(s_1, s_3, s_5, \ldots)$.*

**Example 3.4.26.** *By means of Figures 3.19 and 3.20, we show a systematic closed tableau for the unsatisfiable set $\{p, \Box(\neg p \vee \circ p), \diamond\neg p\}$. In this tableau we use the rules $(\diamond)_3$ and $(\mathcal{U})_3$. In the nodes where $(\diamond)_3$ and $(\mathcal{U})_3$ are applied the context is $\{p, \circ p, \circ\Box(\neg p \vee \circ p)\}$ and the set $\mathsf{persist\_ch}(\{p, \circ p, \circ\Box(\neg p \vee \circ p)\}) = \{p, \circ p\}$. Therefore, when the rule $(\diamond)_3$ is applied, the considered set of formulas is $\Delta_0 = \{p, \circ p\}$ and the formula $\circ((\neg\Delta_0)\,\mathcal{U}\,\neg p)$ is obtained. In the same way, when the rule $(\mathcal{U})_3$ is applied, the considered set of formulas is $\Delta_1 = \{p, \circ p\}$. However, since $\Delta_0 = \Delta_1$, the application of the rule $(\mathcal{U})_3$ yields the formula $\circ(\mathbf{F}\,\mathcal{U}\,\neg p)$ instead of the formula $\circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)$ generated by the rule $(\mathcal{U})_2$ in Figure 3.12. As can be appreciated in the definitions of $\Delta_0$ and $\Delta_1$, the persistent formula $\circ\Box(\neg p \vee \circ p)$ is left out of the context in the applications of the rules $(\diamond)_3$ and $(\mathcal{U})_3$. Additionally, the application of the rule $(\mathcal{U})_3$ avoids the repetition of $\Delta_0$ and obtains a simplified formula by using $\mathbf{F}$. As a consequence of these improvements the tableau has one branch less than the tableau constructed in Example 3.4.14 and the longest branch contains one node less than the longest branch in Example 3.4.14 (Figure 3.12).*

Finally, we formally analyze the gain of using the rule $(\mathcal{U})_3$ instead of the rule $(\mathcal{U})_2$. This analysis yields a small difference between both worst cases, although the improvement is very useful for practical implementation.

We reformulate the notion of closure for the system $(\mathrm{TTM} \setminus \{(\mathcal{U})_2\}) \cup \{(\mathcal{U})_3\}$. To this end, we also need to redefine some other previously defined sets of formulas. However, other auxiliary sets, e.g. preclosure, remain defined as before. In order to stress which sets are redefined,

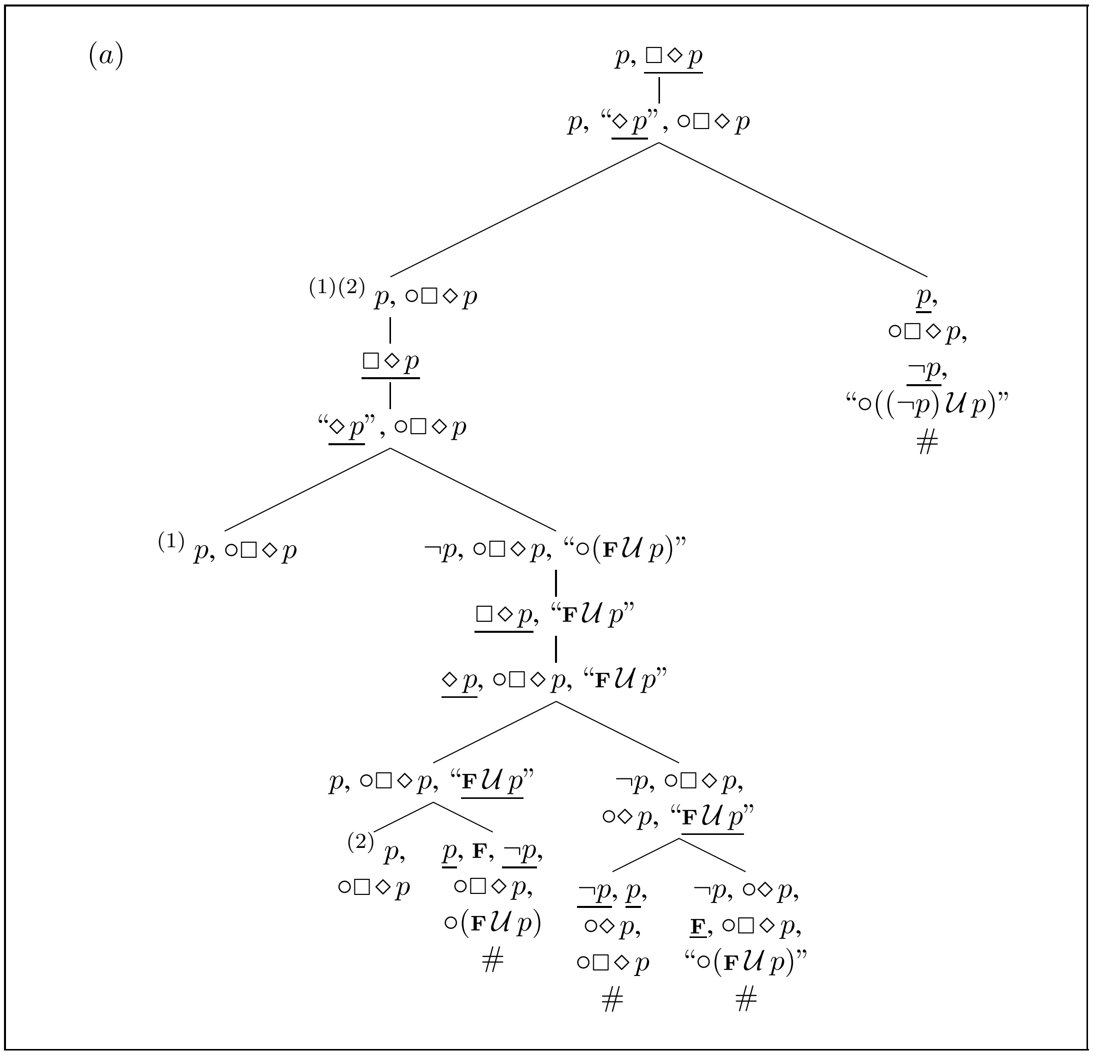


*Figure 3.17:* Systematic expanded tableau for $\{p, \Box\Diamond p\}$ by using $(\Diamond)_3$ and $(\mathcal{U})_3$ (Part 1 of 2)

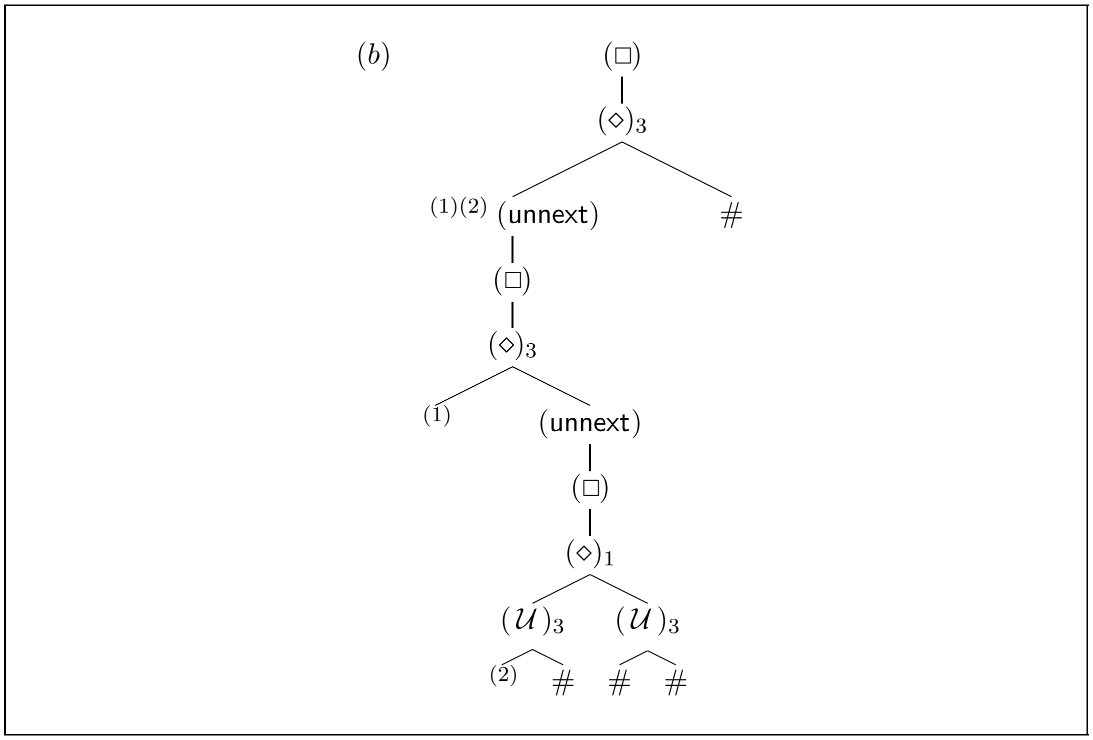


*Figure 3.18:* Systematic expanded tableau for $\{p, \Box\Diamond p\}$ by using $(\Diamond)_3$ and $(\mathcal{U})_3$ (Part 2 of 2)

we use the prefix new_. The new definitions for the sets of negated contexts and conjunctions are:

$$\mathsf{new\_negctx}(\Phi) = \{\neg\Delta \mid \Delta \subseteq (\mathsf{preclo}(\Phi) \setminus \mathsf{persist\_ch}(\mathsf{preclo}(\Phi)))\}.$$

$$\mathsf{new\_conj}(\Phi) = \{\bigwedge_{\delta\in\Gamma} \delta \mid \Gamma \subseteq \mathsf{new\_negctx}(\Phi) \text{ and } \Gamma \text{ is adequate}\}.$$

where we say that $\Gamma \subseteq \mathsf{new\_negctx}(\Phi)$ is *adequate* iff

$$\mathsf{cnjts}(\delta) \neq \mathsf{cnjts}(\delta') \text{ for every pair } (\neg\delta, \neg\delta') \in \Gamma \times \Gamma \text{ such that } \delta \neq \delta'.$$

Now, the closure of $\Phi$ can be redefined as follows:

$\mathsf{new\_clo}(\Phi) = \mathsf{preclo}(\Phi) \cup \mathsf{new\_conj}(\Phi) \cup \Omega$
where
$\Omega = \{(\varphi \wedge \gamma)\,\mathcal{U}\,\psi, \circ((\varphi \wedge \gamma)\,\mathcal{U}\,\psi), \mathbf{F}\,\mathcal{U}\,\psi, \circ(\mathbf{F}\,\mathcal{U}\,\psi) \mid \varphi\,\mathcal{U}\,\psi \in \mathsf{sf}(\Phi) \text{ and } \gamma \in \mathsf{new\_conj}(\Phi)\}$

Hence, the cardinality of this closure is a bit smaller than stated in Proposition 3.4.10. Actually, if $|\mathsf{preclo}(\Phi)| = n$ then $|\mathsf{new\_negctx}(\Phi)| \in O(2^n)$. Therefore

$$|\mathsf{new\_conj}(\Phi)|, |\mathsf{new\_clo}(\Phi)| \in O(2^{2^n}).$$

Recall that $|\mathsf{clo}(\Phi)| \in O(2^{O(2^n)})$.

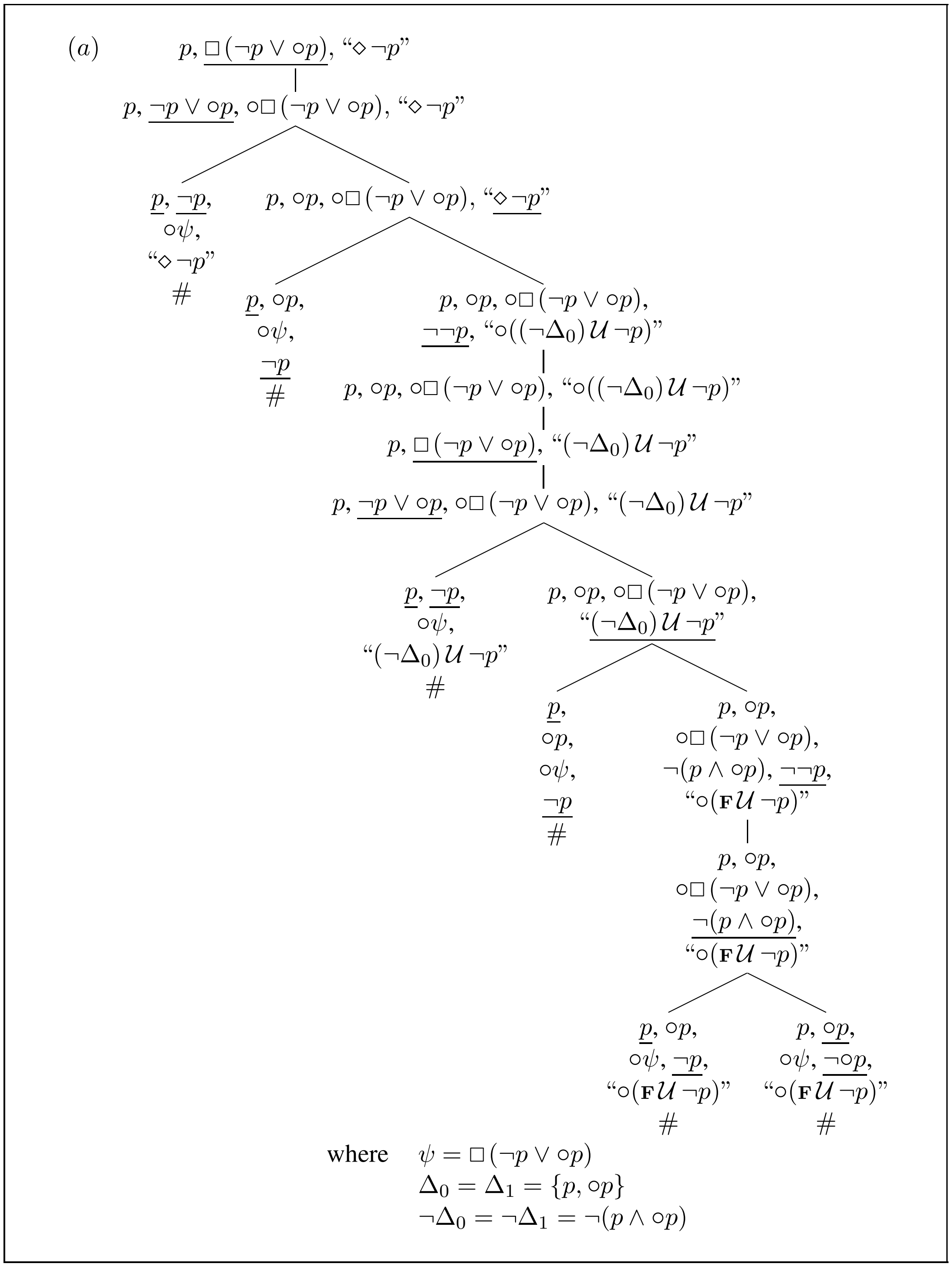

(a)
p, □(¬p ∨ ○p), "◇¬p"
p, ¬p ∨ ○p, ○□(¬p ∨ ○p), "◇¬p"
p, ¬p, ○ψ, "◇¬p" #
p, ○p, ○□(¬p ∨ ○p), "◇¬p"
p, ○p, ○ψ, ¬p #
p, ○p, ○□(¬p ∨ ○p), ¬¬p, "○((¬Δ0) U ¬p)"
p, ○p, ○□(¬p ∨ ○p), "○((¬Δ0) U ¬p)"
p, □(¬p ∨ ○p), "(¬Δ0) U ¬p"
p, ¬p ∨ ○p, ○□(¬p ∨ ○p), "(¬Δ0) U ¬p"
p, ¬p, ○ψ, "(¬Δ0) U ¬p" #
p, ○p, ○□(¬p ∨ ○p), "(¬Δ0) U ¬p"
p, ○p, ○ψ, ¬p #
p, ○p, ○□(¬p ∨ ○p), ¬(p ∧ ○p), ¬¬p, "○(F U ¬p)"
p, ○p, ○□(¬p ∨ ○p), ¬(p ∧ ○p), "○(F U ¬p)"
p, ○p, ○ψ, ¬p, "○(F U ¬p)" #
p, ○p, ○ψ, ¬○p, "○(F U ¬p)" #
where ψ = □(¬p ∨ ○p)
Δ0 = Δ1 = {p, ○p}
¬Δ0 = ¬Δ1 = ¬(p ∧ ○p)

*Figure 3.19:* Systematic closed tableau for $\{p, \Box(\neg p \vee \circ p), \Diamond\neg p\}$ by using $(\Diamond)_3$ and $(\mathcal{U})_3$ (Part 1 of 2)

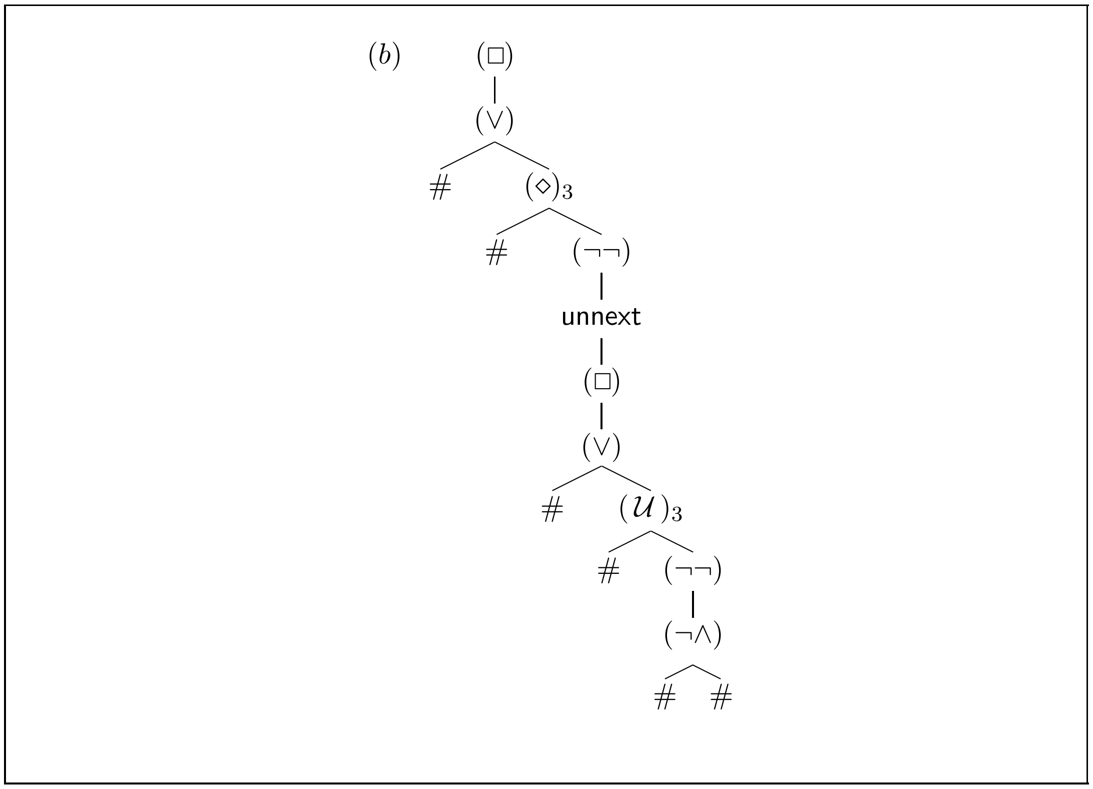


*Figure 3.20:* Systematic closed tableau for $\{p, \square(\neg p \vee \circ p), \diamond \neg p\}$ by using $(\diamond)_3$ and $(\mathcal{U})_3$ (Part 2 of 2)

## 3.5 The Sequent Calculus TTC

In this section we introduce the sequent calculus TTC that directly corresponds to the previously introduced tableau system TTM. It is a reformulation of TTM as a one-sided sequent calculus that serves as a bridge from TTM to the two-sided sequent calculus GTC that we introduce in the next section (Section 3.6).

The sequent calculus TTC follows the left-handed one-sided approach (also known as Tait-style, [123]), where sequents are formed by a set of formulas. We write $\Delta \vdash$ to represent a sequent whose set of formulas is $\Delta$ and whose intended meaning is $\bigwedge \Delta \to \mathrm{F}$, i.e. $\neg(\bigwedge \Delta)$.

The rules of TTC (see Figure 3.21) are obtained essentially from the TTM-rules writing them upside down with the difference that in TTC we have left-handed sequents and in TTM we have simply sets of formulas. The only exception is the rule $(\circ)$ that corresponds to the application of the operator unnext in TTM. This direct relation between both systems makes possible to obtain a TTC-proof from any closed TTM-tableau in a straightforward manner.

The strong similarity between tableau refutations and left-handed sequent proofs that are cut-free, contraction-free and weakening-free is evident. As a consequence, TTC is cut-free, invariant-free, weakening-free and contraction-free.

We have split the primitive rules of TTC into three packages. Two of them consist of rules for classical and temporal connectives, respectively. These rules follow the traditional style of introduction of the connective and its negation in the sequent. In addition, we need two structural rules which form the third package.

Rules for the Classical Connectives

$$(\neg\neg)\ \frac{\Delta, \varphi \vdash}{\Delta, \neg\neg\varphi \vdash} \qquad (\wedge)\ \frac{\Delta, \varphi, \psi \vdash}{\Delta, \varphi \wedge \psi \vdash} \qquad (\neg\wedge)\ \frac{\Delta, \neg\varphi \vdash \qquad \Delta, \neg\psi \vdash}{\Delta, \neg(\varphi \wedge \psi) \vdash}$$

Rules for the Temporal Connectives

$$(\circ)\ \frac{\mathsf{unnext}(\Delta) \vdash}{\Delta \vdash} \qquad (\mathcal{U})_1\ \frac{\Delta, \psi \vdash \qquad \Delta, \varphi, \neg\psi, \circ(\varphi \mathcal{U} \psi) \vdash}{\Delta, \varphi \mathcal{U} \psi \vdash}$$

$$(\neg\circ)\ \frac{\Delta, \circ\neg\varphi \vdash}{\Delta, \neg\circ\varphi \vdash} \qquad (\mathcal{U})_2\ \frac{\Delta, \psi \vdash \qquad \Delta, \varphi, \neg\psi, \circ((\varphi \wedge \neg\Delta) \mathcal{U} \psi) \vdash}{\Delta, \varphi \mathcal{U} \psi \vdash}$$

$$(\neg\mathcal{U})\ \frac{\Delta, \neg\varphi, \neg\psi \vdash \qquad \Delta, \varphi, \neg\psi, \neg\circ(\varphi \mathcal{U} \psi) \vdash}{\Delta, \neg(\varphi \mathcal{U} \psi) \vdash}$$

Structural Rules

$$(Cd_1)\ \frac{}{\Delta, \varphi, \neg\varphi \vdash} \qquad (Cd_2)\ \frac{}{\Delta, \mathbf{F} \vdash}$$

*Figure 3.21:* Primitive TTC-Rules

As TTC is sound and complete (Theorems 3.5.1 and 3.5.3), given a set of formulas $\Delta$, it holds that $\Delta$ is unsatisfiable if and only if there is a TTC-proof for $\Delta \vdash$.

A TTC-derivation is a possibly infinite tree labelled with sequents and built according to the inference rules in TTC. A TTC-proof is a finite derivation where the sequent to be proved labels its root and the leaves are labelled with axioms (which are rules without premises).

A set of formulas $\Gamma$ is TTC-*consistent* if and only if there is no any TTC-proof for the sequent $\Gamma \vdash$.

The soundness of TTC means that every TTC-provable sequent, namely $\Gamma \vdash$, is correct regarding to satisfiability. In particular, every satisfiable set of formulas $\Gamma$ is TTC-consistent.

In the TTC sequent calculus all the non-structural rules are *invertible* with the exception of the rule $(\circ)$. A rule is invertible when it holds that if the conclusion is provable, so are the premises.

**Theorem 3.5.1. (Soundness)** *For any set of formulas* $\Gamma$*, if* $\Gamma$ *is not* TTC*-consistent, i.e., if there exists a* TTC*-proof, then* $\Gamma$ *is unsatisfiable.*

*Proof.* By induction on the length of the TTC-proof, it suffices to prove that every primitive rule of TTC (see Figure 3.21) is correct in the sense that if the set of formulas of each premise is unsatisfiable then the set of formulas of the conclusion is unsatisfiable. The only difficult case is the case of the rule $(\mathcal{U})_2$. The justification for that case is already given in Theorem 3.4.2. ∎

Next, we prove that TTC is a complete calculus by relating its completeness to the completeness of TTM.

**Proposition 3.5.2.** *For any set of formulas $\Phi$, if $\mathcal{T}_\Phi$ is a closed expanded tableau for $\Phi$ then there exists a* TTC*-proof for the sequent $\Phi \vdash$.*

*Proof.* Since each TTM-rule has its corresponding TTC-rule, the TTC-proof is directly obtained from the closed TTM-tableau for $\Phi$. ∎

**Theorem 3.5.3. (Completeness)** *For any set of formulas $\Phi$, if $\Phi$ is unsatisfiable, then there exists a* TTC*-proof for $\Phi$.*

*Proof.* If $\Phi$ is unsatisfiable then there exists a closed TTM-tableau for $\Phi$. Hence, by Proposition 3.5.2 there exists a TTC-proof for $\Phi$. ∎

As in the case of TTM, the exhaustive application of the rules in the calculus TTC, without any additional restriction or strategy, does not yield a decision procedure for PLTL. The reason is that TTC, by itself, does not satisfy the weak analytic superformula property (WASP) (see Subsection 3.4.2). Remember that the systematic tableau algorithm of Subsection 3.4.2 incorporates a strategy for the application of $(\mathcal{U})_2$ which contributes to the satisfaction of the WASP.

When building a TTC-derivation we can use primitive rules, derived rules and also admissible rules. The admissible rules are new sound rules that cannot be derived from the primitive rules of TTC, but do not add deductive power to the system. That is, a set $\Phi$ is consistent with respect to TTC if and only if $\Phi$ is consistent with respect to TTC plus the admissible rules. In other words, for every TTC-proof that includes the use of some admissible rules there exists another TTC-proof that does not use any admissible rule.

The derived rules can be used as a shortcut for several lines of proofs that are built by using only primitive and admissible rules.

Among the admissible rules the most outstanding ones are the following classical structural rules of Weakening and Cut:

$$(Wk)\,\frac{\Delta \vdash}{\Delta, \Delta' \vdash} \qquad\qquad (Cut)\,\frac{\Delta, \varphi \vdash \qquad \Delta, \neg\varphi \vdash}{\Delta \vdash}$$

The sequent calculus TTC is cut-free since we have already proved its soundness and completeness and the cut rule is omitted in TTC. Since TTC is complete without the cut rule, the cut rule is admissible in TTC. However, the classical syntactical techniques for cut elimination cannot be applied here because of the context used in the rule $(\mathcal{U})_2$. Hence, we have been unable to give a syntactic proof of cut elimination. However, we are aware of the work of K. Brünnler, who introduced the notion of *deep sequent* and gave a cut-elimination procedure for modal logic ([19]). It remains open to see whether the same technique applied to our calculi (extended with the cut rule) could yield a syntactical cut-elimination procedure for PLTL.

$$(\vee)\,\frac{\Delta, \varphi \vdash \qquad \Delta, \psi \vdash}{\Delta, \varphi \vee \psi \vdash} \qquad (\mathcal{R})\,\frac{\Delta, \varphi, \psi \vdash \qquad \Delta, \neg\varphi, \psi, \circ(\varphi \mathcal{R} \psi) \vdash}{\Delta, \varphi \mathcal{R} \psi \vdash}$$

$$(\neg\vee)\,\frac{\Delta, \neg\varphi, \neg\psi \vdash}{\Delta, \neg(\varphi \vee \psi) \vdash} \qquad (\neg\mathcal{R})_1\,\frac{\Delta, \neg\psi \vdash \qquad \Delta, \neg\varphi, \psi, \circ(\neg\varphi \mathcal{U} \neg\psi) \vdash}{\Delta, \neg(\varphi \mathcal{R} \psi) \vdash}$$

$$(\neg\mathcal{R})_2\,\frac{\Delta, \neg\psi \vdash \qquad \Delta, \neg\varphi, \psi, \circ((\neg\varphi \wedge \neg\Delta) \mathcal{U} \neg\psi) \vdash}{\Delta, \neg(\varphi \mathcal{R} \psi) \vdash}$$

$$(\Box)\,\frac{\Delta, \varphi, \circ\Box\varphi \vdash}{\Delta, \Box\varphi \vdash} \qquad (\Diamond)_1\,\frac{\Delta, \varphi \vdash \qquad \Delta, \neg\varphi, \circ(\mathbf{T} \mathcal{U} \varphi) \vdash}{\Delta, \Diamond\varphi \vdash}$$

$$(\neg\Diamond)\,\frac{\Delta, \neg\varphi, \neg\circ\Diamond\varphi \vdash}{\Delta, \neg\Diamond\varphi \vdash} \qquad (\Diamond)_2\,\frac{\Delta, \varphi \vdash \qquad \Delta, \neg\varphi, \circ((\neg\Delta) \mathcal{U} \varphi) \vdash}{\Delta, \Diamond\varphi \vdash}$$

$$(\neg\Box)_1\,\frac{\Delta, \neg\varphi \vdash \qquad \Delta, \varphi, \circ(\mathbf{T} \mathcal{U} \neg\varphi) \vdash}{\Delta, \neg\Box\varphi \vdash}$$

$$(\neg\Box)_2\,\frac{\Delta, \neg\varphi \vdash \qquad \Delta, \varphi, \circ(\neg\Delta \mathcal{U} \neg\varphi) \vdash}{\Delta, \neg\Box\varphi \vdash}$$

*Figure 3.22:* Some Derived Rules for TTC

The weakening rule $(Wk)$ is non-invertible so it must be used carefully. The rules $(\mathbf{T})$ and $(\neg\mathbf{F})$, that appear below, are particular cases of the rule $(Wk)$ but they are invertible. So they can be used to eliminate the formulas $\mathbf{T}$ and $\neg\mathbf{F}$ knowing that the equivalence with respect to the TTC-consistency is preserved:

$$(\mathbf{T})\,\frac{\Delta \vdash}{\Delta, \mathbf{T} \vdash} \qquad (\neg\mathbf{F})\,\frac{\Delta \vdash}{\Delta, \neg\mathbf{F} \vdash}$$

Since TTC is also contraction-free, admissible rules could be obtained by associating to every non-structural rule $(R)$ the rule $(RC)$ that produces an (implicit) contraction in $(R)$. For example, the rule below $(\wedge C)$ is the admissible rule that corresponds to the primitive rule $(\wedge)$.

$$(\wedge C)\,\frac{\Delta, \varphi \wedge \psi, \varphi, \psi \vdash}{\Delta, \varphi \wedge \psi \vdash}$$

Regarding derived rules, first we use the usual abbreviations of defined connectives in order to derive the rules in Figure 3.22. It is easy to check that $(\vee)$ is derived from $(\neg\wedge)$ and $(\neg\neg)$; $(\neg\vee)$ from $(\neg\neg)$ and $(\wedge)$; $(\mathcal{R})$ from $(\neg\mathcal{U})$ and $(\neg\neg)$; for $i \in \{1, 2\}$: $(\neg\mathcal{R})_i$ is derived from

$$
\dfrac{\dfrac{}{q,\underline{\mathbf{F}}\vdash}\,(Cd_2)\qquad
\dfrac{\dfrac{\dfrac{}{\underline{\mathbf{F}}\vdash}\,(Cd_2)\qquad
\dfrac{\dfrac{\dfrac{}{\underline{\mathbf{F}}\vdash}\,(Cd_2)\qquad \dfrac{}{\underline{\mathbf{F}},\neg\mathbf{F},\circ(\mathbf{F}\mathcal{U}\,\mathbf{F})\vdash}\,(Cd_2)}{\underline{\mathbf{F}\mathcal{U}\,\mathbf{F}}\vdash}\,(\mathcal{U})_3}{p\wedge\neg q,\neg\mathbf{F},\circ(\mathbf{F}\mathcal{U}\,\mathbf{F})\vdash}\,(\circ)}{\underline{(p\wedge\neg q)\mathcal{U}\,\mathbf{F}}\vdash}\,(\mathcal{U})_3}{q,p,\neg\mathbf{F},\circ((p\wedge\neg q)\mathcal{U}\,\mathbf{F})\vdash}\,(\circ)}{q,\underline{p\mathcal{U}\,\mathbf{F}}\vdash}\,(\mathcal{U})_3
$$

*Figure 3.23:* TTC-proof for the set of formulas $\{q, p\mathcal{U}\,\mathbf{F}\}$

$(\neg\neg)$ and $(\mathcal{U})_i$; for $i \in \{1,2\}$: $(\diamond)_i$ is derived from $(\mathcal{U})_i$ and $(\mathrm{T})$; $(\neg\diamond)$ is derived from $(\neg\mathcal{U})$, $(\mathrm{T})$, $(\neg\neg)$ and $(Cd)_2$; $(\Box)$ from $(\neg\diamond)$, $(\neg\neg)$, $(\mathrm{T})$ and $(\neg\circ)$; and for $i \in \{1,2\}$: $(\neg\Box)_i$ from $(\neg\neg)$, $(\diamond)_i$ and $(\mathrm{T})$.

The soundness and invertibility of these derived rules is guaranteed by the fact that they have been obtained using only sound and invertible rules. Note that if the rule $(Wk)$ is used instead of $(\mathrm{T})$ for deriving the previous rules their invertibility could not be directly guaranteed.

It is well known that the connective $\mathcal{U}$ is not expressible in temporal logic with only $\circ$, $\Box$, and $\diamond$ as temporal connectives (cf. [80, 53]). As a consequence, a complete calculus for the sublogic that uses $\diamond$ instead of $\mathcal{U}$ cannot be derived (by abbreviation) from TTC, since the rule $(\diamond)_2$ needs the connective $\mathcal{U}$ for expressing its second premise.

Finally, let us recall the respective refinements $(\diamond)_3$ and $(\mathcal{U})_3$ of the rules $(\diamond)_2$ and $(\mathcal{U})_2$ that allow us to avoid the inclusion of persistent formulas and duplications in the negation of the context (see Subsection 3.4.5):

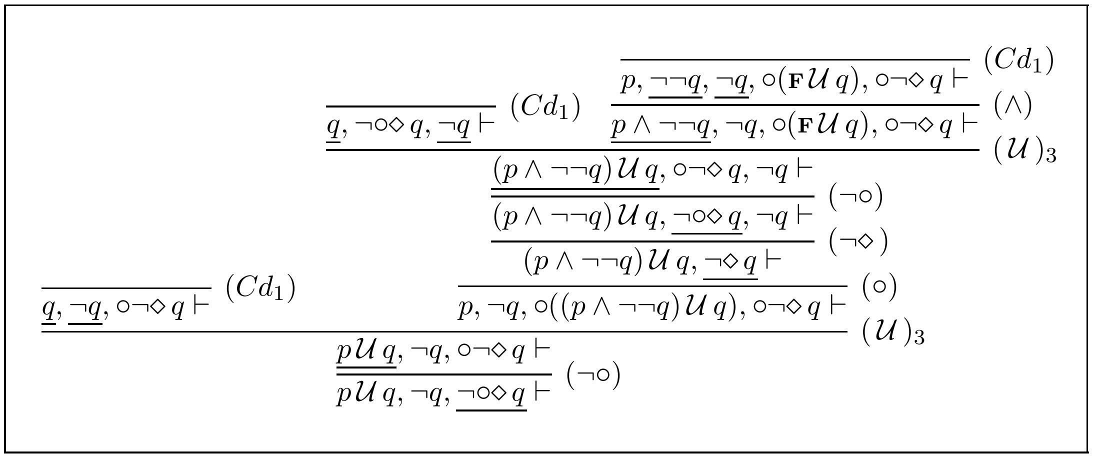
$$
(\diamond)_3\;\dfrac{\begin{array}{l}\Delta,\varphi\vdash\\ \Delta,\neg\varphi,\circ(\widetilde{\Delta}\,\mathcal{U}\,\varphi)\vdash\end{array}}{\Delta,\diamond\varphi\vdash}
\qquad\qquad
(\mathcal{U})_3\;\dfrac{\begin{array}{l}\Delta,\psi\vdash\\ \Delta,\varphi,\neg\psi,\circ((\varphi\Cap\widetilde{\Delta})\,\mathcal{U}\,\psi)\vdash\end{array}}{\Delta,\varphi\,\mathcal{U}\,\psi\vdash}
$$

Now, let us illustrate the TTC-style of reasoning by means of some examples of TTC-proofs. In order to enhance readability, we have underlined, at each step, the principal formula. However, when the rule $(\circ)$ is applied, we do not underline any formula. In the nodes in which we apply the rules $(\mathcal{U})_2$, $(\diamond)_2$, $(\mathcal{U})_3$ or $(\diamond)_3$, we only underline the eventuality to which the rule is applied.

Actually, each derivation can be seen as an inverted closed TTM tableau.

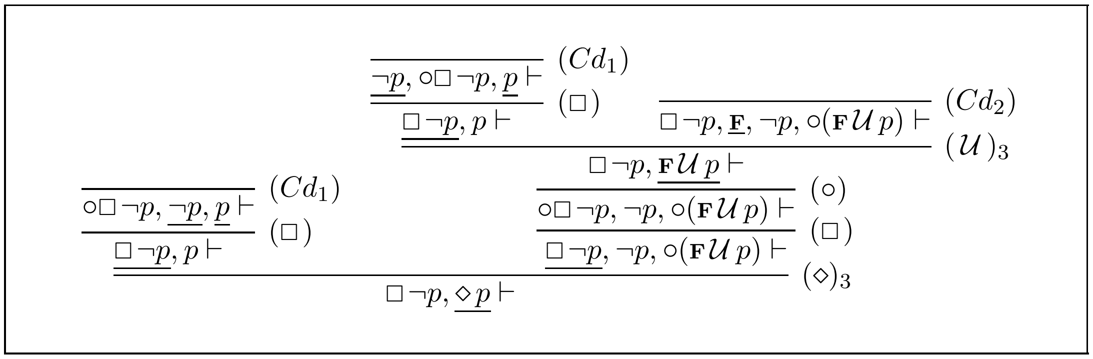
**Example 3.5.4.** *The* TTC*-proof in Figure 3.23 shows that the set of formulas* $\{q, p\mathcal{U}\,\mathbf{F}\}$ *is unsatisfiable.*

*Note that in the first application (from the bottom) of the rule* $(\mathcal{U})_3$ *the obtained premises coincide with the ones that we would obtain by using the rule* $(\mathcal{U})_2$*. By contrast, in the second application of the rule* $(\mathcal{U})_3$*, the right-hand premise is different from the one that we would obtain by using the rule* $(\mathcal{U})_2$*. By using* $(\mathcal{U})_2$ *we would obtain the sequent* $p\wedge\neg q, \neg\mathbf{F}, \circ((p\wedge\neg q\wedge\mathbf{F})\mathcal{U}\,\mathbf{F})\vdash$ *instead of the sequent* $p\wedge\neg q,\neg\mathbf{F},\circ(\mathbf{F}\mathcal{U}\,\mathbf{F})\vdash$*. It is also worth noting that this* TTC*-proof does not exactly follow the strategy formalized by means of the systematic tableau algorithm in Figure 3.9. In particular, in the second application (from the bottom) of the rule* $(\circ)$ *the sequent* $p\wedge\neg q,\neg\mathbf{F},\circ(\mathbf{F}\mathcal{U}\,\mathbf{F})\vdash$ *is not formed only by elementary formulas.*



$p, \neg\neg q, \neg q, \circ(\mathbf{F}\,\mathcal{U}\,q), \circ\neg\diamond q \vdash$ $(Cd_1)$
$q, \neg\circ\diamond q, \neg q \vdash$ $(Cd_1)$
$p \wedge \neg\neg q, \neg q, \circ(\mathbf{F}\,\mathcal{U}\,q), \circ\neg\diamond q \vdash$ $(\wedge)$
$(p \wedge \neg\neg q)\,\mathcal{U}\,q, \circ\neg\diamond q, \neg q \vdash$ $(\mathcal{U})_3$
$(p \wedge \neg\neg q)\,\mathcal{U}\,q, \neg\circ\diamond q, \neg q \vdash$ $(\neg\circ)$
$(p \wedge \neg\neg q)\,\mathcal{U}\,q, \neg\diamond q \vdash$ $(\neg\diamond)$
$q, \neg q, \circ\neg\diamond q \vdash$ $(Cd_1)$
$p, \neg q, \circ((p \wedge \neg\neg q)\,\mathcal{U}\,q), \circ\neg\diamond q \vdash$ $(\circ)$
$p\,\mathcal{U}\,q, \neg q, \circ\neg\diamond q \vdash$ $(\mathcal{U})_3$
$p\,\mathcal{U}\,q, \neg q, \neg\circ\diamond q \vdash$ $(\neg\circ)$

*Figure 3.24:* TTC-proof for the set of formulas $\{p\,\mathcal{U}\,q, \neg\circ\diamond q, \neg q\}$



$\neg p, \circ\Box\neg p, p \vdash$ $(Cd_1)$
$\Box\neg p, p \vdash$ $(\Box)$
$\Box\neg p, \mathbf{F}, \neg p, \circ(\mathbf{F}\,\mathcal{U}\,p) \vdash$ $(Cd_2)$
$\Box\neg p, \mathbf{F}\,\mathcal{U}\,p \vdash$ $(\mathcal{U})_3$
$\circ\Box\neg p, \neg p, p \vdash$ $(Cd_1)$
$\Box\neg p, p \vdash$ $(\Box)$
$\circ\Box\neg p, \neg p, \circ(\mathbf{F}\,\mathcal{U}\,p) \vdash$ $(\circ)$
$\Box\neg p, \neg p, \circ(\mathbf{F}\,\mathcal{U}\,p) \vdash$ $(\Box)$
$\Box\neg p, \diamond p \vdash$ $(\diamond)_3$

*Figure 3.25:* TTC-proof for the set of formulas $\{\Box\neg p, \diamond p\}$

**Example 3.5.5.** *In Figure 3.24 we depict a* TTC*-proof for the unsatisfiable set of formulas* $\{p\,\mathcal{U}\,q, \neg\circ\diamond q, \neg q\}$.

*Note that in the first application (from the bottom) of the rule* $(\mathcal{U})_3$*, we avoid to consider the permanent formula* $\circ\neg\diamond q$ *in the negation of the context. Consequently in the right-hand premise we obtain the sequent* $p, \neg q, \circ((p \wedge \neg\neg q)\,\mathcal{U}\,q), \circ\neg\diamond q \vdash$ *instead of the sequent* $p, \neg q, \circ((p \wedge \neg\neg q \wedge \neg\circ\neg\diamond q)\,\mathcal{U}\,q), \circ\neg\diamond q \vdash$ *that we would obtain by using the rule* $(\mathcal{U})_2$*. In the second application of the rule* $(\mathcal{U})_3$*, we obtain the sequent* $p \wedge \neg\neg q, \neg q, \circ(\mathbf{F}\,\mathcal{U}\,q), \circ\neg\diamond q \vdash$ *as the right-hand premise because we dispense with the persistent formula* $\circ\neg\diamond q$ *and because the negation of* $\neg q$ *(i.e. the negation of the context without persistent formulas) is a conjunct of the left-hand subformula of* $(p \wedge \neg\neg q)\,\mathcal{U}\,q$.

**Example 3.5.6.** *In Figure 3.25 we show a* TTC*-proof for the unsatisfiable set of formulas* $\{\Box\neg p, \diamond p\}$.

*Note that, when the rule* $(\diamond)_3$ *is applied to the sequent* $\Box\neg p, \diamond p \vdash$*, the formula* $\Box\neg p$ *is left out of the negation of the context. Therefore the negation of the context without persistent formulas is* $\mathbf{F}$*. When the rule* $(\mathcal{U})_3$ *is applied to the sequent* $\Box\neg p, \mathbf{F}\,\mathcal{U}\,p \vdash$*, on one hand the formula* $\Box\neg p$ *is left out of the negation of the context. On the other hand, the negation of the* *context without persistent formulas is* $\mathbf{F}$. *However,* $\mathbf{F}$ *is not repeated in the new formula that contains the connective* $\mathcal{U}$, *i.e., the new formula is* $\circ(\mathbf{F}\,\mathcal{U}\,p)$ *instead of* $\circ((\mathbf{F}\wedge\mathbf{F})\,\mathcal{U}\,p)$, *which we would obtain if the rule* $(\mathcal{U})_2$ *were used. Note also that this* TTC*-proof does not exactly follow the strategy formalized by means of the systematic tableau algorithm in Figure 3.9, because the rules* $(\mathcal{U})_3$ *and* $(\diamond)_3$ *are applied to sets of formulas that are not elementary.*

**Example 3.5.7.** *The* TTC*-proof in Figure 3.26 shows that the set of formulas* $\{p, \Box(\neg p \vee \circ p), \diamond\neg p\}$ *is unsatisfiable. Actually, this proof can be obtained by inverting the closed tableau built in Example 3.4.14 (Figures 3.12 and 3.13). Note that every set* $\Sigma_i$, *with* $i \in \{0,\dots,5\}$, *is inconsistent and the rule* $(Cd_1)$ *is used for each of them. In particular, sets* $\Sigma_0,\dots,\Sigma_3$ *contain* $p$ *and* $\neg p$, $\Sigma_4$ *contains* $\circ p$ *and* $\neg\circ p$ *and* $\Sigma_5$ *contains* $\circ\psi$ *and* $\neg\circ\psi$ *where* $\psi = \Box(\neg p \vee \circ p)$.

## 3.6 The Sequent Calculus GTC

In this section we present the sequent calculus GTC (see Figure 3.27) that is two-sided and one-conclusioned (or asymmetric). We prove the soundness of GTC and, then, we discuss about admissible and derived rules. Afterwards, we prove the completeness of GTC with the help of some previously derived rules. Finally, we give four examples of GTC-proofs.

The calculus GTC (see Figure 3.27) is straightforwardly obtained from the previous calculus TTC. Actually, almost each primitive rule of TTC has a counterpart in GTC that results from adding a conclusion $\chi$ to each sequent in the rule. The only exception are the rules where the context is combined with the principal formula to produce the sequents in the numerator, where $\chi$ (or better $\neg\chi$) behaves as part of the context. Moreover, admissible or derived rules in GTC are the same kind of counterparts of TTC rules as the primitive ones.

The soundness of GTC means that every GTC-provable sequent, namely $\Gamma \vdash \chi$, is correct regarding to logical consequence. In particular, every satisfiable set of formulas is GTC-consistent.

**Theorem 3.6.1. (Soundness)** *For any set of formulas* $\Gamma \cup \{\chi\}$, *if* $\Gamma \vdash \chi$ *is* GTC*-provable then* $\Gamma \models \chi$.

*Proof.* By induction on the length of the GTC-proof, it suffices to prove that every primitive rule of GTC (see Figure 3.27) is correct in the sense of preserving the logical consequence relation between the antecedent and the consequent.

The correctness proof of most rules is just routine. Actually, the only correctness proof that poses some difficulties is the proof of the rule $(\mathcal{U}\,L)_2$. Hence, we only give the details for this rule.

We prove, by contradiction, that if $\chi$ is a logical consequence of the antecedents of the premises of the rule $(\mathcal{U}\,L)_2$ then, $\chi$ is also a logical consequence of $\Delta\cup\{\varphi\,\mathcal{U}\,\psi\}$. Let us assume that $\chi$ is not a logical consequence of the set of formulas $\Delta\cup\{\varphi\,\mathcal{U}\,\psi\}$, i.e. the set $\Delta\cup\{\varphi\,\mathcal{U}\,\psi, \neg\chi\}$ is satisfiable. Then, by Proposition 3.3.3, the set $\Delta\cup\{\psi,\neg\chi\}$ or the set $\Delta\cup\{\varphi,\neg\psi,\circ((\varphi\wedge\neg(\Delta\cup\{\neg\chi\}))\,\mathcal{U}\,\psi),\neg\chi\}$ (at least one them) is satisfiable. Consequently, $\chi$ is not a logical consequence of $\Delta\cup\{\psi\}$ or $\chi$ is not a logical consequence of $\Delta\cup\{\varphi,\neg\psi,\circ((\varphi\wedge\neg(\Delta\cup\{\neg\chi\}))\,\mathcal{U}\,\psi)\}$. So that, we can build a countermodel for some of the two premises of the rule $(\mathcal{U}\,L)_2$. ■

$$
\dfrac{\dfrac{}{\Sigma_0 \vdash}\,(Cd_1) \quad \dfrac{\dfrac{}{\Sigma_1 \vdash}\,(Cd_1) \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma_2 \vdash}\,(Cd_1) \quad \dfrac{\dfrac{}{\Sigma_1 \vdash}\,(Cd_1) \quad \dfrac{\dfrac{\dfrac{}{\Sigma_3 \vdash}\,(Cd_1) \quad \dfrac{\dfrac{}{\Sigma_4 \vdash}\,(Cd_1) \quad \dfrac{}{\Sigma_5 \vdash}\,(Cd_1)}{p, \circ p, \circ\psi, \underline{\neg(\circ p \wedge \circ\psi)}, \varphi \vdash}\,(\neg\wedge)}{p, \circ p, \circ\psi, \underline{\neg(p \wedge \circ p \wedge \circ\psi)}, \varphi \vdash}\,(\neg\wedge)}{p, \circ p, \circ\psi, \underline{\neg\neg p}, \neg(p \wedge \circ p \wedge \circ\psi), \varphi \vdash}\,(\neg\neg)}{p, \circ p, \circ\psi, \underline{(\neg\Delta_0)\,\mathcal{U}\,\neg p} \vdash}\,(\mathcal{U})_2}{p, \underline{\neg p \vee \circ p}, \circ\Box(\neg p \vee \circ p), (\neg\Delta_0)\,\mathcal{U}\,\neg p \vdash}\,(\vee)}{p, \underline{\Box(\neg p \vee \circ p)}, (\neg\Delta_0)\,\mathcal{U}\,\neg p \vdash}\,(\Box)}{p, \circ p, \circ\Box(\neg p \vee \circ p), \circ((\neg\Delta_0)\,\mathcal{U}\,\neg p) \vdash}\,(\circ)}{p, \circ p, \circ\Box(\neg p \vee \circ p), \underline{\neg\neg p}, \circ((\neg\Delta_0)\,\mathcal{U}\,\neg p) \vdash}\,(\neg\neg)}{p, \circ p, \circ\Box(\neg p \vee \circ p), \underline{\Diamond\neg p} \vdash}\,(\Diamond)_2}{\dfrac{p, \underline{\neg p \vee \circ p}, \circ\Box(\neg p \vee \circ p), \Diamond\neg p \vdash}{p, \underline{\Box(\neg p \vee \circ p)}, \Diamond\neg p \vdash}\,(\Box)}\,(\vee)
$$

where
$\psi = \Box(\neg p \vee \circ p)$
$\varphi = \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)$
$\Delta_0 = \Delta_1 = \{p, \circ p, \circ\psi\}$
$\neg\Delta_0 = \neg\Delta_1 = \neg(p \wedge \circ p \wedge \circ\psi)$
$\Sigma_0 = \{\underline{p}, \underline{\neg p}, \circ\psi, \Diamond\neg p\}$
$\Sigma_1 = \{\underline{p}, \circ p, \circ\psi, \underline{\neg p}\}$
$\Sigma_2 = \{\underline{p}, \underline{\neg p}, \circ\psi, (\neg\Delta_0)\,\mathcal{U}\,\neg p\}$
$\Sigma_3 = \{\underline{p}, \circ p, \circ\psi, \underline{\neg p}, \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)\}$
$\Sigma_4 = \{p, \underline{\circ p}, \circ\psi, \underline{\neg\circ p}, \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)\}$
$\Sigma_5 = \{p, \circ p, \underline{\circ\psi}, \underline{\neg\circ\psi}, \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)\}$

*Figure 3.26:* TTC-proof for the set of formulas $\{p, \Box(\neg p \vee \circ p), \Diamond\neg p\}$

Rules for the Classical Connectives

$$(\neg L)\ \frac{\Delta \vdash \varphi}{\Delta, \neg\varphi \vdash \chi} \qquad (R\neg)\ \frac{\Delta, \varphi \vdash \mathbf{F}}{\Delta \vdash \neg\varphi}$$

$$(\wedge L)\ \frac{\Delta, \varphi, \psi \vdash \chi}{\Delta, \varphi \wedge \psi \vdash \chi} \qquad (R\wedge)\ \frac{\Delta \vdash \varphi \qquad \Delta \vdash \psi}{\Delta \vdash \varphi \wedge \psi}$$

Rules for the Temporal Connectives

$$(\neg\circ L)\ \frac{\Delta, \circ\neg\varphi \vdash \chi}{\Delta, \neg\circ\varphi \vdash \chi} \qquad (R\circ L)\ \frac{\mathsf{unnext}(\Delta) \vdash \varphi}{\Delta \vdash \circ\varphi} \qquad (R\circ\neg)\ \frac{\Delta \vdash \neg\circ\varphi}{\Delta \vdash \circ\neg\varphi}$$

$$(\mathcal{U}\,L)_1\ \frac{\Delta, \psi \vdash \chi \qquad \Delta, \varphi, \neg\psi, \circ(\varphi\,\mathcal{U}\,\psi) \vdash \chi}{\Delta, \varphi\,\mathcal{U}\,\psi \vdash \chi}$$

$$(R\,\mathcal{U})\ \frac{\Delta, \neg\varphi \vdash \psi \qquad \Delta, \varphi, \neg\circ(\varphi\,\mathcal{U}\,\psi) \vdash \psi}{\Delta \vdash \varphi\,\mathcal{U}\,\psi}$$

$$(\mathcal{U}\,L)_2\ \frac{\Delta, \psi \vdash \chi \qquad \Delta, \varphi, \neg\psi, \circ((\varphi \wedge \neg(\Delta \cup \{\neg\chi\}))\,\mathcal{U}\,\psi) \vdash \chi}{\Delta, \varphi\,\mathcal{U}\,\psi \vdash \chi}$$

Structural Rules

$$(As)\ \frac{}{\Delta, \varphi \vdash \varphi} \qquad (Cd)\ \frac{\Delta, \neg\varphi \vdash \mathbf{F}}{\Delta \vdash \varphi} \qquad (\circ\mathbf{F})\ \frac{\Delta \vdash \circ\mathbf{F}}{\Delta \vdash \chi}$$

*Figure 3.27:* Primitive Rules for the Sequent Calculus GTC

$$(\text{F}L)\ \frac{}{\Delta, \text{F} \vdash \chi} \qquad (CdL)\ \frac{}{\Delta, \varphi, \neg\varphi \vdash \chi} \qquad (\neg\neg L)\ \frac{\Delta, \varphi \vdash \chi}{\Delta, \neg\neg\varphi \vdash \chi}$$

$$(\neg\wedge L)\ \frac{\Delta, \neg\varphi \vdash \chi \qquad \Delta, \neg\psi \vdash \chi}{\Delta, \neg(\varphi \wedge \psi) \vdash \chi} \qquad (\circ L)\ \frac{\mathsf{unnext}(\Delta) \vdash \text{F}}{\Delta \vdash \chi}$$

$$(\neg\mathcal{U} L)\ \frac{\Delta, \neg\varphi, \neg\psi \vdash \chi \qquad \Delta, \varphi, \neg\psi, \neg\circ(\varphi\mathcal{U}\psi) \vdash \chi}{\Delta, \neg(\varphi\mathcal{U}\psi) \vdash \chi}$$

*Figure 3.28:* Some Derived GTC-Rules

The calculus GTC is more versatile than TTC, in particular GTC allows not only refutation proofs, but also goal-directed proofs or, in general, the consequent can directly be used as principal formula in GTC-proofs. As a consequence, in GTC, we can derive rules that have no sense in one-sided systems. For example, the contraposition rules:

$$(Cp1)\ \frac{\Delta, \neg\varphi \vdash \psi}{\Delta, \neg\psi \vdash \varphi} \qquad (Cp2)\ \frac{\Delta, \varphi \vdash \psi}{\Delta, \neg\psi \vdash \neg\varphi}$$

which can be derived in the usual way from the primitive GTC-rules for the classical connectives.

The derived rules in Figure 3.28 are useful for proving the completeness of GTC. They are easily derived with the help of the above rules $(Cp1)$ and $(Cp2)$. It is easy to check that $(\text{F}L)$ is derived from $(Cd)$ and $(As)$; $(CdL)$ from $(\neg L)$ and $(As)$; $(\circ L)$ from $(\circ\text{F})$ and $(R\circ L)$; $(\neg\neg L)$ from $(Cp1)$ and $(Cp2)$; $(\neg\wedge L)$ from $(Cp1)$ and $(R\wedge)$; and $(\neg\mathcal{U} L)$ from $(Cp1)$ and $(R\mathcal{U})$.

Now, we can associate to each TTC-proof a GTC-proof.

**Proposition 3.6.2.** *If $\Phi \vdash$ is TTC-provable then $\Phi \vdash \text{F}$ is GTC-provable.*

*Proof.* Suppose that $\Phi \vdash$ is TTC-provable. Then, by admissibility of the rule $(\neg\text{F})$ (see Section 3.5), $\Phi, \neg\text{F} \vdash$ is also TTC-provable.

It is easy to see that for each TTC-rule there is a closely related (primitive or derived) GTC-rule. In particular, TTC-rules are GTC-derived rules or single instances of GTC-rules. More precisely, the TTC-rules $(\neg\neg)$, $(\vee)$, $(\neg\vee)$, $(\circ)$, $(\mathcal{U})_1$, $(\mathcal{U})_2$, $(\neg\circ)$, $(\neg\mathcal{U})$, $(Cd_1)$ and $(Cd_2)$, respectively correspond to $(\neg\neg L)$, $(\vee L)$, $(\neg\vee L)$, $(\circ L)$, $(\mathcal{U} L)_1$, $(\mathcal{U} L)_2$, $(\neg\circ L)$, $(\neg\mathcal{U} L)$, $(CdL)$ and $(\text{F}L)$. As a consequence, we can construct a GTC-proof of the two-sided sequent $\Phi, \neg\text{F} \vdash \text{F}$. Therefore, using the GTC-rule $(Cd)$, the sequent $\Phi \vdash \text{F}$, is also GTC-provable. ■

**Theorem 3.6.3. (Completeness)** *For any set of formulas $\Gamma \cup \{\chi\}$, if $\Gamma \models \chi$ then $\Gamma \vdash \chi$ is GTC-provable.*

*Proof.* If $\Gamma \vdash \chi$ is not GTC-provable, then by rule $(Cd)$ the sequent $\Gamma \cup \{\neg\chi\} \vdash \text{F}$ is not GTC-provable. By Proposition 3.6.2, $\Gamma \cup \{\neg\chi\} \vdash$ is not TTC-provable, which is a contradiction by Theorem 3.5.3. ■

Using the abbreviations $\diamond\varphi$ and $\square\varphi$ for $\mathbf{T}\,\mathcal{U}\,\varphi$ and $\neg\diamond\neg\varphi$, respectively, we are also able to derive the following useful rules:

$$(\diamond L)_1\ \frac{\begin{array}{l}\Delta,\varphi\vdash\chi\\ \Delta,\neg\varphi,\circ(\mathbf{T}\,\mathcal{U}\,\varphi)\vdash\chi\end{array}}{\Delta,\diamond\varphi\vdash\chi}\qquad (\diamond L)_2\ \frac{\begin{array}{l}\Delta,\varphi\vdash\chi\\ \Delta,\neg\varphi,\circ(\neg(\Delta\cup\{\neg\chi\})\,\mathcal{U}\,\varphi)\vdash\chi\end{array}}{\Delta,\diamond\varphi\vdash\chi}$$

$$(R\diamond)\ \frac{\Delta,\neg\circ\diamond\varphi\vdash\varphi}{\Delta\vdash\diamond\varphi}\qquad (\square L)\ \frac{\Delta,\varphi,\circ\square\varphi\vdash\chi}{\Delta,\square\varphi\vdash\chi}$$

$$(R\square)_1\ \frac{\begin{array}{l}\Delta\vdash\varphi\\ \Delta,\circ(\mathbf{T}\,\mathcal{U}\,\neg\varphi)\vdash\neg\varphi\end{array}}{\Delta\vdash\square\varphi}\qquad (R\square)_2\ \frac{\begin{array}{l}\Delta\vdash\varphi\\ \Delta,\circ(\neg\Delta\,\mathcal{U}\,\neg\varphi)\vdash\neg\varphi\end{array}}{\Delta\vdash\square\varphi}$$

In addition, the TTC-rules $(\mathcal{U})_3$ and $(\diamond)_3$ produce the corresponding GTC-rules where $\Delta' = \Delta\cup\{\neg\chi\}$:

$$(\mathcal{U}\,L)_3\ \frac{\begin{array}{l}\Delta,\psi\vdash\chi\\ \Delta,\varphi,\neg\psi,\circ((\varphi\Cap\widetilde{\Delta'})\,\mathcal{U}\,\psi)\vdash\chi\end{array}}{\Delta,\varphi\,\mathcal{U}\,\psi\vdash\chi}\qquad (\diamond L)_3\ \frac{\begin{array}{l}\Delta,\varphi\vdash\chi\\ \Delta,\neg\varphi,\circ(\widetilde{\Delta'}\,\mathcal{U}\,\varphi)\vdash\chi\end{array}}{\Delta,\diamond\varphi\vdash\chi}$$

and it is easy to derive the following rule $(R\square)_3$ for the defined connective $\square$:

$$(R\square)_3\ \frac{\begin{array}{l}\Delta\vdash\varphi\\ \Delta,\circ(\widetilde{\Delta}\,\mathcal{U}\,\neg\varphi)\vdash\neg\varphi\end{array}}{\Delta\vdash\square\varphi}$$

Note that, by $(\square L)$ and $(CdL)$, the following contradiction rule is also derivable:

$$(Cd\square)\ \frac{}{\Delta,\square\varphi,\neg\circ\square\varphi\vdash\chi}$$

Let us now illustrate the GTC-style of reasoning by means of some examples of GTC-proofs. In order to enhance readability, we underline, at each step, the principal formula. However, we do not underline any formula in the applications of the rules $(R\circ L)$, $(\circ\mathbf{F})$ and $(\circ L)$. Both primitive and derived rules are used in the derivations.

**Example 3.6.4.** *The* GTC*-proof in Figure 3.29 shows that the formula $q$ is a logical consequence of the set of formulas $\{p\,\mathcal{U}\,q,\ \neg\circ\diamond q\}$. This* GTC*-proof is similar to the* TTC*-proof showed in Example 3.5.5 (Figure 3.24).*

*Note that in the first application (from the bottom) of the rule $(\mathcal{U}\,L)_3$ the persistent formula $\circ\neg\diamond q$ is left out of the negation of the context. In the second application of the rule $(\mathcal{U}\,L)_3$ we obtain, in the right-hand premise, the formula $\circ(\mathbf{F}\,\mathcal{U}\,q)$ because we dispense with the persistent formula $\circ\neg\diamond q$ and because the negation of $\neg q$ is a conjunct of the subformula $p\wedge\neg\neg q$ in the formula $(p\wedge\neg\neg q)\,\mathcal{U}\,q$.*

$$\begin{array}{c}\dfrac{\dfrac{}{\underline{q}, \circ\neg\Diamond q \vdash \underline{q}}\,(As) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{\dfrac{}{\underline{q}, \neg\circ\Diamond q \vdash \underline{q}}\,(As) \qquad \dfrac{\dfrac{}{p, \underline{\neg\neg q}, \underline{\neg q}, \circ(\mathbf{F}\,\mathcal{U}\,q), \circ\neg\Diamond q \vdash q}\,(CdL)}{\underline{p \wedge \neg\neg q}, \neg q, \circ(\mathbf{F}\,\mathcal{U}\,q), \circ\neg\Diamond q \vdash q}\,(\wedge)}{\underline{(p \wedge \neg\neg q)\,\mathcal{U}\,q}, \circ\neg\Diamond q \vdash q}\,(\mathcal{U}\,L)_3 \\ \hline\hline (p \wedge \neg\neg q)\,\mathcal{U}\,q, \underline{\neg\circ\Diamond q} \vdash q \end{array}\,(\neg\circ L)}{(p \wedge \neg\neg q)\,\mathcal{U}\,q \vdash \underline{\Diamond q}}\,(R\Diamond)}{(p \wedge \neg\neg q)\,\mathcal{U}\,q, \underline{\neg\Diamond q} \vdash \mathbf{F}}\,(\neg L)}{p, \neg q, \circ((p \wedge \neg\neg q)\,\mathcal{U}\,q), \circ\neg\Diamond q \vdash \circ\mathbf{F}}\,(R\circ L)}{p, \neg q, \circ((p \wedge \neg\neg q)\,\mathcal{U}\,q), \circ\neg\Diamond q \vdash q}\,(\circ\mathbf{F})}{\underline{p\,\mathcal{U}\,q}, \circ\neg\Diamond q \vdash q}\,(\mathcal{U}\,L)_3 \\ \hline\hline p\,\mathcal{U}\,q, \underline{\neg\circ\Diamond q} \vdash q \end{array}\,(\neg\circ L)$$

*Figure 3.29:* GTC-proof that shows that the formula $q$ is a logical consequence of $\{p\,\mathcal{U}\,q, \neg\circ\Diamond\, q\}$

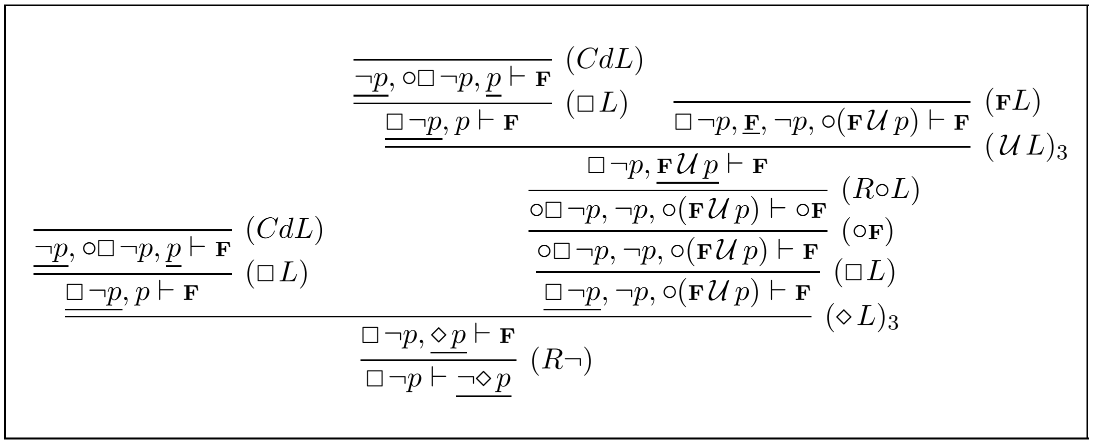
**Example 3.6.5.** *The* GTC*-proof in Figure 3.30 shows that the formula $\neg\Diamond\, p$ is a logical consequence of the set of formulas $\{\Box\neg p\}$. This* GTC*-proof is similar to the* TTC*-proof in Example 3.5.6 (Figure 3.25).*

*Note that, when applying the rule $(\Diamond\, L)_3$ and $(\mathcal{U}\,L)_3$, the persistent formulas $\Box\neg p$ and $\neg\mathbf{F}$ are left out of the negation of the context. As in the case of the* TTC*-proof in Example 3.5.6 (Figure 3.25), this* GTC*-proof does not strictly follow the strategy presented by means of the systematic tableau algorithm in Figure 3.9, because the rules $(\mathcal{U}\,L)_3$ and $(\Diamond L)_3$ are applied to sets of formulas that are non-elementary.*

**Example 3.6.6.** *By means of the* GTC*-proof in Figure 3.31, we show that the formula $\Box p$ is a logical consequence of the set of formulas $\{p, \Box(\neg p \vee \circ p)\}$. The sets $\Sigma_i$, with $i \in \{0, \ldots, 5\}$, are inconsistent since they contain a formula and its negation and the derived rule $(CdL)$ is applied to each of them. Although the structure of the proof is the same as the* TTC*-proof in Figure 3.26 of Example 3.5.7, the set $\Sigma_0$ is different and the set $\Sigma_1$ appears only once. In the place of the first appearance (from the left) of the set $\Sigma_1$ in Figure 3.26 of Example 3.5.7 now, in Figure 3.31, we use the structural rule $(As)$.*

*Note that, since we use $(R\Box)_2$ and $(\mathcal{U}\,L)_2$, the persistent formula $\circ\psi = \circ\Box(\neg p \vee \circ p)$ is included in the negation of the context and that repetitions are not avoided in the formula $\varphi = \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)$. However, the formula $\circ\psi$ could be left out of the negation of the context and the repetition in $\varphi$ could also be avoided as shown in Example 3.6.7.*

**Example 3.6.7.** *As well as the proof given in Example 3.6.6, the* GTC*-proof in Figure 3.32 shows that the formula $\Box p$ is a logical consequence of $\{p, \Box(\neg p \vee \circ p)\}$. However, in Figure 3.32 we use the rules $(R\Box)_3$ and $(\mathcal{U}\,)_3$ whereas in Figure 3.31 the rules $(R\Box)_2$ and $(\mathcal{U}\,)_2$ are used. Additionally, the approach in Figure 3.32 is more "goal-directed" in the sense that in order to prove that the formula $\Box p$ follows from the set of formulas $\{p, \Box(\neg p \vee \circ p)\}$, in the first derivation step $\Box p$ is the principal formula. By contrast, the* GTC*-proof in Figure 3.31 is*



$\neg p, \circ\Box\neg p, p \vdash \mathbf{F}$ $(CdL)$
$\Box\neg p, p \vdash \mathbf{F}$ $(\Box L)$
$\Box\neg p, \mathbf{F}, \neg p, \circ(\mathbf{F}\,\mathcal{U}\,p) \vdash \mathbf{F}$ $(\mathbf{F}L)$
$\Box\neg p, \mathbf{F}\,\mathcal{U}\,p \vdash \mathbf{F}$ $(\mathcal{U}\,L)_3$
$\circ\Box\neg p, \neg p, \circ(\mathbf{F}\,\mathcal{U}\,p) \vdash \circ\mathbf{F}$ $(R\circ L)$
$\circ\Box\neg p, \neg p, \circ(\mathbf{F}\,\mathcal{U}\,p) \vdash \mathbf{F}$ $(\circ\mathbf{F})$
$\Box\neg p, \neg p, \circ(\mathbf{F}\,\mathcal{U}\,p) \vdash \mathbf{F}$ $(\Box L)$
$\Box\neg p, \Diamond p \vdash \mathbf{F}$ $(\Diamond L)_3$
$\Box\neg p \vdash \neg\Diamond p$ $(R\neg)$

*Figure 3.30:* GTC-proof that shows that the formula $\neg\Diamond p$ is a logical consequence of $\{\Box\neg p\}$

*a direct adaptation of the* TTC*-proof in Figure 3.26 and is not driven by the "goal", i.e. by the formula* $\Box p$*, which we want to proof from* $\{p, \Box(\neg p \vee \circ p)\}$.

*The* GTC*-proof in Figure 3.32 does not strictly follow the strategy presented by means of the systematic tableau algorithm in Figure 3.9. In order to follow such strategy, either the rule* $(\mathcal{U}\,L)_3$ *or the rule* $(\mathcal{U}\,L)_2$ *should be used instead of the rule* $(\mathcal{U}\,L)_1$.

## 3.7 Related Work

In Section 3.1 we have briefly surveyed the main representatives of the different approaches in the tableau and sequent frameworks. In this section we add more details about these related proposals.

### 3.7.1 Tableau Systems

The traditional tableau methods for temporal logic (e.g. [128, 73, 8, 87, 79, 81]) are based on the usual inductive definitions of the temporal connectives. A traditional rule system for the tableau framework can be obtained from TTM (Figure 3.1) by just removing the rule $(\mathcal{U})_2$. In such systems an auxiliary graph is built in a first pass. For instance, an auxiliary graph for the set of formulas $\{p, \Box(\neg p \vee \circ p), \Diamond\neg p\}$ is very similar to the right-most branch of the tableau in Figure 3.14 $(a)$. Edges would be directed downwards and instead of the last node of the branch, there would be an edge from the previous node to the root node. So that, the whole auxiliary graph would be a strongly connected component made up of five nodes. The second pass serves to check whether an infinite path that yields a model for the root set can be built from the graph. With that purpose, maximal strongly connected components that are not fulfilling for some eventuality and from which no other maximal strongly connected component can be reached, are deleted. This process is repeated until no node can be eliminated. In the above mentioned example, the only maximal strongly connected component (i.e. the whole auxiliary graph) would be removed because it is not fulfilling for the eventuality $\Diamond\neg p$ and no other nodes can be reached from it. Since the result would be an empty graph, the root set would be classified as unsatisfiable. Our tableau method TTM is one-pass. In Figure 3.12 it

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{$(CdL)$}
\UnaryInfC{$\Sigma_0 \vdash \Box p$}
\AxiomC{}
\RightLabel{$(As)$}
\UnaryInfC{$\underline{p}, \circ p, \circ\psi \vdash \underline{p}$}
\AxiomC{}
\RightLabel{$(CdL)$}
\UnaryInfC{$\Sigma_2 \vdash \mathbf{F}$}
\AxiomC{}
\RightLabel{$(CdL)$}
\UnaryInfC{$\Sigma_1 \vdash \mathbf{F}$}
\AxiomC{}
\RightLabel{$(CdL)$}
\UnaryInfC{$\Sigma_3 \vdash \mathbf{F}$}
\AxiomC{}
\RightLabel{$(CdL)$}
\UnaryInfC{$\Sigma_4 \vdash \mathbf{F}$}
\AxiomC{}
\RightLabel{$(cdL)$}
\UnaryInfC{$\Sigma_5 \vdash \mathbf{F}$}
\RightLabel{$(\neg\wedge)$}
\BinaryInfC{$p, \circ p, \circ\psi, \underline{\neg(\circ p \wedge \circ\psi)}, \varphi \vdash \mathbf{F}$}
\RightLabel{$(\neg\wedge)$}
\BinaryInfC{$p, \circ p, \circ\psi, \underline{\neg(p \wedge \circ p \wedge \circ\psi)}, \varphi \vdash \mathbf{F}$}
\RightLabel{$(\neg\neg L)$}
\UnaryInfC{$p, \circ p, \circ\psi, \underline{\neg\neg p}, \neg(p \wedge \circ p \wedge \circ\psi), \varphi \vdash \mathbf{F}$}
\RightLabel{$(\mathcal{U}\, L)_2$}
\BinaryInfC{$p, \circ p, \circ\psi, \underline{(\neg\Delta_0)\,\mathcal{U}\,\neg p} \vdash \mathbf{F}$}
\RightLabel{$(\vee)$}
\BinaryInfC{$p, \underline{\neg p \vee \circ p}, \circ\Box(\neg p \vee \circ p), (\neg\Delta_0)\,\mathcal{U}\,\neg p \vdash \mathbf{F}$}
\RightLabel{$(\Box L)$}
\UnaryInfC{$p, \underline{\Box(\neg p \vee \circ p)}, (\neg\Delta_0)\,\mathcal{U}\,\neg p \vdash \mathbf{F}$}
\RightLabel{$(\circ L)$}
\UnaryInfC{$p, \circ p, \circ\Box(\neg p \vee \circ p), \circ((\neg\Delta_0)\,\mathcal{U}\,\neg p) \vdash \neg p$}
\RightLabel{$(R\Box)_2$}
\BinaryInfC{$p, \circ p, \circ\Box(\neg p \vee \circ p) \vdash \underline{\Box p}$}
\RightLabel{$(\vee)$}
\BinaryInfC{$p, \underline{\neg p \vee \circ p}, \circ\Box(\neg p \vee \circ p) \vdash \Box p$}
\RightLabel{$(\Box L)$}
\UnaryInfC{$p, \underline{\Box(\neg p \vee \circ p)} \vdash \Box p$}
\end{prooftree}
$$

where
$$
\begin{aligned}
\psi &= \Box(\neg p \vee \circ p)\\
\varphi &= \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)\\
\Delta_0 = \Delta_1 &= \{p, \circ p, \circ\psi\}\\
\neg\Delta_0 = \neg\Delta_1 &= \neg(p \wedge \circ p \wedge \circ\psi)\\
\Sigma_0 &= \{\underline{p}, \underline{\neg p}, \circ\psi\}\\
\Sigma_1 &= \{\underline{p}, \circ p, \circ\psi, \underline{\neg p}\}\\
\Sigma_2 &= \{\underline{p}, \underline{\neg p}, \circ\psi, (\neg\Delta_0)\,\mathcal{U}\,\neg p\}\\
\Sigma_3 &= \{\underline{p}, \circ p, \circ\psi, \underline{\neg p}, \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)\}\\
\Sigma_4 &= \{p, \underline{\circ p}, \circ\psi, \underline{\neg\circ p}, \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)\}\\
\Sigma_5 &= \{p, \circ p, \underline{\circ\psi}, \underline{\neg\circ\psi}, \circ((\neg\Delta_0 \wedge \neg\Delta_1)\,\mathcal{U}\,\neg p)\}
\end{aligned}
$$

*Figure 3.31:* GTC-proof that shows that the formula $\Box\, p$ is a logical consequence of the set of formulas $\{p, \Box(\neg p \vee \circ p)\}$ (1st version)

$$
\dfrac{
  \dfrac{}{\underline{p}, \psi \vdash \underline{p}}\,(As)
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{}{p, \underline{\neg p}, \circ\psi, \circ(\neg p\,\mathcal{U}\,\neg p) \vdash \underline{\neg p}}\,(As)
      \qquad
      \dfrac{
        \dfrac{
          \dfrac{}{\underline{p}, \psi, \underline{\neg p} \vdash \mathbf{F}}\,(CdL)
          \qquad
          \dfrac{}{p, \psi, \underline{\neg p, \neg\neg p}, \circ(\neg p\,\mathcal{U}\,\neg p) \vdash \mathbf{F}}\,(CdL)
        }{p, \Box\,(\neg p \vee \circ p), \underline{\neg p\,\mathcal{U}\,\neg p} \vdash \mathbf{F}}\,(\mathcal{U}\,L)_1
      }{p, \circ p, \circ\Box\,(\neg p \vee \circ p), \circ(\neg p\,\mathcal{U}\,\neg p) \vdash \neg p}\,(\circ L)
    }{p, \underline{\neg p \vee \circ p}, \circ\Box\,(\neg p \vee \circ p), \circ(\neg p\,\mathcal{U}\,\neg p) \vdash \neg p}\,(\vee L)
  }{p, \underline{\Box\,(\neg p \vee \circ p)}, \circ(\neg p\,\mathcal{U}\,\neg p) \vdash \neg p}\,(\Box\, L)
}{p, \Box\,(\neg p \vee \circ p) \vdash \underline{\Box\, p}}\,(R\Box)_3
$$

where $\psi$ is $\Box\,(\neg p \vee \circ p)$

*Figure 3.32:* GTC-proof that shows that the formula $\Box\, p$ is a logical consequence of the set of formulas $\{p, \Box\,(\neg p \vee \circ p)\}$ (2nd version)

can be appreciated that, by using the TTM-rules $(\diamond)_2$ and $(\mathcal{U})_2$ and by following the systematic tableau algorithm in Figure 3.9, we are able to close all the branches without a second pass.

The first one-pass tableau method for PLTL was presented by Schwendimann in [117] and it avoids the second pass by adding extra information to the nodes in the tableau. This method is also based on the usual inductive definition of the temporal connectives. As in TTM, branches can be seen as sequences of stages $s_0, s_1, \dots, s_i$ where each stage $s_j$ is a sequence of nodes $n_j^0, n_j^1, \dots, n_j^{k_j}$. Each application of the rule $Nexttime$ (which corresponds to an application of the operator unnext in TTM) to an elementary set of formulas gives rise to a new stage. Each node $n_j^h$ of a tableau is labelled with a triple of the form $(\Gamma_j^h, Save_j^h, Res_j^h)$ where

- $\Gamma_j^h$ is a finite set of formulas.
- $Save_j^h$ is a pair of the form $(Ev_j^h, Br_j)$ that serves to store history information. More precisely, $Ev_j^h$ is a set of formulas representing the eventualities that are fulfilled in the nodes $n_j^0, n_j^1, \dots, n_j^h$, and $Br_j$ (which only depends on the stage) is the sequence of pairs $(\Gamma_0^{k_0}, Ev_0^{k_0}), (\Gamma_1^{k_1}, Ev_1^{k_1}), \dots, (\Gamma_{j-1}^{k_{j-1}}, Ev_{j-1}^{k_{j-1}})$ representing the current branch. Note that $\Gamma_g^{k_g}$ is the set of formulas of the last node of the stage $s_g$ for every $g \in \{0, \dots, j-1\}$.
- $Res_j^h$ is a pair of the form $(r_j^h, uev_j^h)$ that is used to store partial result information. More precisely, $r_j^h$ is a natural number that represents the earliest node $n_x^y$ (i.e. $x \leq j$) that is reachable from $n_j^h$. On the other hand, $uev_j^h$ is the set formed by the <u>u</u>nfulfilled <u>ev</u>entualities of the current branch.

The information in $Save_j^h$ is produced in a top-down manner, from parent to child, whereas the information in $Res_j^h$ is synthesized bottom-up, from children to parent. The bottom-up synthesis of information starts once a terminal rule is applied (i.e. a leaf is generated). The information synthesized bottom-up in $Res_j^h$ is needed because an eventuality that appears in a cyclic branch but is not fulfilled directly in such branch, can be fulfilled in some other reachable branch.

In TTM the fulfillment of an eventuality depends only on one branch. Consequently, given a satisfiable set of formulas as root set, an expanded open branch yields a model for the root set in TTM whereas in [117] more than one cyclic branch may be required to obtain a cycle that gives rise to a model for the set of formulas in the root. Additionally, nodes in TTM do not require so much extra information. Moreover, given an unsatisfiable set of formulas, instead of expanded non-fulfilling cyclic branches, TTM obtains closed branches (whose last nodes are inconsistent, see Definition 3.3.5). For instance, if we consider the set of formulas $\{p, \square(\neg p \vee \circ p), \diamond\neg p\}$, Schwendimann's tableau method would obtain a tree that contains the same nodes as the tableau in Figure 3.14 $(a)$, but with the above indicated extra information in each node. Then, in the right-most branch, the bottom-up synthesis would be necessary to detect that $\diamond\neg p$ cannot be fulfilled. By contrast, TTM obtains the closed tableau showed by means of Figures 3.12 and 3.13. The rule $(\mathcal{U})_2$ together with the strategy expressed by means of the systematic tableau algorithm in Figure 3.9 are the key for this different deductive approach for PLTL.

In order to detect whether an open cyclic branch is expanded, i.e. in order to decide whether a cyclic branch is fulfilling, the systematic tableau algorithm for TTM does not directly check whether each eventuality is fulfilled, instead it checks whether the eventualities that belong to the first node of each stage of the cycle have been selected along the cycle (see Definition 3.3.12

and Remarks 3.3.13 and 3.4.8). This is another remarkable difference with respect to the above mentioned approaches.

The complexity of the two-pass methods is exponential (even in the average case) due to the fact that the size of the graph is exponential in the size of the set of formulas in the root, although some improvements such as not building the unreachable nodes can be considered (see e.g. [81]). The worst case complexity for Schwendimann's tableau method and TTM is doubly exponential. However, it has been shown by means of experimental analysis (see e.g. [76]) that, in some cases, doubly exponential algorithms can outperform exponential ones because the occurrence of the worst case in the doubly exponential algorithms is rare. We are convinced that a practical implementation that incorporates the simplifications explained in Section 3.4.5 may compete with traditional methods in several cases –e.g. when most of the formulas (in the context) are always-formulas– and even be faster in others, e.g. when satisfiability can be detected without constructing the whole graph. Of course, a lot of experimental work needs to be done in order to precisely compare the performance of these different approaches. As a first step, we have implemented a preliminary prototype for the TTM tableau method which is available online in `http://www.sc.ehu.es/jiwlucap/TTM.html`. A report about the implementation details of this prototype for the TTM tableau method can be found in [63].

### 3.7.2 Sequent Systems

The sequent calculus $\mathcal{FC}$ introduced in [60] is the first finitary sequent calculus for PLTL that dispenses with the cut rule and also with invariant-based rules. This cut-freeness and invariant-freeness is achieved by means of the rule $(\mathcal{U})_2$ and the strategy represented by the systematic tableau algorithm for TTM. The sequent calculus $\mathcal{FC}$ is very similar to GTC. However, in order to prove the completeness of $\mathcal{FC}$, the weakening rule $(Wk)$, as well as a hidden contraction, are used in [60] (in Lemma 22 and Lemma 11, respectively). By contrast, the sequent calculus GTC is weakening-free and contraction-free. In this sense, the completeness result obtained for GTC is an improvement of the completeness result obtained for $\mathcal{FC}$. This improvement is achieved by using the duality of GTC with the tableau system TTM.

Traditional sequent systems include either an infinitary rule or an invariant-based rule. For instance, in one of the sequent calculi presented in [105] we can find an infinitary rule that, in terms of this dissertation, i.e., with non-relevant minor syntactical changes, is as follows:

$$\frac{\Delta, \varphi \vdash \chi \qquad \Delta, \circ\varphi \vdash \chi \qquad \Delta, \circ^2\varphi \vdash \chi \qquad \dots}{\Delta, \diamond\varphi \vdash \chi}$$

Note that the above rule contains infinite premises.

We can also find an invariant-based rule in the sequent calculi introduced in [105] which, in our notation, can be presented as follows:

$$\frac{\Delta \vdash \psi \qquad \psi \vdash \circ\psi \qquad \psi \vdash \varphi}{\Delta \vdash \Box\varphi}$$

The above formula $\psi$ is called the invariant formula. These kinds of invariant-based rules require an additional search for the invariant that is not addressed by the sequent calculi. A similar invariant-based rule can be found in e.g. [104, 121].

The cut-free and also invariant-free sequent calculus LT2 for PLTL introduced in [20] is right handed. So that, sequents are of the form $\vdash \psi_1, \dots, \psi_n$. The meaning of a sequent $\vdash \psi_1, \dots, \psi_n$

is $\mathrm{T} \to \psi_1 \vee \ldots \vee \psi_n$ or equivalently $\psi_1 \vee \ldots \vee \psi_n$. This sequent calculus is dual to our sequent calculus TTC in the sense that a TTC-proof for $\Delta \vdash$ states that $\Delta$ is unsatisfiable whereas an LT2-proof for $\vdash \Delta$ states that $\Delta$ is valid, i.e., $\Delta$ is true in every state of every PLTL-structure. Additionally, the LT2-rule that corresponds to the TTM-rule $(\mathcal{U})_2$ deals with annotated formulas of the form $\varphi \mathcal{R}_H \psi$, [7] where the annotation or history $H$ is a finite set of finite sets of formulas $\{\Gamma_1, \ldots, \Gamma_n\}$. If $\Gamma_i = \{\varphi_1, \ldots, \varphi_m\}$, the meaning of $\Gamma_i$ is $\varphi_1 \vee \ldots \vee \varphi_m$ and the corresponding formula for $H$ is $\Gamma_1 \wedge \ldots \wedge \Gamma_n$. The formula represented by the annotated formula $\varphi \mathcal{R}_H \psi$ is $(\varphi \vee \neg H) \mathcal{R} (\psi \vee \neg H)$. The key rule that deals with the annotated formulas is as follows:

$$\frac{\vdash \Delta, \psi, \neg H \qquad \vdash \Delta, \circ(\varphi \mathcal{R}_{H,\Delta} \psi), \varphi, \neg H}{\vdash \Delta, \varphi \mathcal{R}_H \psi}$$

where $H, \Delta$ means $H \cup \{\Delta\}$ in the subindex of the connective $\mathcal{R}$. This rule is similar to our rule $(\mathcal{U})_2$. As already mentioned in Section 3.1, the idea behind the way in which eventualities are dealt with and the strategy that leads to completeness coincide in the sequent calculi LT2 and TTC, even in the fairness requirement in the selection of eventualities. However, unlike TTC, the sequent calculus LT2 incorporates the selection of eventualities in the rule system by means of a rule that carries out the selection of an eventuality by generating an annotated formula $\varphi \mathcal{R}_\emptyset \psi$ from a formula $\varphi \mathcal{R} \psi$. Additionally, the strategy of sticking to a selected eventuality –which is an annotated formula– until it is fulfilled, is also incorporated in the system sequent by not allowing more than one annotated formula in each sequent of a derivation. Note also that annotated formulas do not belong to the logical language. In other words, an additional variable –or annotation– for saving the history is used in LT2 whereas in TTC all the formulas belong to the logical language and no extra variable is used for history management. Moreover, in TTC, the restrictions that lead to completeness are not incorporated in the sequent system. As a consequence, we allow different strategies and different derivations, although we follow the systematic tableau algorithm to guarantee completeness.

[7] Note that the use of the connective $\mathcal{R}$ on the right-hand side of a sequent corresponds to the use of the connective $\mathcal{U}$ on the left-hand side. So that, a formula of the form $\varphi \mathcal{R} \psi$ on the right-hand side of a sequent represents an eventuality.

# 4. INVARIANT-FREE CLAUSAL TEMPORAL RESOLUTION FOR PLTL

## *4.1 Introduction*

In this chapter, we deal with clausal resolution for PLTL. The method of resolution, invented by J.A. Robinson in 1965 ([111]), is an efficient refutation proof method that has provided the basis for several well-known theorem provers for classical logics. As well as tableau methods, in the case of decidable logics, resolution methods yield decision procedures for the satisfiability of sets of formulas.

Different approaches have been proposed in the literature for adapting the classical resolution method to temporal logic but without consensus in the clausal normal form or in the temporal resolution itself. The earliest temporal resolution method [1] uses a non-clausal approach, hence a large number of rules are required for handling general formulas instead of clauses. There is also early work (e.g. [12, 29]) related to clausal resolution for (less expressive) sublogics of PLTL. In the language considered in [12] there are no eventualities at all, whereas in [29] the authors consider the strictly less expressive sublanguage of PLTL defined by using only $\circ$ and $\diamond$ as temporal connectives. The early clausal method presented in [126] tackles full PLTL and uses a clausal form similar to ours, but completeness is only achieved in absence of eventualities. More recently, a fruitful trend of clausal temporal resolution methods, starting with the seminal paper of M. Fisher [40], achieves completeness for full PLTL by means of a specialized *temporal resolution* rule that needs to generate an invariant formula from a set of clauses that behaves as a loop. The methods and techniques developed in such an approach have been successfully adapted to Computation Tree Logic (CTL) (see [18]) and some extensions of CTL such as ECTL and ECTL$^+$ (see [17, 16]), but not to Full Computation Tree Logic (CTL$^\star$). It is remarkable that the clausal normal forms used in [12], [29], [126] and [40] are quite different.

In this thesis, we introduce a new clausal resolution method that is sound and complete for full PLTL. Our method is based on the dual methods of tableaux and sequents for PLTL presented in the previous chapter. On this basis we are able to perform clausal resolution in the presence of eventualities avoiding the requirement of invariant generation. We define a notion of *clausal normal form* and prove that every PLTL-formula can be translated into an equisatisfiable set of clauses. Our resolution mechanism explicitly simulates the transition from one world to the next one. Inside each world, we apply two kinds of rules: (1) the resolution and subsumption rules and (2) the fixpoint rules for decomposing temporal literals. The latter split a clause with a temporal literal into a finite number of new clauses. We prove that the method is sound and complete. In fact, it finishes for any set of clauses deciding its (un)satisfiability, hence it gives rise to a new decision procedure for PLTL. In Section 4.8 we compare our approach with the methods in [29, 1, 126, 40]. We also give more details on the relation between TRS-resolution and the TTM tableau method that is its forerunner.

*Outline of the chapter.* In Section 4.2 we introduce the syntactic notion of clause (Subsection 4.2.1), we show that any PLTL-formula can be transformed into a set of clauses (Subsection 4.2.2) and we analyze the complexity of this transformation (Subsection 4.2.3). In Section 4.3 we introduce the system TRS of inference rules in two subsections: the first one presents the basic rules and the second one presents the rule for solving eventualities in a way that prevents their indefinite delay. Then, in Section 4.4, we present the notion of TRS-derivation, provide some sample derivations and study the relationship between TRS-resolution and classical (propositional) resolution. The soundness of TRS is proved in Section 4.5. In Section 4.6 we propose an algorithm for systematically obtaining, for any set of clauses $\Gamma$, a finite derivation that proves that $\Gamma$ is either satisfiable or unsatisfiable. We also show some examples of application of the algorithm in Subsection 4.6.2. An important issue for this algorithm is to prove its termination for every input. This proof is presented in Subsection 4.6.3. In Subsection 4.6.4 we provide a bound of the worst-case complexity of the algorithm. In Section 4.7, we prove the completeness of TRS-resolution on the basis of the algorithm that outputs a finite derivation for every set of clauses. Finally, in Section 4.8 we discuss significant related work.

## 4.2 The Clausal Language

In this section we first define the conjunctive normal form of a formula. This is the basis for our notion of clause. In the second subsection we explain how to convert any formula into a set of clauses. Thirdly, we give the worst case complexity of the translation.

### 4.2.1 Conjunctive Normal Form for Formulas

Our notion of literal extends the classical notion of propositional literal. This extension introduces both temporal literals and (possibly empty) prefixed chains of the connective $\circ$ in front of temporal and propositional literals. That is, using the usual BNF-notation:

$$P ::= p \mid \neg p$$
$$T ::= P_1 \mathcal{U} P_2 \mid P_1 \mathcal{R} P_2 \mid \diamond P \mid \square P$$
$$L ::= \circ^i P \mid \circ^i T$$

where $p \in \mathsf{Prop}$ and $i \in I\!N$. $P$ stands for a propositional literal, $T$ for a (basic) temporal literal and $L$ for a literal. In the sequel, we use the term literal in the latter sense and only if needed we will specify whether a literal is propositional or temporal.[1] Sub- and superscripts are used when necessary.

We extend the classical notion of *the complement* $\widetilde{L}$ *of a literal* $L$ as follows:

$$\widetilde{p} = \neg p, \quad \widetilde{\neg p} = p, \quad \widetilde{\circ L} = \circ \widetilde{L}, \quad \widetilde{P_1 \mathcal{U} P_2} = \widetilde{P_1} \mathcal{R} \widetilde{P_2} \quad \text{and} \quad \widetilde{P_1 \mathcal{R} P_2} = \widetilde{P_1} \mathcal{U} \widetilde{P_2}$$

It is easy to see that $\widetilde{\square P} = \diamond \widetilde{P}$ and $\widetilde{\diamond P} = \square \widetilde{P}$. Although $\diamond P$ and $\square P$ can be respectively defined by $\widetilde{P} \mathcal{U} P$ and $\widetilde{P} \mathcal{R} P$, we have intentionally introduced $\diamond P$ and $\square P$ as temporal literals because of technical convenience.

A *now-clause* $N$ is a finite disjunction of literals (above denoted by $L$):

$$N ::= \bot \mid L \vee N$$

[1] Note that $\circ$ is the only temporal connective that does not occur in the so-called (basic) temporal literals.

where $\bot$ represents the empty disjunction (or the empty now-clause). We identify finite disjunctions of literals with sets of literals. Hence, we assume that there are neither repetitions nor any established order in the literals of a clause. This assumption is especially advantageous for presenting the resolution rule, because it avoids factoring and ordering problems. However, for readability, we always write the disjunction symbol between the literals of a clause.

A clause is either a now-clause or a now-clause preceded by the connective $\Box$

$$C ::= N \mid \Box N$$

A clause of the form $\Box N$ is called an *always-clause*. In this chapter, we use the superscript $b$ varying in $\{0,1\}$ to represent a formula with or without a prefixed unary connective (in particular a clause with or without a prefixed $\Box$). For instance, $\Box^b\varphi$ is $\Box\varphi$ whenever $b$ is $1$ and $\varphi$ whenever $b$ is $0$. Along the rest of the chapter superscripts starting by $b$ (from bit) range in $\{0,1\}$. These kinds of superscripts are notation, hence they are not part of the syntax. Note that the formula $\Box^b\bot$ represents the two possible syntactic forms of the empty clause, as now- or always-clause.

For a clause $C = \Box^b(L_1 \vee \ldots \vee L_n)$ we denote by $\mathsf{Lits}(C)$ the set $\{L_1, \ldots, L_n\}$ and for a set of clauses $\Gamma$ we denote by $\mathsf{Lits}(\Gamma)$ the set $\bigcup_{C\in\Gamma} \mathsf{Lits}(C)$.

**Definition 4.2.1.** *The set of all clauses in* $\Gamma$ *that contain the literal* $L$ *is denoted by* $\Gamma \restriction \{L\}$*, i.e.* $\Gamma \restriction \{L\} = \{C \in \Gamma \mid L \in \mathsf{Lits}(C)\}$.

Since $\circ$ distributes over disjunction, for a given now-clause $N = L_1 \vee \ldots \vee L_n$, we denote by $\circ N$ the now-clause $\circ L_1 \vee \ldots \vee \circ L_n$. We say that a clause $C$ is $\circ$-free if $\mathsf{Lits}(C)$ does not contain any literal of the form $\circ L$.

**Definition 4.2.2.** *Given a set of clauses* $\Gamma$*, we define* $\mathsf{alw}(\Gamma) = \{\Box N \mid \Box N \in \Gamma\}$ *and* $\mathsf{now}(\Gamma) = \Gamma \setminus \mathsf{alw}(\Gamma)$.

Note that a formula of the form $\Box P$, can be understood as a now-clause consisting of one temporal literal or as an always-clause consisting of one propositional literal. If a set of clauses $\Gamma$ contains this kind of formulas, by convention those formulas are considered to be in $\mathsf{alw}(\Gamma)$.

**Definition 4.2.3.** *For any set of clauses* $\Gamma$

*(a)* $\mathsf{drop}_\Box(\Gamma) = \mathsf{now}(\Gamma) \cup \{N \mid \Box N \in \mathsf{alw}(\Gamma)\}$.

*(b)* $\mathsf{BTL}(\Gamma) = \{T \mid T \vee N \in \mathsf{drop}_\Box(\Gamma)\}$.

*(c)* $\mathsf{unnext}(\Gamma) = \mathsf{alw}(\Gamma) \cup \{N \mid \Box^b(\circ N) \in \Gamma\}$.

The set $\mathsf{drop}_\Box(\Gamma)$ is formed by all the now-clauses in $\Gamma$ together with the inner now-clause of all the always-clauses in $\Gamma$.

$\mathsf{BTL}(\Gamma)$ is the set of all the (basic) temporal literals that occur in $\Gamma$. Hence, $\mathsf{BTL}(\Gamma)$ is a subset of $\mathsf{Lits}(\Gamma)$. It is worth noting that any literal in $\mathsf{Lits}(\Gamma)$ that does not belong to $\mathsf{BTL}(\Gamma)$ is either a propositional literal $P$ or a literal of the form $\circ L$, according to the grammar at the beginning of this subsection.

The set $\mathsf{unnext}(\Gamma)$ consists of all the clauses that should be satisfied at the next state of a state that satisfies $\Gamma$. This definition of $\mathsf{unnext}$ is an adaptation to clauses of the operator unnext presented in Definition 3.3.4. Note also that $\mathsf{unnext}$ implicitly uses the equivalence between $\Box N$ and $\{N, \Box\circ N\}$.

A formula is in *conjunctive normal form* whenever it is a conjunction of clauses. For simplicity, we identify a set of clauses with the conjunction of the clauses in it. Concretely, we identify any formula in conjunctive normal form

$$N_1 \wedge N_2 \wedge \ldots \wedge N_r \wedge \Box N_{r+1} \wedge \ldots \wedge \Box N_k$$

with the set of clauses

$$\{N_1, N_2, \ldots, N_r, \Box N_{r+1}, \ldots, \Box N_k\}$$

where each $N_i$ is a now-clause, $k \geq 1$ and $r \in \{0, \ldots, k\}$.

### 4.2.2 Transforming Formulas into CNF

In this subsection we present a transformation $\mathsf{CNF}$ which maps any formula $\varphi$ to its *conjunctive normal form* $\mathsf{CNF}(\varphi)$. First, we show that any formula $\varphi$ can be transformed into another formula $\mathsf{NNF}(\varphi)$, called the *negation normal form* of $\varphi$, such that every connective $\neg$ is in front of a proposition. Second, we introduce an intermediate notion of normal form, called *distributed normal form*, denoted $\mathsf{DtNF}(\varphi)$ for input formula $\varphi$. The transformations $\mathsf{NNF}$ and $\mathsf{DtNF}$ preserve logical equivalence. Finally we present the transformation of any formula to its conjunctive normal form. The formulas $\varphi$ and $\mathsf{CNF}(\varphi)$ are equisatisfiable (Definition 2.2.2) although, in general, they are not logically equivalent.

**Proposition 4.2.4.** *For any formula $\varphi$ there exists a logically equivalent formula $\mathsf{NNF}(\varphi)$ such that $\chi \in \mathsf{Prop}$ for every subformula of $\mathsf{NNF}(\varphi)$ of the form $\neg\chi$.*

*Proof.* $\mathsf{NNF}(\varphi)$ is obtained by repeatedly applying to any subformula of $\varphi$ the following reduction rules until no one can be applied

$$\begin{array}{ll}
\neg\neg\psi \overset{\mathsf{nnf}}{\longmapsto} \psi & \neg(\psi_1 \vee \psi_2) \overset{\mathsf{nnf}}{\longmapsto} \neg\psi_1 \wedge \neg\psi_2 \\
\neg\circ\psi \overset{\mathsf{nnf}}{\longmapsto} \circ\neg\psi & \neg(\psi_1 \wedge \psi_2) \overset{\mathsf{nnf}}{\longmapsto} \neg\psi_1 \vee \neg\psi_2 \\
\neg\Diamond\psi \overset{\mathsf{nnf}}{\longmapsto} \Box\neg\psi & \neg(\psi_1 \,\mathcal{U}\, \psi_2) \overset{\mathsf{nnf}}{\longmapsto} \neg\psi_1 \,\mathcal{R}\, \neg\psi_2 \\
\neg\Box\psi \overset{\mathsf{nnf}}{\longmapsto} \Diamond\neg\psi & \neg(\psi_1 \,\mathcal{R}\, \psi_2) \overset{\mathsf{nnf}}{\longmapsto} \neg\psi_1 \,\mathcal{U}\, \neg\psi_2
\end{array}$$

It is routine to see that the relation $\overset{\mathsf{nnf}}{\longmapsto}$ (defined above) preserves logical equivalence and the process of repeatedly applying the transformation $\overset{\mathsf{nnf}}{\longmapsto}$ stops after a finite number of steps. Therefore, $\varphi$ and $\mathsf{NNF}(\varphi)$ are logically equivalent. ∎

Now, in the distributed normal form, every connective $\neg$ is in front of a propositional variable, every connective $\vee$ is distributed over $\wedge$, temporal connectives that are distributive over $\vee$ and $\wedge$ are distributed, for formulas of the form $\varphi\,\mathcal{U}\,(\delta\,\mathcal{U}\,\psi)$ and of the form $\varphi\,\mathcal{R}\,(\delta\,\mathcal{R}\,\psi)$ the subformulas $\varphi$ and $\delta$ are different and non-empty sequences of the form $\Diamond\ldots\Diamond$ and of the form $\Box\ldots\Box$ are of length 1.

**Definition 4.2.5.** *A formula is in* distributed normal form *if it has the form* $(\gamma_1^1 \vee \ldots \vee \gamma_1^{k_1}) \wedge \ldots \wedge (\gamma_n^1 \vee \ldots \vee \gamma_n^{k_n})$ *where each* $\gamma_g^j$ *denotes a formula of one of the following forms*

- $\circ^i P$

- $\circ^i(\alpha\,\mathcal{R}\,\beta)$ *for some* $\alpha$ *and* $\beta \neq \alpha\,\mathcal{R}\,\psi$ *for any* $\psi$
- $\circ^i(\beta\,\mathcal{U}\,\alpha)$ *for some* $\beta$ *and* $\alpha \neq \beta\,\mathcal{U}\,\psi$ *for any* $\psi$
- $\circ^i\Box\beta$ *for some* $\beta \neq \Box\psi$ *for any* $\psi$
- $\circ^i\Diamond\alpha$ *for some* $\alpha \neq \Diamond\psi$ *for any* $\psi$

*where* $\alpha$ *and* $\beta$ *denote two special cases of distributed normal form. Concretely,* $\beta$ *stands for a formula of the form* $(\gamma_1^1 \vee \ldots \vee \gamma_1^{k_1})$ *with* $k_1 \geq 1$ *and* $\alpha$ *stands for either a formula* $\gamma_1^1$ *or a formula* $(\gamma_1^1 \vee \ldots \vee \gamma_1^{k_1}) \wedge \ldots \wedge (\gamma_n^1 \vee \ldots \vee \gamma_n^{k_n})$ *with* $n \geq 2$ *and* $k_h \geq 1$ *for every* $h \in \{1, \ldots, n\}$.

Note that if a formula is in distributed normal form then it is also in negation normal form.

**Proposition 4.2.6.** *For any formula* $\varphi$ *there exists a logically equivalent formula* $\mathsf{DtNF}(\varphi)$ *such that* $\mathsf{DtNF}(\varphi)$ *is in distributed normal form.*

*Proof.* First, we transform $\varphi$ into $\mathsf{NNF}(\varphi)$ and then we repeatedly apply to $\mathsf{NNF}(\varphi)$ the following reduction rules

$$(\varphi_1 \wedge \varphi_2) \vee \psi \overset{\mathsf{dtnf}}{\longmapsto} (\varphi_1 \vee \psi) \wedge (\varphi_2 \vee \psi) \qquad \psi \vee (\varphi_1 \wedge \varphi_2) \overset{\mathsf{dtnf}}{\longmapsto} (\psi \vee \varphi_1) \wedge (\psi \vee \varphi_2)$$

$$\circ(\varphi_1 \vee \varphi_2) \overset{\mathsf{dtnf}}{\longmapsto} \circ\varphi_1 \vee \circ\varphi_2 \qquad \circ(\varphi_1 \wedge \varphi_2) \overset{\mathsf{dtnf}}{\longmapsto} \circ\varphi_1 \wedge \circ\varphi_2$$

$$\psi\,\mathcal{U}\,(\varphi_1 \vee \varphi_2) \overset{\mathsf{dtnf}}{\longmapsto} (\psi\,\mathcal{U}\,\varphi_1) \vee (\psi\,\mathcal{U}\,\varphi_2) \qquad \psi\,\mathcal{R}\,(\varphi_1 \wedge \varphi_2) \overset{\mathsf{dtnf}}{\longmapsto} (\psi\,\mathcal{R}\,\varphi_1) \wedge (\psi\,\mathcal{R}\,\varphi_2)$$

$$(\varphi_1 \wedge \varphi_2)\,\mathcal{U}\,\psi \overset{\mathsf{dtnf}}{\longmapsto} (\varphi_1\,\mathcal{U}\,\psi) \wedge (\varphi_2\,\mathcal{U}\,\psi) \qquad (\varphi_1 \vee \varphi_2)\,\mathcal{R}\,\psi \overset{\mathsf{dtnf}}{\longmapsto} (\varphi_1\,\mathcal{R}\,\psi) \vee (\varphi_2\,\mathcal{R}\,\psi)$$

$$\Diamond(\varphi_1 \vee \varphi_2) \overset{\mathsf{dtnf}}{\longmapsto} \Diamond\varphi_1 \vee \Diamond\varphi_2 \qquad \Box(\varphi_1 \wedge \varphi_2) \overset{\mathsf{dtnf}}{\longmapsto} \Box\varphi_1 \wedge \Box\varphi_2$$

$$\psi_1\,\mathcal{U}\,(\psi_1\,\mathcal{U}\,\psi_2) \overset{\mathsf{dtnf}}{\longmapsto} \psi_1\,\mathcal{U}\,\psi_2 \qquad \psi_1\,\mathcal{R}\,(\psi_1\,\mathcal{R}\,\psi_2) \overset{\mathsf{dtnf}}{\longmapsto} \psi_1\,\mathcal{R}\,\psi_2$$

$$\Diamond\Diamond\psi \overset{\mathsf{dtnf}}{\longmapsto} \Diamond\psi \qquad \Box\Box\psi \overset{\mathsf{dtnf}}{\longmapsto} \Box\psi$$

It is routine to see that this reduction always terminates giving a formula in distributed normal form. Additionally, it must be proved that every $\overset{\mathsf{dtnf}}{\longmapsto}$-rule preserves logical equivalence. For that, the only non-trivial $\overset{\mathsf{dtnf}}{\longmapsto}$-rules are the ones for transforming $\psi\,\mathcal{U}\,(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \wedge \varphi_2)\,\mathcal{U}\,\psi$, $\psi\,\mathcal{R}\,(\varphi_1 \wedge \varphi_2)$, and $(\varphi_1 \vee \varphi_2)\,\mathcal{R}\,\psi$. Here, we give the proof details for the first one. The remaining three are similar.
Suppose that $\langle \mathcal{M}, s_j \rangle \models \psi\,\mathcal{U}\,(\varphi_1 \vee \varphi_2)$. Then, there exists $k \geq j$ such that $\langle \mathcal{M}, s_k \rangle \models \varphi_1 \vee \varphi_2$ and $\langle \mathcal{M}, s_i \rangle \models \psi$ for every $i$ such that $j \leq i < k$. Hence, for such $k$, either $\langle \mathcal{M}, s_k \rangle \models \varphi_1$ or $\langle \mathcal{M}, s_k \rangle \models \varphi_2$. In the former case, $\langle \mathcal{M}, s_j \rangle \models \psi\,\mathcal{U}\,\varphi_1$, whereas in the latter $\langle \mathcal{M}, s_j \rangle \models \psi\,\mathcal{U}\,\varphi_2$. Therefore $\langle \mathcal{M}, s_j \rangle \models (\psi\,\mathcal{U}\,\varphi_1) \vee (\psi\,\mathcal{U}\,\varphi_2)$.
Conversely, if $\langle \mathcal{M}, s_j \rangle \models (\psi\,\mathcal{U}\,\varphi_1) \vee (\psi\,\mathcal{U}\,\varphi_2)$, then either $\langle \mathcal{M}, s_j \rangle \models (\psi\,\mathcal{U}\,\varphi_1)$ or $\langle \mathcal{M}, s_j \rangle \models (\psi\,\mathcal{U}\,\varphi_2)$. Hence, there exists $k \geq j$ such that $\langle \mathcal{M}, s_i \rangle \models \psi$ for all $i$ such that $j \leq i < k$ and $\langle \mathcal{M}, s_k \rangle \models \varphi_1$ or $\langle \mathcal{M}, s_k \rangle \models \varphi_2$. Then, $\langle \mathcal{M}, s_k \rangle \models \varphi_1 \vee \varphi_2$ and $\langle \mathcal{M}, s_i \rangle \models \psi$ for every $i$ such that $j \leq i < k$. Therefore, $\langle \mathcal{M}, s_j \rangle \models \psi\,\mathcal{U}\,(\varphi_1 \vee \varphi_2)$. ■

As the following theorem shows, we will use the distributed normal form as a preliminary step for transforming a formula into its conjunctive normal form.

**Theorem 4.2.7.** *For any formula $\varphi$ there exists an equisatisfiable formula* $\mathsf{CNF}(\varphi)$ *such that* $\mathsf{CNF}(\varphi)$ *is in conjunctive normal form.*

*Proof.* First, we transform $\varphi$ into $\mathsf{DtNF}(\varphi)$. Second, we repeatedly apply the following rules until no one can be applied. In the rules bellow $\psi$ is the whole formula (in distributed normal form) and the expressions of the form $\psi[\alpha \Rightarrow \beta]$ denote the formula obtained by simultaneously replacing all the occurrences of the subformula $\alpha$ in $\psi$ by the formula $\beta$, where $\alpha$ is any non-literal subformula of any conjunct of $\psi$ that is not a clause yet.

$$\psi \xmapsto{\mathsf{cnf}} \psi[\circ^i(\varphi_1 \mathcal{U} \varphi_2) \Rightarrow \circ^i(p_1 \mathcal{U} p_2)] \;\wedge\; \mathsf{CNF}(\square(\neg p_1 \vee \varphi_1)) \;\wedge\; \mathsf{CNF}(\square(\neg p_2 \vee \varphi_2))$$

$$\psi \xmapsto{\mathsf{cnf}} \psi[\circ^i(\varphi_1 \mathcal{R} \varphi_2) \Rightarrow \circ^i(p_1 \mathcal{R} p_2)] \;\wedge\; \mathsf{CNF}(\square(\neg p_1 \vee \varphi_1)) \;\wedge\; \mathsf{CNF}(\square(\neg p_2 \vee \varphi_2))$$

$$\psi \xmapsto{\mathsf{cnf}} \psi[\circ^i\square\gamma \Rightarrow \circ^i\square p] \;\wedge\; \mathsf{CNF}(\square(\neg p \vee \gamma))$$

$$\psi \xmapsto{\mathsf{cnf}} \psi[\circ^i\diamond\gamma \Rightarrow \circ^i\diamond p] \;\wedge\; \mathsf{CNF}(\square(\neg p \vee \gamma))$$

$$\psi \xmapsto{\mathsf{cnf}} \psi[\square(\gamma \vee \square\chi) \Rightarrow \square(\gamma \vee \square p)] \;\wedge\; \mathsf{CNF}(\square(\neg p \vee \chi))$$

$$\psi \xmapsto{\mathsf{cnf}} \psi[\square(\square\chi \vee \gamma) \Rightarrow \square(\square p \vee \gamma)] \;\wedge\; \mathsf{CNF}(\square(\neg p \vee \chi))$$

where $p$, $p_1$ and $p_2$ are fresh new propositional variables and the formula $\chi$ is not a propositional literal. Note that the new conjunctions of the form $\mathsf{CNF}(\square(\neg\psi_1 \vee \psi_2))$ serve to define the fresh new symbols $\psi_1$. We will prove that the transformation from $\varphi$ to $\mathsf{CNF}(\varphi)$ stops after a finite number of steps and both formulas are equisatisfiable.

On one hand, each application of a $\xmapsto{\mathsf{cnf}}$-rule reduces the depth of (at least) one non-literal subformula of a formula in DtNF-form. Additionally, the number of fresh new variables is bounded by the number of subformulas. These two facts ensure termination.

On the other hand we prove, by structural induction, that the formulas in both sides of each $\xmapsto{\mathsf{cnf}}$-rule are equisatisfiable. Here we only show the details for the first rule above (the remaining rules are similar or particular cases). Suppose that $\langle \mathcal{M}, s_j\rangle \models \psi$ where $\psi$ is in distributed normal form and $\circ^i(\varphi_1 \mathcal{U} \varphi_2)$ is a non-literal subformula of any conjunct of $\psi$ that is not a clause yet. Then, since $p_1$ and $p_2$ are fresh, $p_1, p_2 \notin V_{\mathcal{M}}(s_k)$ for all $k \geq 0$. Therefore, we define $\mathcal{M}'$ to be the extension of $\mathcal{M}$ such that $p_h \in V_{\mathcal{M}'}(s'_k)$ iff $\langle \mathcal{M}, s_k\rangle \models \varphi_h$ for all $k \geq 0$ and $h \in \{1, 2\}$. As a consequence, for all $k \geq 0$, $\langle \mathcal{M}, s_k\rangle \models \circ^i(\varphi_1 \mathcal{U} \varphi_2)$ iff $\langle \mathcal{M}', s'_k\rangle \models \circ^i(p_1 \mathcal{U} p_2)$ and $\langle \mathcal{M}', s'_k\rangle \models \square(\neg p_1 \vee \varphi_1) \wedge \square(\neg p_2 \vee \varphi_2)$. Hence,

$$\langle \mathcal{M}', s'_k\rangle \models \psi[\circ^i(\varphi_1 \mathcal{U} \varphi_2) \Rightarrow \circ^i(p_1 \mathcal{U} p_2)] \wedge \square(\neg p_1 \vee \varphi_1) \wedge \square(\neg p_2 \vee \varphi_2).$$

By the induction hypothesis, the transformation of $\square(\neg p_1 \vee \varphi_1)$ and $\square(\neg p_2 \vee \varphi_2)$ to conjunctive normal form preserves equisatisfiability.

Conversely, consider any model $\mathcal{M}$ of the right-hand part of the first $\xmapsto{\mathsf{cnf}}$-rule. If $\langle \mathcal{M}, s_0\rangle \not\models \circ^i(p_1 \mathcal{U} p_2)$, then $\langle \mathcal{M}, s_0\rangle$ must satisfy some other disjunct in every conjunct of the formula $\psi$ where $\circ^i(p_1 \mathcal{U} p_2)$ occurs in. Therefore $\mathcal{M}$ is also a model of $\psi$. If $\langle \mathcal{M}, s_0\rangle \models \circ^i(p_1 \mathcal{U} p_2)$, then there exists a $j \geq i$ such that $\langle \mathcal{M}, s_j\rangle \models p_2$ and $\langle \mathcal{M}, s_k\rangle \models p_1$ for all $k$ such that $i \leq k < j$. Additionally, for all $k \geq 0$, $\langle \mathcal{M}, s_k\rangle \models \square(\neg p_h \vee \varphi_h)$ for $h \in \{1, 2\}$. Therefore, $\langle \mathcal{M}, s_j\rangle \models \varphi_2$ and $\langle \mathcal{M}, s_k\rangle \models \varphi_1$ for all $k$ such that $i \leq k < j$. Hence, $\langle \mathcal{M}, s_0\rangle \models \circ^i(\varphi_1 \mathcal{U} \varphi_2)$, which means that $\mathcal{M}$ must be a model of $\psi$. ■

**Example 4.2.8.** *Let us consider the following formula* $\varphi = \neg(p \wedge r \wedge \Box(\neg(p \wedge r) \vee \circ(p \wedge r)))$ *Note that* $\varphi$ *is equivalent to* $\neg\Box(p \wedge r)$ *by means of induction on time. First, we transform* $\varphi$ *into*

$$\mathsf{NNF}(\varphi) = \neg p \vee \neg r \vee \Diamond(p \wedge r \wedge \circ(\neg p \vee \neg r))$$

*Then, its distributed normal form is*

$$\mathsf{DtNF}(\varphi) = \neg p \vee \neg r \vee \Diamond(p \wedge r \wedge (\circ\neg p \vee \circ\neg r))$$

*Finally, the conjunctive (or clausal) normal form of* $\varphi$ *is*

$$\begin{aligned}\mathsf{CNF}(\varphi) &= (\neg p \vee \neg r \vee \Diamond a) \wedge \mathsf{CNF}(\Box(\neg a \vee (p \wedge r \wedge (\circ\neg p \vee \circ\neg r)))) = \\ &= (\neg p \vee \neg r \vee \Diamond a) \wedge \Box(\neg a \vee p) \wedge \Box(\neg a \vee r) \wedge \Box(\neg a \vee \circ\neg p \vee \circ\neg r)\end{aligned}$$

*where a new propositional variable* $a \in \mathsf{Prop}$ *has been introduced and new clauses that define the variable* $a$ *have been added. The formula* $\mathsf{CNF}(\varphi)$ *can also be understood as the set of clauses* $\{(\neg p \vee \neg r \vee \Diamond a), \Box(\neg a \vee p), \Box(\neg a \vee r), \Box(\neg a \vee \circ\neg p \vee \circ\neg r)\}$.

### 4.2.3 Complexity of the Translation

In this subsection we show that the worst case of the translation to CNF is bounded by an exponential on the size of the input formula.

**Definition 4.2.9.** *Given a formula* $\varphi$*, we define the size of* $\varphi$*, namely* $\mathsf{size}(\varphi)$*, as the number of connectives* $\mathsf{cnt}(\varphi)$ *plus the number of propositional variables,* $\mathsf{pv}(\varphi)$ *in* $\varphi$*.*

**Proposition 4.2.10.** *For any formula* $\varphi$*,* $\mathsf{size}(\mathsf{CNF}(\varphi)) \in 2^{\mathcal{O}(\mathsf{size}(\varphi))}$.

*Proof.* The complexity of the first transformation from $\varphi$ to $\mathsf{NNF}(\varphi)$ is linear because the worst case is when the connective $\neg$ appears only once and it occurs as the outermost connective, i.e. $\varphi$ is of the form $\neg\psi$ for some formula $\psi$. In such a case $\neg$ will end up appearing in front of every propositional variable. Hence, $\mathsf{size}(\mathsf{NNF}(\varphi)) = \mathsf{cnt}(\varphi) - 1 + 2 \times \mathsf{pv}(\varphi)$ which is smaller or equal than $2 \times \mathsf{size}(\varphi)$.
In the second transformation to $\mathsf{DtNF}(\varphi)$, each use of the distribution laws can almost double the size of the initial formula. So, we only can ensure that $\mathsf{size}(\mathsf{DtNF}(\varphi)) \leq 2^{\mathsf{size}(\mathsf{NNF}(\varphi))}$ or equivalently that $\mathsf{size}(\mathsf{DtNF}(\varphi)) \in \mathcal{O}(2^{\mathsf{size}(\varphi)})$.
Finally, the last transformation to $\mathsf{CNF}(\varphi)$ has again linear complexity. This is basically because –in the rules of Theorem 4.2.7– each new variable replaces a subformula of a formula $\psi$ that is already in DtNF form.
Summarizing, $\mathsf{size}(\mathsf{CNF}(\varphi)) \in \mathcal{O}(2^{\mathcal{O}(\mathsf{size}(\varphi))}) = 2^{\mathcal{O}(\mathsf{size}(\varphi))}$. ∎

We would like to remark that the exponential blow-up is only due –as in classical cnf– to the distribution laws and it can be prevented using fresh variables as it is made in the so-called *definitional cnf* (see [39]). Therefore, as in classical cnf, for practical purposes, we could use new variables to achieve a transformation to clausal form of linear complexity.

$$(Res)\ \frac{\Box^{b}(L \vee N) \qquad \Box^{b'}(\widetilde{L} \vee N')}{\Box^{b \times b'}(N \vee N')}$$

*Figure 4.1:* The Resolution Rule

$$(Sbm)\ \{\Box^{b}N, \Box^{b}N'\} \longmapsto \{\Box^{b}N'\} \quad \text{if } N' \subseteq N$$

*Figure 4.2:* The Subsumption Rule

## 4.3 The Temporal Resolution Rules

In this section, we present the rules of our temporal resolution system. In addition to a resolution-like rule $(Res)$, the Temporal Resolution System TRS includes a subsumption rule $(Sbm)$ and also the three so-called fixpoint rules –$(\mathcal{R}\,Fix)$, $(\mathcal{U}\,Fix)$ and $(\mathcal{U}\,Set)$– for decomposing temporal literals. The rule $(Sbm)$ is a natural extension of (traditional) clausal subsumption. The rules $(\mathcal{R}\,Fix)$ and $(\mathcal{U}\,Fix)$ are based on the usual inductive definition of the connectives $\mathcal{R}$ and $\mathcal{U}$, respectively, whereas $(\mathcal{U}\,Set)$ is based on a more complex inductive definition of $\mathcal{U}$ (already explained in the previous chapter of this thesis) that is the basis of our approach. Therefore, this section is split into two subsections. The first subsection is devoted to the first four rules which we call *Basic Rules*. The details about the rule $(\mathcal{U}\,Set)$ are explained in the second subsection. The corresponding derived rules for $\Box$ and $\Diamond$ are showed in both subsections. In the sequel, the rules explained in this section are called TRS-rules and the system is called TRS.

### 4.3.1 Basic Rules

Considering that $\Gamma$ is the current set of clauses, the resolution rule $(Res)$ in Figure 4.1 is applied to two clauses (the premises) in $\Gamma$ and obtains a new clause (the resolvent). The rule $(Res)$ is a very natural generalization of classical resolution for always-clauses, and it is written in the usual format of premises and resolvent separated by a horizontal line. $(Res)$ applies to two clauses (the premises) that contain two complementary literals. Both premises can be headed or not by an always connective (depending on superscripts $b$ and $b'$ whose range is $\{0,1\}$). By means of the product $b \times b'$ in the superscript of the resolvent, only when both premises are always-clauses, the resolvent is also an always-clause. In particular, when $N$ and $N'$ are both $\bot$, the resolvent is $\Box^{b \times b'}\bot$, i.e. either $\Box\bot$ or $\bot$. The resolvent is added to $\Gamma$ while the premises remain in $\Gamma$. That is, each application of the rule $(Res)$ adds a clause to the current set of clauses. On the contrary, the remaining TRS-rules replace a set of clauses $\Sigma \subseteq \Gamma$ with another set of clauses, namely $\Psi$. We write them as transformation rules $\Sigma \mapsto \Psi$. The sets $\Sigma$ and $\Psi$ are respectively called the antecedent and the consequent and they are in general equisatisfiable but in some cases logically equivalent. So that, each application of these transformation rules removes the clauses in $\Sigma$ from the current set of clauses and adds the clauses in $\Psi$.

The first transformation rule is the subsumption rule $(Sbm)$ in Figure 4.2, which generalizes

$$\boxed{\begin{array}{llll} (\mathcal{R}\,Fix) & \{\Box^b((P_1\,\mathcal{R}\,P_2)\vee N)\} & \longmapsto & \{\Box^b(P_2\vee N) \\ & & & \;\;\Box^b(P_1\vee\circ(P_1\,\mathcal{R}\,P_2)\vee N)\} \\ & & & \\ (\mathcal{U}\,Fix) & \{\Box^b((P_1\,\mathcal{U}\,P_2)\vee N)\} & \longmapsto & \{\Box^b(P_2\vee P_1\vee N) \\ & & & \;\;\Box^b(P_2\vee\circ(P_1\,\mathcal{U}\,P_2)\vee N)\} \end{array}}$$

*Figure 4.3:* The Fixpoint Rules $(\mathcal{R}\,Fix)$ and $(\mathcal{U}\,Fix)$

$$\boxed{\begin{array}{l} (\Box Fix)\ \{\Box^b(\Box P\vee N)\}\longmapsto\{\Box^b(P\vee N),\ \ \Box^b(\circ\Box P\vee N)\} \\ \\ (\Diamond Fix)\ \{\Box^b(\Diamond P\vee N)\}\longmapsto\{\Box^b(P\vee\circ\Diamond P\vee N)\} \end{array}}$$

*Figure 4.4:* The Fixpoint Rules $(\Box\,Fix)$ and $(\Diamond\,Fix)$

classical subsumption to always-clauses.[2] This rule can be applied to any set that contains a clause of the form $\Box^b N$ and a clause of the form $\Box^b N'$, such that $N'\subseteq N$. The application of the rule $(Sbm)$ eliminates the clause $\Box^b N$ while the clause $\Box^b N'$ remains. Regarding these two clauses in the antecedent, it is said that the clause $\Box^b N$ is subsumed by the clause $\Box^b N'$. Our resolution mechanism requires the rule $(Sbm)$ for completeness. Actually, subsumption is used in Lemma 4.6.13 which allows to prove Theorem 4.6.14.

The fixpoint rules $(\mathcal{R}\,Fix)$ and $(\mathcal{U}\,Fix)$ in Figure 4.3 serve to replace a clause of the form $\Box^b(T\vee N)$ with a logically equivalent set of clauses. The rule $(\mathcal{R}\,Fix)$ splits the temporal literal $P_1\,\mathcal{R}\,P_2$ by using the well-known inductive definition of the connective $\mathcal{R}:\ P_1\,\mathcal{R}\,P_2\equiv P_2\wedge(P_1\vee\circ(P_1\,\mathcal{R}\,P_2))$. Likewise, the rule $(\mathcal{U}\,Fix)$ uses the inductive definition of the connective $\mathcal{U}:\ P_1\,\mathcal{U}\,P_2\equiv P_2\vee(P_1\wedge\circ(P_1\,\mathcal{U}\,P_2))$. In both cases, a simple distribution gives the equivalent set of two clauses that is shown in the consequent of each rule. In order to illustrate this point let us consider the case of the connective $\mathcal{U}$. By the inductive definition of $\mathcal{U}$ and distributivity of $\vee$ over $\wedge$,

$$P_1\,\mathcal{U}\,P_2\equiv P_2\vee(P_1\wedge\circ(P_1\,\mathcal{U}\,P_2))\equiv(P_2\vee P_1)\wedge(P_2\vee\circ(P_1\,\mathcal{U}\,P_2)).$$

Hence, $\Box^b((P_1\,\mathcal{U}\,P_2)\vee N)$ is logically equivalent to the conjunction of the two clauses $\Box^b(P_2\vee P_1\vee N)$ and $\Box^b(P_2\vee\circ(P_1\,\mathcal{U}\,P_2)\vee N)$. So that, the antecedent of the rule $(\mathcal{U}\,Fix)$ is logically equivalent to the conjunction of the two clauses in the consequent.

Since the connectives $\Box$ and $\Diamond$ can be seen as particular cases of $\mathcal{R}$ and $\mathcal{U}$ respectively, the rules in Figure 4.4 constitute the corresponding specializations of the rules in Figure 4.3.

### 4.3.2 The Rule $(\mathcal{U}\,Set)$

The rule $(\mathcal{U}\,Set)$ in Figure 4.5 is an adaptation to the resolution system of the TTM-rule $(\mathcal{U})_2$ presented in Figure 3.1. The construction of the consequent of the rule $(\mathcal{U}\,Set)$ takes into

[2] Note that the same superscript $b$ occurs in both clauses.

$$\begin{array}{lll}
(\mathcal{U}\,Set) & \Phi\ \cup & \{\Box^{b_i}((P_1\,\mathcal{U}\,P_2) \vee N_i) \mid 1 \leq i \leq n\} \\
& \longmapsto \Phi & \cup\ \{P_2 \vee P_1 \vee N_i \mid\ 1 \leq i \leq n\} \\
& & \cup\ \{P_2 \vee \circ(a\,\mathcal{U}\,P_2) \vee N_i\ \mid\ 1 \leq i \leq n\} \\
& & \cup\ \mathsf{CNF}(\mathsf{def}(a, P_1, \Delta)) \\
& & \cup\ \{\Box\,(\circ(P_1\,\mathcal{U}\,P_2) \vee \circ N_i)\ \mid\ b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}$$

where $n \geq 1$

$\Delta = \mathsf{now}(\Phi)$

$a \in \mathsf{Prop}$ is fresh

$\mathsf{def}(a, P_1, \Delta) = \Box\,(\neg a \vee (P_1 \wedge \neg\Delta))$ if $\Delta \neq \emptyset$

$\mathsf{def}(a, P_1, \Delta) = \Box\,\neg a$ if $\Delta = \emptyset$

*Figure 4.5:* The Rule $(\mathcal{U}\,Set)$

account, not only a (non-empty) set whose clauses include a temporal atom $P_1\,\mathcal{U}\,P_2$, but also the remaining clauses. Consequently, the antecedent of the rule $(\mathcal{U}\,Set)$ is

$$\Phi \cup \{\Box^{b_i}((P_1\,\mathcal{U}\,P_2) \vee N_i) \mid 1 \leq i \leq n\}$$

where $n \geq 1$ and $\Phi$ stands for the set consisting of all the remaining clauses in the set to which $(\mathcal{U}\,Set)$ is applied. The antecedent of $(\mathcal{U}\,Set)$ must be interpreted as a partition of the whole set of clauses (on which we are applying temporal resolution) into two sets. The second set $\{\Box^{b_i}((P_1\,\mathcal{U}\,P_2) \vee N_i) \mid 1 \leq i \leq n\}$ in the antecedent is a non-empty set of clauses that contain the same (basic) temporal literal $P_1\,\mathcal{U}\,P_2$. It is worth noting that the literal $P_1\,\mathcal{U}\,P_2$ can also occur in $\Phi$. The opposite restriction is not required for soundness. However, for achieving completeness the rule $(\mathcal{U}\,Set)$ is applied over a partition of the current set of clauses into a set formed by all the clauses that include $P_1\,\mathcal{U}\,P_2$ and the remaining clauses.

**Example 4.3.1.** *Let us apply the rule $(\mathcal{U}\,Set)$ to the eventuality $r\,\mathcal{U}\,s$ in the set of clauses*

$$\{p, \circ q, \Box u, \Box\,((r\,\mathcal{U}\,s) \vee (\circ t))\}.$$

*Then $\Phi = \{p, \circ q, \Box u\}$ and $\Delta = \mathsf{now}(\Phi) = \{p, \circ q\}$, where* now *is the operator on sets of clauses introduced in Definition 4.2.2. Therefore, the consequent of this $(\mathcal{U}\,Set)$ application is*

$$\begin{array}{ll}
\{p, \circ q, \Box u\} & \cup\ \{s \vee r \vee \circ t, s \vee \circ(a\,\mathcal{U}\,s) \vee \circ t\} \\
& \cup\ \{\Box\,(\neg a \vee r), \Box\,(\neg a \vee \neg p \vee \circ\neg q)\} \\
& \cup\ \{\Box\,((\circ(r\,\mathcal{U}\,s)) \vee (\circ\circ t))\}
\end{array}$$

*where $a$ is the fresh variable and $\mathsf{def}(a, r, \Delta) = \{\Box\,(\neg a \vee r), \Box\,(\neg a \vee \neg p \vee \circ\neg q)\}$. Below we justify the construction of $\Delta = \mathsf{now}(\Phi)$ for excluding always-clauses from the definition of the fresh variable $a$. We call $\Delta$ the context. Let us give a clue on context handling through this example. If we used the whole set $\Phi$ instead of $\Delta$ in the definition of $a$, then the second clause in $\mathsf{def}(a, r, \Phi)$ would be $\Box\,(\neg a \vee \neg p \vee \circ\neg q \vee \Diamond\,\neg u)$. However, since $\Box\,u$ is in $\Phi$, the clause $\Box\,u$ also belongs to the consequent. Therefore, the disjunct $\Diamond\,\neg u$ of the above clause, would never be satisfied.*

Next, we explain the intuition behind the rule $(\mathcal{U}\, Set)$ and introduce the definition of *context*.

The crucial idea behind the rule $(\mathcal{U}\, Set)$ (and, hence, behind the TRS resolution system) is based on the equisatisfiability result presented in Proposition 3.3.3 (Section 3.3.2). Here we provide an adaptation of Proposition 3.3.3 to the clausal language.

**Proposition 4.3.2.** *Let $\Delta$ be a set of formulas, $\Sigma_1 = \Delta \cup \{P_2 \vee P_1 \vee \beta, P_2 \vee \circ(P_1 \mathcal{U}\, P_2) \vee \beta\}$ and $\Sigma_2 = \Delta \cup \{P_2 \vee P_1 \vee \beta, P_2 \vee \circ((P_1 \wedge \neg\Delta)\, \mathcal{U}\, P_2) \vee \beta\}$. Then $\Sigma_1$ and $\Sigma_2$ are equisatisfiable.*

*Proof.* Suppose that $\Sigma_1$ has a model $\mathcal{M}$. If $\langle \mathcal{M}, s_0\rangle \models \Delta \cup \{P_2\}$ or $\langle \mathcal{M}, s_0\rangle \models \Delta \cup \{\beta\}$, then $\mathcal{M}$ is also a model of $\Sigma_2$. Otherwise, $\langle \mathcal{M}, s_0\rangle \models \{P_1, \circ(P_1 \mathcal{U}\, P_2)\}$ and $P_2$ should be satisfied in some state $s_j$ with $j \geq 1$ and $P_1$ is true in all the states $s_h$ such that $0 \leq h < j$. Let $k$ be the greatest index in $\{0, \ldots, j-1\}$ such that $\langle \mathcal{M}, s_k\rangle \models \Delta$ and $\Delta$ is not satisfied in the states $s_{k+1}, \ldots, s_{j-1}$ of $\mathcal{M}$. Then, we can construct a model $\mathcal{M}'$ of $\Delta$ by simply deleting the states $s_0, \ldots, s_{k-1}$ in $\mathcal{M}$. As a consequence of the choice of $k$, the PLTL-structure $\mathcal{M}'$ is also a model of $\{P_1, \circ((P_1 \wedge \neg\Delta)\, \mathcal{U}\, P_2)\}$. Hence, $\mathcal{M}' \models \Sigma_2$. Conversely, any model of $\Sigma_2$ is a model of $\Sigma_1$. ∎

Now, we transform the antecedent of $(\mathcal{U}\, Set)$ into its consequent, while preserving equisatisfiability (indeed, logical equivalence is preserved at most steps).
The first transformation step is based on the equivalence $\square\psi \equiv \psi \wedge \square\circ\psi$. Consequently, each clause $\square^{b_i}((P_1 \mathcal{U}\, P_2) \vee N_i)$ such that $b_i = 1$ is split (while clauses with $b_i = 0$ remain unchanged). So that, the set in the antecedent of $(\mathcal{U}\, Set)$:

$$\Psi_0 = \Phi \quad \cup \quad \{\square^{b_i}((P_1 \mathcal{U}\, P_2) \vee N_i) \mid 1 \leq i \leq n\}$$

is equivalent to

$$\begin{aligned}\Psi_1 = \Phi \quad &\cup \quad \{(P_1 \mathcal{U}\, P_2) \vee N_i \mid 1 \leq i \leq n\}\\ &\cup \quad \{\square^{b_i}((P_1 \mathcal{U}\, P_2) \vee N_i) \mid b_i = 1, 1 \leq i \leq n\}\end{aligned}$$

Then, as explained for the rule $(\mathcal{U}\, Fix)$, the set $\Psi_1$ is equivalent to the set

$$\begin{aligned}\Psi_2 = \Phi \quad &\cup \quad \{P_2 \vee P_1 \vee N_i \mid\ 1 \leq i \leq n\}\\ &\cup \quad \underline{\{P_2 \vee \circ(P_1 \mathcal{U}\, P_2) \vee N_i \ \mid\ 1 \leq i \leq n\}}\\ &\cup \quad \{\square^{b_i}(\circ(P_1 \mathcal{U}\, P_2) \vee \circ N_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\end{aligned}$$

Let $\Upsilon$ be the last set in the description of $\Psi_2$, that is

$$\Upsilon = \{\square^{b_i}(\circ(P_1 \mathcal{U}\, P_2) \vee \circ N_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\},$$

we replace the above underlined set (inside $\Psi_2$) with the following set

$$\{P_2 \vee \circ((P_1 \wedge \neg(\Phi \cup \Upsilon))\, \mathcal{U}\, P_2) \vee N_i \ \mid\ 1 \leq i \leq n\} \tag{4.1}$$

Hence, we obtain

$$\begin{aligned}\Psi_3 = \Phi \quad &\cup \quad \{P_2 \vee P_1 \vee N_i \mid\ 1 \leq i \leq n\}\\ &\cup \quad \{P_2 \vee \circ((P_1 \wedge \neg(\Phi \cup \Upsilon))\, \mathcal{U}\, P_2) \vee N_i \ \mid\ 1 \leq i \leq n\}\\ &\cup \quad \{\square^{b_i}(\circ(P_1 \mathcal{U}\, P_2) \vee \circ N_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\end{aligned}$$

By Proposition 4.3.2, the sets $\Psi_2$ and $\Psi_3$ are equisatisfiable. Additionally, any set of the form

$$\{\square\chi_1, \square\chi_2, \ldots, \square\chi_m, \circ((\varphi \wedge (\gamma \vee \neg\square\chi_1 \vee \neg\square\chi_2 \vee \ldots \vee \neg\square\chi_m))\, \mathcal{U}\, \psi)\}$$

is equivalent to the set

$$\{\Box\chi_1, \Box\chi_2, \ldots, \Box\chi_m, \circ((\varphi \wedge \gamma)\,\mathcal{U}\,\psi)\}$$

because if the formulas $\chi_1, \chi_2, \ldots, \chi_m$ are true from now forever, then the truth of the formula

$$\circ((\varphi \wedge (\gamma \vee \neg\Box\chi_1 \vee \neg\Box\chi_2 \vee \ldots \vee \neg\Box\chi_m))\,\mathcal{U}\,\psi)$$

does not depend on the truth of the disjunction $\neg\Box\chi_1 \vee \neg\Box\chi_2 \vee \ldots \vee \neg\Box\chi_m$ which should be false. Consequently, it is not necessary to consider the clauses that belong to $\mathsf{alw}(\Phi) \cup \Upsilon$ (see Definition 4.2.2) in the subset of $\Psi_3$ considered in (4.1) because only clauses in $\mathsf{now}(\Phi)$ are needed. Therefore, we replace the subformula $\neg(\Phi \cup \Upsilon)$ with $\neg\mathsf{now}(\Phi)$ in $\Psi_3$ and we obtain the following (logically equivalent) set

$$\begin{aligned}\Psi_4 = \Phi \;&\cup\; \{P_2 \vee P_1 \vee N_i \mid 1 \leq i \leq n\}\\ &\cup\; \{P_2 \vee \circ((P_1 \wedge \neg\mathsf{now}(\Phi))\,\mathcal{U}\,P_2) \vee N_i \mid 1 \leq i \leq n\}\\ &\cup\; \{\Box^{b_i}(\circ(P_1\,\mathcal{U}\,P_2) \vee \circ N_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\end{aligned}$$

The logical equivalence of $\Psi_3$ and $\Psi_4$ motivates the following notion of context.

**Definition 4.3.3.** *In an application of the rule* $(\mathcal{U}\,Set)$ *(see Figure 4.5) to an antecedent that is partitioned in the two sets* $\Phi$ *and* $\{\Box^{b_i}((P_1\,\mathcal{U}\,P_2) \vee N_i) \mid 1 \leq i \leq n\}$*, we say that* $\Delta = \mathsf{now}(\Phi)$ *is the* context. [3]

Since the above formula $\circ((P_1 \wedge \neg\mathsf{now}(\Phi))\,\mathcal{U}\,P_2)$ is not (in general) a literal, we should transform $\Psi_4$ into clausal form. For that, we substitute the subformula $P_1 \wedge \neg\mathsf{now}(\Phi)$ by the fresh variable $a$ and we add the clauses that define the meaning of $a$. This gives the following set $\Psi_5$ where $\mathsf{def}(a, P_1, \mathsf{now}(\Phi))$ is the result of transforming the formula $\Box(\neg a \vee (P_1 \wedge \neg\mathsf{now}(\Phi)))$ to a set of clauses (whose definition is given in Figure 4.5):

$$\begin{aligned}\Psi_5 = \Phi \;&\cup\; \{P_2 \vee P_1 \vee N_i \mid 1 \leq i \leq n\}\\ &\cup\; \{P_2 \vee \circ(a\,\mathcal{U}\,P_2) \vee N_i \mid 1 \leq i \leq n\}\\ &\cup\; \mathsf{CNF}(\mathsf{def}(a, P_1, \mathsf{now}(\Phi)))\\ &\cup\; \{\Box^{b_i}(\circ(P_1\,\mathcal{U}\,P_2) \vee \circ N_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\end{aligned}$$

Finally, let us check that the sets $\Psi_4$ and $\Psi_5$ are equisatisfiable. On the one hand, since $a$ does not appear in $\Psi_4$, a model $\mathcal{M}'$ of $\Psi_5$ can be built from a model $\mathcal{M}$ of $\Psi_4$ by just defining $V_{\mathcal{M}'}(s'_j)$ as $V_{\mathcal{M}}(s_j) \cup \{a\}$ if $P_1 \wedge \neg\mathsf{now}(\Phi)$ is true in the state $s_j$ of $\mathcal{M}$ and by defining $V_{\mathcal{M}'}(s'_j)$ as $V_{\mathcal{M}}(s_j) \setminus \{a\}$ if $P_1 \wedge \neg\mathsf{now}(\Phi)$ is not true in the state $s_j$ of $\mathcal{M}$. On the other hand, since every model of $\Psi_5$ satisfies the formula $\Box(\neg a \vee (P_1 \wedge \neg\mathsf{now}(\Phi)))$, we can ensure that $P_1 \wedge \neg\mathsf{now}(\Phi)$ is true in any state $s$ of a model of $\Psi_5$ whenever $a$ is true in $s$. Consequently, $\circ((P_1 \wedge \neg\mathsf{now}(\Phi))\,\mathcal{U}\,P_2)$ is true in any state $s$ of a model of $\Psi_5$ whenever $\circ(a\,\mathcal{U}\,P_2)$ is true in $s$. Therefore, every model of $\Psi_5$ is also a model of $\Psi_4$.

At first sight it could seem that the definition of the fresh variable $a$ should be given by the cnf form of the formula $\Box(\neg a \vee (P_1 \wedge \neg\mathsf{now}(\Phi))) \wedge \Box(a \vee \neg(P_1 \wedge \neg\mathsf{now}(\Phi)))$. However, as can be seen in the above reasoning, the clauses that correspond to the formula $\Box(a \vee \neg(P_1 \wedge \neg\mathsf{now}(\Phi)))$ are not needed for equisatisfiability. Therefore, we do not add the clauses that correspond to $\Box(a \vee \neg(P_1 \wedge \neg\mathsf{now}(\Phi)))$.
To summarize, the initial set $\Psi_0$ –which is the antecedent of the rule $(\mathcal{U}\,Set)$– and the last set $\Psi_5$ –which is the consequent of the rule $(\mathcal{U}\,Set)$– are equisatisfiable.

[3] The operator now is introduced in Definition 4.2.2.

$$
\begin{array}{lll}
(\diamond Set) & \Phi \ \cup & \{\Box^{b_i}(\diamond P \vee N_i) \mid 1 \leq i \leq n\} \\
& \longmapsto \Phi & \cup \{P \vee \circ(a \,\mathcal{U}\, P) \vee N_i \ \mid \ 1 \leq i \leq n\} \\
& & \cup \ \mathsf{CNF}(\mathsf{def}(a, \Delta)) \\
& & \cup \{\Box(\circ\diamond P \vee \circ N_i) \ \mid \ b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}
$$

where $n \geq 1$
$\Delta = \mathsf{now}(\Phi)$
$a \in \mathsf{Prop}$ is fresh
$\mathsf{def}(a, \Delta) = \Box(\neg a \vee \neg\Delta)$ if $\Delta \neq \emptyset$
$\mathsf{def}(a, \Delta) = \Box\neg a$ if $\Delta = \emptyset$

*Figure 4.6:* The Rule $(\diamond Set)$

The correctness of the rule $(\mathcal{U}\, Set)$ is shown in detail in the proof of Proposition 4.5.2.

The rule $(\mathcal{U}\, Set)$ leads to a complete resolution method that does not require invariant generation. This is mainly due to the above explained management of the so-called contexts by means of the rule $(\mathcal{U}\, Set)$. An adaptation to clauses of the strategy followed in the systematic tableau algorithm for TTM (Figure 3.9), prevents from postponing indefinitely the satisfaction of $P_1 \mathcal{U} P_2$. Example 4.4.4 in Section 4.4 illustrates how contexts are handled to cause inconsistency whenever the fulfillment of an eventuality could be infinitely delayed. There is a finite number of possible different contexts and the repetition of a previous context, while postponing an eventuality, also causes inconsistency. Therefore, there is a clear strategy to achieve termination and completeness.

The rule $(\diamond Set)$ in Figure 4.6 is the specialization of $(\mathcal{U}\, Set)$ that corresponds to the equivalence of $\diamond P \equiv \widetilde{P} \mathcal{U} P$. Consequently, along the rest of the chapter, the rule $(\diamond Set)$ is treated as a derived rule, in the sense that most technical details are given only for the general rule $(\mathcal{U}\, Set)$.

## 4.4 Temporal Resolution Derivations

A classical resolution derivation for a set of propositional clauses $\Gamma$ is a sequence of sets of clauses

$$\Gamma_0 \mapsto \Gamma_1 \mapsto \ldots \mapsto \Gamma_k$$

where $\Gamma = \Gamma_0$ and each $\Gamma_{i+1}$ is obtained from $\Gamma_i$ by means of a resolution-step that consists in applying the (classical) resolution rule. The sequence ends when either $\Gamma_k$ contains the empty clause $\bot$ or every application of the resolution rule on formulas in $\Gamma_k$ yields a formula that is already in $\Gamma_k$. For classical propositional logic, resolution is sound, refutationally complete and, even, complete. Soundness and refutational completeness mean that the method obtains a set $\Gamma_k$ that contains $\bot$ for some $k \in \mathbb{N}$ if and only if $\Gamma$ is unsatisfiable. Moreover, in classical propositional resolution the sequence obtained is always finite (if the pairs of clauses for applying the resolution rule are selected fairly) and consequently classical propositional resolution is also complete and serves as a decision procedure.

In this section we first extend the classical notion of derivation –to the temporal case of PLTL– introducing TRS-derivations. We also provide some sample TRS-derivations. The notion of TRS-derivation is the basis of the sound, refutationally complete, and complete resolution mechanism that is presented in this chapter. In the second subsection we prove technical results on the relationship between TRS-resolution and classical (propositional) resolution.

### 4.4.1 TRS-*Derivations and Examples*

Our notion of derivation explicitly simulates the transition from one state to the next one, in the sense that whenever in the current set of clauses no more resolution resolvents can be added, then we use the operator unnext (see Definition 4.2.3) to get the clauses that must be satisfied in the state that follows (is next to) the current one. Inside each state, the TRS-rules are applied, hence the so-called local derivations are (roughly speaking) an extension of classical derivations.

**Definition 4.4.1.** *A* TRS-derivation *for a set of clauses* $\Gamma$ *is a sequence*

$$\mathcal{D} = \Gamma_0^0 \mapsto \Gamma_0^1 \mapsto \ldots \mapsto \Gamma_0^{h_0} \Mapsto \Gamma_1^0 \mapsto \Gamma_1^1 \mapsto \ldots \mapsto \Gamma_1^{h_1} \Mapsto \ldots \Mapsto \Gamma_i^0 \mapsto \Gamma_i^1 \mapsto \ldots$$

*where*

*(a)* $\Gamma_0^0 = \Gamma$

*(b)* $\mapsto$ *represents the application of a* TRS*-rule*

*(c)* $\Mapsto$ *represents the application of the operator* unnext.

*If any set* $\Gamma_i^j$ *in* $\mathcal{D}$ *contains* $\Box^b\bot$*, then* $\mathcal{D}$ *is called a* refutation *for* $\Gamma$*. We say that a* TRS*-derivation is a* local derivation *if it does not contain any application of the operator* unnext*. A local derivation is called a* local refutation *if it is a refutation.*

Note that we use two different symbols ($\mapsto$ and $\Mapsto$) to highlight the difference between the application of a TRS-rule and the application of the operator unnext. The former applications produce sets $\Gamma_i^{j+1}$ from $\Gamma_i^j$ and are called TRS*-steps*. The latter applications yield $\Gamma_{i+1}^0$ from $\Gamma_i^{h_i}$ and are called *unnext-steps*.

In the sequel we only use the prefix TRS- whenever confusion might result, otherwise we simply say derivation.

Now we give four examples of refutations. For readability, the derivations are represented as vertical sequences of rule applications with the name of the applied rule at the right-hand side of each step. In addition, the clauses to which each rule affects have been underlined. However, we do not underline any formula in the applications of the operator unnext. The first example shows that in some cases, even if temporal literals are involved, the refutation is achieved using only the resolution rule $(Res)$ and the operator unnext. The second example illustrates that sometimes the rule $(\mathcal{U}\,Set)$ is not necessary and the rule $(\mathcal{U}\,Fix)$ is enough. The third example shows how contexts are handled to cause inconsistency whenever the fulfillment of an eventuality could be infinitely delayed. Finally, in the fourth example, the rule $(\mathcal{U}\,Set)$ is applied to a proper subset of the set of clauses that contain the literal $p\,\mathcal{U}\,q$. In general, it can be applied to any non-empty subset.

**Example 4.4.2.** *In Figure 4.7 a* TRS*-refutation for the unsatisfiable set of clauses*

$$\boxed{\begin{array}{lll}
\Gamma_0^0 = & \{\Box(r \vee \Diamond p), \Box\circ\neg r, \circ\Box\neg p, \Box(\circ r \vee \neg q \vee \Diamond p), p \vee q, \neg q\} & (\mathsf{unnext}) \\
\Gamma_1^0 = & \{\underline{\Box(r \vee \Diamond p)}, \Box\circ\neg r, \underline{\neg r}, \Box\neg p, \Box(\circ r \vee \neg q \vee \Diamond p)\} & (Res) \\
\Gamma_1^1 = & \{\Box(r \vee \Diamond p), \Box\circ\neg r, \neg r, \underline{\Box\neg p}, \Box(\circ r \vee \neg q \vee \Diamond p), \underline{\Diamond p}\} & (Res) \\
\Gamma_1^2 = & \{\Box(r \vee \Diamond p), \Diamond p, \Box\circ\neg r, \neg r, \Box\neg p, \Box(\circ r \vee \neg q \vee \Diamond p), \bot\} &
\end{array}}$$

*Figure 4.7:* TRS-refutation for the set of clauses $\{\Box(r \vee \Diamond p), \Box\circ\neg r, \circ\Box\neg p, \Box(\circ r \vee \neg q \vee \Diamond p), p \vee q, \neg q\}$

$$\boxed{\begin{array}{lll}
\Gamma_0^0 = & \{\Box\neg p, \underline{\Box(r\,\mathcal{U}\,p)}, (\neg r)\,\mathcal{U}\,p\} & (\mathcal{U}\,Fix) \\
\Gamma_0^1 = & \{\Box\neg p, \underline{(\neg r)\,\mathcal{U}\,p}, \Box(p \vee r), \Box(p \vee \circ(r\,\mathcal{U}\,p))\} & (\mathcal{U}\,Fix) \\
\Gamma_0^2 = & \{\underline{\Box\neg p}, \underline{\Box(p \vee r)}, \Box(p \vee \circ(r\,\mathcal{U}\,p)), p \vee \neg r, p \vee \circ((\neg r)\,\mathcal{U}\,p)\} & (Res) \\
\Gamma_0^3 = & \{\underline{\Box\neg p}, \Box(p \vee r), \Box(p \vee \circ(r\,\mathcal{U}\,p)), \underline{p \vee \neg r}, p \vee \circ((\neg r)\,\mathcal{U}\,p), \Box r\} & (Res) \\
\Gamma_0^4 = & \{\Box\neg p, \Box(p \vee r), \Box(p \vee \circ(r\,\mathcal{U}\,p)), p \vee \neg r, p \vee \circ((\neg r)\,\mathcal{U}\,p), \underline{\Box r}, \underline{\neg r}\} & (Res) \\
\Gamma_0^5 = & \{\Box\neg p, \Box(p \vee r), \Box(p \vee \circ(r\,\mathcal{U}\,p)), p \vee \neg r, p \vee \circ((\neg r)\,\mathcal{U}\,p), \Box r, \neg r, \bot\} &
\end{array}}$$

*Figure 4.8:* TRS-refutation for the set of clauses $\{\Box\neg p, \Box(r\,\mathcal{U}\,p), (\neg r)\,\mathcal{U}\,p\}$

$$\{\Box(r \vee \Diamond p), \Box\circ\neg r, \circ\Box\neg p, \Box(\circ r \vee \neg q \vee \Diamond p), p \vee q, \neg q\}$$

*is provided. It is worth remarking that in the* TRS*-step that yields $\Gamma_1^2$ from $\Gamma_1^1$ the formula $\Box\neg p$ is treated as a now-clause formed by a temporal literal. Although (basic) temporal literals are involved in the derivation process, the rules for decomposing temporal literals are not needed.*

**Example 4.4.3.** *In Figure 4.8 a* TRS*-refutation for the unsatisfiable set of clauses*

$$\{\Box\neg p, \Box(r\,\mathcal{U}\,p), (\neg r)\,\mathcal{U}\,p\}$$

*is showed. In this example the formulas $\Box\neg p$ and $\Box r$ are treated as always-clauses formed by one propositional literal. Although literals that contain the connective $\mathcal{U}$ are involved, the refutation is obtained without using the rule $(\mathcal{U}\,Set)$. The rule $(\mathcal{U}\,Fix)$ is enough in this case.*

**Example 4.4.4.** *Let $\Gamma_0^0 = \{\Box(\neg p \vee \circ p), p, r\,\mathcal{U}\,\neg p\}$. Then, by applying $(\mathcal{U}\,Set)$ to $r\,\mathcal{U}\,\neg p$ in $\Gamma_0^0$ where $\Phi = \{\Box(\neg p \vee \circ p), p\}$ and $\Delta = \{p\}$, we obtain*

$$\Gamma_0^1 = \{\Box(\neg p \vee \circ p), p, \neg p \vee r, \neg p \vee \circ(a\,\mathcal{U}\,\neg p), \Box(\neg a \vee \neg p), \Box(\neg a \vee r)\}$$

*where $a$ is the fresh variable whose meaning is defined to be $r \wedge \neg p$ by the last two clauses. Note that $\neg p$ is $\neg\Delta$. Then, by four applications of the rule $(Res)$ that respectively resolve the singleton clause $p$ with the four occurrences of $\neg p$, we obtain*

$$\Gamma_0^5 = \{\Box(\neg p \vee \circ p), \circ p, r, p, \neg p \vee r, \neg p \vee \circ(a\,\mathcal{U}\,\neg p), \circ(a\,\mathcal{U}\,\neg p), \neg a, \Box(\neg a \vee \neg p), \Box(\neg a \vee r)\}.$$

$$\begin{array}{lll}
\Gamma_0^0 = & \{\Box((p\,\mathcal{U}\,q)\vee r), \underline{\Box((p\,\mathcal{U}\,q)\vee \Diamond s)}, \Box\neg q, \underline{\Box\neg s}\} & (Res) \\
\Gamma_0^1 = & \{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee \Diamond s), \Box\neg q, \Box\neg s, \underline{(p\,\mathcal{U}\,q)}\} & (\mathcal{U}\,Set) \\
\Gamma_0^2 = & \{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee \Diamond s), \underline{\Box\neg q}, \Box\neg s, q\vee p, & (Res) \\
& \underline{q\vee\circ(a\,\mathcal{U}\,q)}, \Box\neg a\} & \\
\Gamma_0^3 = & \{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee \Diamond s), \Box\neg q, \Box\neg s, q\vee p, & (\mathsf{unnext}) \\
& q\vee\circ(a\,\mathcal{U}\,q), \Box\neg a, \circ(a\,\mathcal{U}\,q)\} & \\
\Gamma_1^0 = & \{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee \Diamond s), \Box\neg q, \Box\neg s, \Box\neg a, \underline{a\,\mathcal{U}\,q}\} & (\mathcal{U}\,Set) \\
\Gamma_1^1 = & \{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee \Diamond s), \Box\neg q, \Box\neg s, \underline{\Box\neg a}, \underline{q\vee a}, & (Res) \\
& q\vee\circ(b\,\mathcal{U}\,q), \Box\neg b\} & \\
\Gamma_1^2 = & \{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee \Diamond s), \underline{\Box\neg q}, \Box\neg s, \Box\neg a, q\vee a, & (Res) \\
& q\vee\circ(b\,\mathcal{U}\,q), \Box\neg b, \underline{q}\} & \\
\Gamma_1^3 = & \{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee \Diamond s), \Box\neg q, \Box\neg s, \Box\neg a, q\vee a, & \\
& q\vee\circ(b\,\mathcal{U}\,q), \Box\neg b, q, \bot\} &
\end{array}$$

*Figure 4.9:* TRS-refutation for the set of clauses $\{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee\Diamond s), \Box\neg q, \Box\neg s\}$

*Now, the operator* unnext *produces*

$$\Gamma_1^0 = \{\Box(\neg p\vee\circ p), p, a\,\mathcal{U}\neg p, \Box(\neg a\vee\neg p), \Box(\neg a\vee r)\}.$$

*Hence, the application of* $(\mathcal{U}\,Set)$ *to* $a\,\mathcal{U}\neg p$ *in* $\Gamma_1^0$ *where* $\Phi = \{\Box(\neg p\vee\circ p), p, \Box(\neg a\vee\neg p), \Box(\neg a\vee r)\}$ *and* $\Delta = \{p\}$ *yields*

$$\Gamma_1^1 = \{\Box(\neg p\vee\circ p), p, \neg p\vee a, \neg p\vee\circ(b\,\mathcal{U}\neg p), \Box(\neg b\vee\neg p), \Box(\neg b\vee a), \Box(\neg a\vee\neg p), \Box(\neg a\vee r)\}$$

*where the fresh variable* $b$ *is defined as* $a\wedge\neg p$ *by the clauses* $\Box(\neg b\vee\neg p), \Box(\neg b\vee a)$*. Then, the application of* $(Res)$ *to* $p$ *and* $\neg p\vee a$ *yields* $a$*. Finally, the resolution of* $p$ *and* $\Box(\neg a\vee\neg p)$ *yields* $\neg a$*. Hence, the empty clause is immediately obtained from* $a$ *and* $\neg a$*.*
*Roughly speaking,* $a$ *holds whenever the satisfaction of* $\neg p$ *(or equivalently the fullfilment of* $r\,\mathcal{U}\neg p$*) is postponed. However,* $a$ *means* $r\wedge\neg p$*, where* $\neg p$ *is the negated context. So that, the part of the definition of* $a$ *given by the clause* $\Box(\neg a\vee\neg p)$ *allows the inference of* $\neg a$*, which leads to the inconsistency.*

**Example 4.4.5.** *In Figure 4.9 we show a* TRS*-refutation for the unsatisfiable set of clauses* $\{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee\Diamond s), \Box\neg q, \Box\neg s\}$*. Note that the formula* $\Box\neg s$ *is treated as a literal in* $\Gamma_0^0$ *and as an always-clause in* $\Gamma_0^1$*. Besides, it is worth noting that in* $\Gamma_0^1$ *there are three occurrences of* $p\,\mathcal{U}\,q$*, but the rule* $(\mathcal{U}\,Set)$ *is applied by considering the set* $\Phi$ *to be formed by the first four clauses, i.e.,* $\Phi = \{\Box((p\,\mathcal{U}\,q)\vee r), \Box((p\,\mathcal{U}\,q)\vee\Diamond s), \Box\neg q, \Box\neg s\}$*. So that, in this case the set* $\Phi$ *includes clauses that contain the literal* $p\,\mathcal{U}\,q$*.*

### 4.4.2 Relating TRS-Resolution to Classical Resolution

In this subsection we define the notion of linear local derivation and, based on it, we establish a relation between TRS-resolution and classical resolution that enables us to use well-known results from classical propositional logic.

**Definition 4.4.6.** *A set of clauses $\Gamma$ is* closed *with respect to* TRS*-rules (shortly,* TRS*-closed) iff it satisfies the following three conditions:*

*(a)* $\mathsf{BTL}(\Gamma) = \emptyset$ *(i.e. any literal in $\Gamma$ is either propositional ($p$ or $\neg p$) or starts by $\circ$)*[4]

*(b) The subsumption rule $(Sbm)$ cannot be applied to $\Gamma$*

*(c) Every clause obtained from $\Gamma$ by application of the resolution rule $(Res)$ is already in $\Gamma$ or it is subsumed by some clause in $\Gamma$.*

**Definition 4.4.7.** *Let $\Gamma$ be a set of clauses, we denote by $\Gamma^*$ any set such that there exists a local derivation $\Gamma \mapsto \ldots \mapsto \Gamma^*$ and either $\Box^b\bot \in \Gamma^*$ or $\Gamma^*$ is* TRS*-closed.*

Note that, in general, given a set of clauses $\Gamma$, a local derivation that yields a set $\Gamma^*$ that either contains the clause $\Box^b\bot$ or is TRS-closed, may include some applications of the rules $(\mathcal{U}\,Set)$ and $(\diamond\,Set)$.

**Definition 4.4.8.** *Let $\Gamma$ be a set of clauses, the non-deterministic operation that yields $\Gamma^*$ from $\Gamma$ without any application of the rules $(\mathcal{U}\,Set)$ and $(\diamond\,Set)$ is denoted by* fix_close.

In the algorithm presented in Figure 4.10 (Section 4.6) we use the procedure fix_close that implements the operation fix_close during the construction of a derivation.

**Definition 4.4.9.** *A set of clauses $\Gamma$ is* locally inconsistent *iff there exists a local refutation for $\Gamma$. Otherwise it is* locally consistent.

**Proposition 4.4.10.** *For any* TRS*-closed set of clauses $\Gamma$, if $\Box^b\bot \notin \Gamma$ then $\Gamma$ is locally consistent.*

*Proof.* If $\Gamma$ is TRS-closed, every clause that can be obtained by means of the rule $(Res)$ is already in $\Gamma$ or is subsumed by some other clause in $\Gamma$. If $\Box^b\bot$ is not in $\Gamma$ then there is no way to obtain it by means of a local derivation. ■

The following notion is an adaptation of the concept of *linear resolution based on a clause* (see e.g. Section 2.6 in [115]).

**Definition 4.4.11.** *A local derivation $\mathcal{D}$ for $\Gamma$ is* linear *with respect to a clause $C \in \Gamma$ iff it satisfies the following three conditions*

*(a) Every* TRS*-step in $\mathcal{D}$ is an application of the rule $(Res)$*

*(b) $C$ is one of the premises for $(Res)$ in the first* TRS*-step*

*(c) For every* TRS*-step in $\mathcal{D}$, with the exception of the first one, one of the premises is the resolvent obtained in the previous* TRS*-step.*

[4] see Subsection 4.2.1.

Next, we formulate a useful relationship between TRS-resolution and classical propositional resolution.

**Definition 4.4.12.** *Let* $\Gamma$ *be a set of clauses,* $\mathsf{prop}(\Gamma)$ *is the set that results from* $\mathsf{drop}_\Box(\Gamma)$ *by replacing all the occurrences of each non-propositional literal* $L \in \mathsf{Lits}(\mathsf{drop}_\Box(\Gamma))$ *with a fresh propositional literal in a coherent way, in the sense that complementary literals are replaced with complementary propositional literals.*

**Proposition 4.4.13.** *Let* $\Gamma$ *be a set of clauses such that* $\mathsf{BTL}(\Gamma) = \emptyset$*.*

*(i)* $\mathsf{drop}_\Box(\Gamma)$ *is locally inconsistent iff* $\mathsf{prop}(\Gamma)$ *is inconsistent (in classical logic).*

*(ii)* $\Gamma$ *is locally inconsistent iff* $\mathsf{drop}_\Box(\Gamma)$ *is locally inconsistent.*

*Proof. (i)* For the left to right implication, since $\mathsf{BTL}(\Gamma) = \emptyset$, if $\mathsf{drop}_\Box(\Gamma)$ is locally inconsistent then there exists a local refutation for $\mathsf{drop}_\Box(\Gamma)$ where every TRS-step is an application of the rule $(Res)$ or the rule $(Sbm)$. Hence, we can trivially build a classical refutation for $\mathsf{prop}(\Gamma)$ with the same number of steps and using classical resolution and subsumption instead of $(Res)$ and $(Sbm)$, respectively.
Conversely, if $\mathsf{prop}(\Gamma)$ is inconsistent then by completeness of classical propositional resolution there exists a refutation for $\mathsf{prop}(\Gamma)$ where only the classical resolution rule is used. Then, it is easy to obtain a local refutation for $\mathsf{drop}_\Box(\Gamma)$ applying the resolution rule $(Res)$ to the corresponding clauses.

*(ii)* Since $\mathsf{BTL}(\Gamma) = \emptyset$, if $\Gamma$ is locally inconsistent then there exists a local refutation $\mathcal{D}$ for $\Gamma$ where every TRS-step is an application of the rule $(Res)$ or the rule $(Sbm)$. From $\mathcal{D}$ we can build a local refutation $\mathcal{D}'$ for $\Gamma$ where every TRS-step is an application of the rule $(Res)$. It suffices to remove from $\mathcal{D}$ the TRS-steps in which the rule $(Sbm)$ is applied and to keep (or add) the clauses subsumed in $\mathcal{D}$ by the applications of the rule $(Sbm)$. From $\mathcal{D}'$ we can obtain a local refutation for $\mathsf{drop}_\Box(\Gamma)$ in a trivial manner, by using a clause $N$ whenever the original derivation $\mathcal{D}'$ uses the corresponding $\Box N$.
If $\mathsf{drop}_\Box(\Gamma)$ is locally inconsistent then, by (i) and the completeness of classical propositional resolution, there exists a refutation $\mathcal{D}$ for $\mathsf{prop}(\Gamma)$ where every TRS-step is an application of the classical resolution rule. From $\mathcal{D}$, it is straightforward to obtain a local refutation $\mathcal{D}'$ for $\mathsf{drop}_\Box(\Gamma)$ where every TRS-step is an application of the rule $(Res)$. This local refutation is trivially convertible into a local refutation for $\Gamma$, by using the clause $\Box N \in \Gamma$ instead of $N \in \mathsf{drop}_\Box(\Gamma)$ whenever $N \notin \Gamma$. ∎

Next, we provide a basic result that is used in Section 4.7 for proving completeness. This result is an adaptation of the completeness of classical linear resolution (see Section 2.6 in [115]) that states

> Given a consistent set of propositional clauses $\Phi$, if for a propositional clause $\beta \notin \Phi$ the set $\Phi \cup \{\beta\}$ is inconsistent then there exists a refutation for $\Phi \cup \{\beta\}$ that is linear with respect to the clause $\beta$.

**Proposition 4.4.14.** *Let* $\Gamma$ *be a locally consistent set of clauses such that* $\mathsf{BTL}(\Gamma) = \emptyset$ *and let* $C$ *be a clause that is not in* $\Gamma$ *such that* $\mathsf{BTL}(\{C\}) = \emptyset$*. If* $\Gamma \cup \{C\}$ *is locally inconsistent then there exists a local refutation for* $\Gamma \cup \{C\}$ *that is linear with respect to the clause* $C$*.*

*Proof.* If $\Gamma \cup \{C\}$ is locally inconsistent, by Proposition 4.4.13 the set $\mathsf{prop}(\Gamma \cup \{C\})$ is inconsistent and, by completeness of classical linear resolution (see above), there exists a refutation $\mathcal{D}'$ for $\mathsf{prop}(\Gamma \cup \{C\})$ that is linear with respect to the clause $C' \in \mathsf{prop}(\Gamma \cup \{C\})$ that corresponds to the clause $C$. From $\mathcal{D}'$, it is trivial to build a local refutation $\mathcal{D}$ for $\Gamma \cup \{C\}$ that is linear with respect to $C$. ■

## 4.5 Soundness

A resolution system is *sound* if, whenever a refutation exists for a set of clauses $\Gamma$, then $\Gamma$ is unsatisfiable. The soundness of a system can be guaranteed rule by rule, where a rule is sound whenever it preserves the satisfiability. Often some rules preserve stronger properties than satisfiability. In this section, we analyze each rule from the point of view of soundness and stronger properties and prove that the resolution system TRS is sound.

**Proposition 4.5.1.** *The Basic Rules of Subsection 4.3.1 are sound. Moreover, every application of these rules yields a new set of clauses that is logically equivalent to the initial set.*

*Proof.* When $(Res)$ is applied to two clauses (the premises) $\Box^b(L \vee N)$ and $\Box^{b'}(\widetilde{L} \vee N')$ in $\Gamma$, the resolvent $\Box^{b \times b'}(N \vee N')$ is a logical consequence of $\{\Box^b(L \vee N), \Box^{b'}(\widetilde{L} \vee N')\}$ and, consequently, the new set of clauses $\Gamma' = \Gamma \cup \{\Box^{b \times b'}(N \vee N')\}$ is logically equivalent to the set of clauses $\Gamma$.

For soundness of $(Sbm)$, suppose that $\Box^b N$ and $\Box^b N'$ are in $\Gamma$ and that $N' \subsetneq N$. It is trivial that any model of $\Gamma$ is also a model of $\Gamma \setminus \{\Box^b N\}$ and vice-versa.

Given a set of clauses $\Gamma$, the rule $(\mathcal{U}\,Fix)$ replaces a clause $\Box^b((P_1 \mathcal{U} P_2) \vee N) \in \Gamma$ with two clauses $\Box^b(P_2 \vee P_1 \vee N)$ and $\Box^b(P_2 \vee \circ(P_1 \mathcal{U} P_2) \vee N)$ obtaining a new set $\Gamma'$ = $(\Gamma \setminus \{\Box^b((P_1 \mathcal{U} P_2) \vee N)\}) \cup \{\Box^b(P_2 \vee P_1 \vee N), \Box^b(P_2 \vee \circ(P_1 \mathcal{U} P_2) \vee N)\}$. The two sets, $\Gamma$ and $\Gamma'$, are logically equivalent since the clause that contains the literal of the form $P_1 \mathcal{U} P_2$ is replaced with the clauses obtained by taking into account the inductive definition of the connective $\mathcal{U}$. Similarly, the rule $(\mathcal{R}\,Fix)$ replaces a clause $\Box^b((P_1 \mathcal{R} P_2) \vee N) \in \Gamma$ with two clauses $\Box^b(P_2 \vee N)$ and $\Box^b(P_1 \vee \circ(P_1 \mathcal{R} P_2) \vee N)$ obtaining a new set $\Gamma'$ = $(\Gamma \setminus \{\Box^b((P_1 \mathcal{R} P_2) \vee N)\}) \cup \{\Box^b(P_2 \vee N), \Box^b(P_1 \vee \circ(P_1 \mathcal{R} P_2) \vee N)\}$. The sets $\Gamma$ and $\Gamma'$ are logically equivalent because the clause that contains the literal of the form $P_1 \mathcal{R} P_2$ is substituted by the clauses obtained by using the inductive definition of the connective $\mathcal{R}$. In particular, every application of the rules $(\Box\,Fix)$ and $(\Diamond\,Fix)$ yields a new set of clauses that is logically equivalent to the initial set. Therefore, they are also sound. ■

**Proposition 4.5.2.** *The rule $(\mathcal{U}\,Set)$ is sound. Moreover, the initial and the target sets of every application of $(\mathcal{U}\,Set)$ are equisatisfiable.*

*Proof.* When the rule $(\mathcal{U}\,Set)$ is applied to a set of clauses $\Gamma$, a non-empty subset

$$\{\Box^{b_i}(P_1 \mathcal{U} P_2 \vee N_i) \mid 1 \leq i \leq n\}$$

is replaced with a set of clauses

$$\begin{aligned}\Psi = \;& \{P_2 \vee P_1 \vee N_i,\ P_2 \vee \circ(a\,\mathcal{U}\,P_2) \vee N_i \mid 1 \leq i \leq n\} \\ & \cup\ \mathsf{CNF}(\mathsf{def}(a, P_1, \Delta)) \\ & \cup \{\Box(\circ(P_1\,\mathcal{U}\,P_2) \vee \circ N_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\end{aligned}$$

where $\Delta = \mathsf{now}(\Gamma \setminus \{\Box^{b_i}(P_1\,\mathcal{U}\,P_2 \vee N_i) \mid 1 \leq i \leq n\})$, $a \in \mathsf{Prop}$ is fresh, $\mathsf{def}(a, P_1, \Delta) = \Box(\neg a \vee (P_1 \wedge \neg\Delta))$ if $\Delta \neq \emptyset$ and $\mathsf{def}(a, P_1, \Delta) = \Box\neg a$ if $\Delta = \emptyset$.

So the new set $\Gamma'$ is

$$(\Gamma \setminus \{\Box^{b_i}(P_1\,\mathcal{U}\,P_2 \vee N_i) \mid 1 \leq i \leq n\}) \cup \Psi.$$

We first show, in item $(a)$, that if $\Gamma'$ is satisfiable then $\Gamma$ is satisfiable and then, in item $(b)$, we show that if $\Gamma$ is satisfiable then $\Gamma'$ is satisfiable:

$(a)$ By Theorem 4.2.7, the set $\Psi$ and the following set $\Upsilon$ are equisatisfiable:

$$\begin{aligned}\Upsilon = \;& \{P_2 \vee P_1 \vee N_i,\ P_2 \vee \circ(a\,\mathcal{U}\,P_2) \vee N_i \mid 1 \leq i \leq n\} \\ & \cup\ \mathsf{def}(a, P_1, \Delta) \\ & \cup \{\Box(\circ(P_1\,\mathcal{U}\,P_2) \vee \circ N_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\end{aligned}$$

Consequently, the set $\Gamma'$ and the set

$$\Gamma'' = ((\Gamma' \setminus \Psi) \cup \Upsilon) = ((\Gamma \setminus \{\Box^{b_i}(P_1\,\mathcal{U}\,P_2 \vee N_i) \mid 1 \leq i \leq n\}) \cup \Upsilon)$$

are equisatisfiable. Let $\langle \mathcal{M}'', s''_0\rangle \models \Gamma''$, since $a$ does appear neither in the $P_2 \vee N_i$'s nor in $\Gamma$, we build a model $\mathcal{M}$ of $\Gamma$ in the following two cases:

1. If $\langle \mathcal{M}'', s''_0\rangle \models P_2 \vee N_i$ for all $i \in \{1, \dots, n\}$, then we can define the model $\mathcal{M}$ for $\Gamma$ as follows
   - $a \notin V_{\mathcal{M}}(s_j)$ for every $j \in I\!N$
   - $p \in V_{\mathcal{M}}(s_j)$ iff $p \in V_{\mathcal{M}''}(s''_j)$ for all $j \in I\!N$ and all $p \in \mathsf{Prop}$ such that $p \neq a$.
2. If $\langle \mathcal{M}'', s''_0\rangle \not\models P_2 \vee N_i$ for some $i \in \{1, \dots, n\}$, then it should be that $\langle \mathcal{M}'', s''_0\rangle \models \{P_1, \circ(a\,\mathcal{U}\,P_2)\}$. Let $x$ be the least $z \geq 1$ such that $\langle \mathcal{M}'', s''_z\rangle \models P_2$. If $x = 1$ then, since $a$ does not appear in $P_2$ and $\langle \mathcal{M}'', s''_0\rangle \models P_1$, the model $\mathcal{M}$ of $\Gamma$ can be built just as in the case 1 of this item $(a)$. If $x > 1$, then

   $$\langle \mathcal{M}'', s''_y\rangle \models \{a\} \cup \mathsf{def}(a, P_1, \Delta)$$

   for every $y$ such that $1 \leq y < x$. Note that $\Delta$ cannot be the empty set $\emptyset$ because in such case $a$ would not be true for any $y$ such that $1 \leq y < x$. As a consequence,

   $$\langle \mathcal{M}'', s''_y\rangle \models \{a\} \cup \{P_1 \wedge \neg\Delta\}$$

   for every $y$ such that $1 \leq y < x$.
   So that the model $\mathcal{M}$ of $\Gamma$ can be built just as in the case 1 of this item $(a)$.

$(b)$ Now we show the converse implication. Let $\langle \mathcal{M}, s_0\rangle \models \Gamma$, since $a$ does not appear in the $N_i$'s, we build a model $\mathcal{M}'$ of $\Gamma'$ in the following two cases.

1. Let us consider that $\langle \mathcal{M}, s_0\rangle \models N_i$ for all $i \in \{1,\dots,n\}$. Then we can define a model $\mathcal{M}'$ for $\Gamma'$ as follows:
   - $a \notin V_{\mathcal{M}'}(s'_j)$ for every $j \in I\!N$
   - $p \in V_{\mathcal{M}'}(s'_j)$ iff $p \in V_{\mathcal{M}}(s_j)$ for all $j \in I\!N$ and all $p \in \mathsf{Prop}$ such that $p \neq a$
2. If $\langle \mathcal{M}, s_0\rangle \not\models N_i$ for some $i \in \{1,\dots,n\}$, then it should be that $\langle \mathcal{M}, s_0\rangle \models P_1 \mathcal{U} P_2$. Let $x$ be the least $z \geq 0$ such that $\langle \mathcal{M}, s_z\rangle \models P_2$. If $x = 0$ then, since $a$ does not appear in $P_2$, a model $\mathcal{M}'$ of $\Gamma'$ can be built just as in the case 1 of item $(b)$. If $x > 0$, let $y$ be the greatest $z$ such that $0 \leq z < x$ and

   $$\langle \mathcal{M}, s_z\rangle \models \mathsf{now}(\Gamma \setminus \{\Box^{b_i}(P_1 \mathcal{U} P_2 \vee N_i) \mid 1 \leq i \leq n\}) \cup \{P_1 \mathcal{U} P_2\}.$$

   Note that at least $z = 0$ must satisfy the above set of clauses. As a consequence of the choice of $x$ and $y$, it holds that

   $$\langle \mathcal{M}, s_y\rangle \models \{P_1, \neg P_2, \circ((P_1 \wedge \neg \mathsf{now}(\Gamma \setminus \{\Box^{b_i}(P_1 \mathcal{U} P_2 \vee N_i) \mid 1 \leq i \leq n\}))\mathcal{U} P_2)\}.$$

   Besides, $\langle \mathcal{M}, s_y\rangle \models \mathsf{now}(\Gamma \setminus \{\Box^{b_i}(P_1 \mathcal{U} P_2 \vee N_i) \mid 1 \leq i \leq n\})$. So that, we can define a model $\mathcal{M}'$ for $\Gamma'$ as follows
   - $p \in V_{\mathcal{M}'}(s'_j)$ iff $p \in V_{\mathcal{M}}(s_{j+y})$ for all $j \in I\!N$ and all $p \in \mathsf{Prop}$ such that $p \neq a$
   - $a \notin V_{\mathcal{M}'}(s'_0)$
   - $a \in V_{\mathcal{M}'}(s'_j)$ for every $j \in \{1,\dots,x-y-1\}$
   - $a \notin V_{\mathcal{M}'}(s'_j)$ for every $j \geq x-y$. ■

As a particular case of Proposition 4.5.2, the derived rule $(\diamond Set)$ is also sound.

**Proposition 4.5.3.** *The operator* unnext *(see Definition 4.2.3) preserves satisfiability.*

*Proof.* If $\mathcal{M}$ is a model of $\Gamma$ then $\mathsf{unnext}(\Gamma)$ is true in the state $s_1$ of $\mathcal{M}$, which obviously gives a model for $\mathsf{unnext}(\Gamma)$. ■

Note that the equisatisfiability, in general, of initial and target sets of unnext cannot be ensured. For example, $\{p, \neg p, \circ q\}$ is unsatisfiable, but $\mathsf{unnext}(\{p, \neg p, \circ q\}) = \{q\}$ is satisfiable.

As a direct consequence of the above Propositions 4.5.1, 4.5.2 and 4.5.3, we have the following soundness theorem:

**Theorem 4.5.4.** *If the resolution system* TRS *produces a refutation from* $\Gamma$*, then* $\Gamma$ *is unsatisfiable.* ■

## 4.6 The Algorithm $\mathcal{SR}$ for Systematic TRS-Resolution

The nondeterministic application of the set of TRS-rules yields sound derivations but it does not guarantee completeness, even with the proviso of fairness. In this section we first introduce an algorithm for systematic resolution derivation called $\mathcal{SR}$ that uses the system TRS in a more (not fully) deterministic way which ensures completeness. Then, in the second subsection we provide some detailed examples of application of $\mathcal{SR}$. In the third and fourth subsections we respectively provide the termination and the worst case complexity results for $\mathcal{SR}$.

```
Input: A finite set of clauses $\Gamma$
Output: A resolution proof for $\Gamma$ called $\mathcal{D}(\Gamma)$

$\Gamma^0_0 := \Gamma$;  $i := 0$;  $j := 0$;
$\mathsf{sel\_ev\_set}_0 := \mathsf{fair\_select}(\Gamma^0_0)$;
loop
   if $\mathsf{sel\_ev\_set}_i \neq \emptyset$
     then $(\Gamma^1_i, \mathsf{sel\_ev\_set}^*_i) := \mathsf{apply\_}\mathcal{U}\mathsf{\_Set}(\Gamma^0_i, \mathsf{sel\_ev\_set}_i)$;  $j := 1$;
     else $\mathsf{sel\_ev\_set}^*_i := \emptyset$
   end if;
   $\Gamma^*_i := \mathsf{fix\_close}(\Gamma^j_i)$;
   if $\Box^b\bot \in \Gamma^*_i$ or $\mathsf{is\_cycling}(\mathcal{D}(\Gamma))$ then exit; end if;
   $\Gamma^0_{i+1} := \mathsf{unnext}(\Gamma^*_i)$;
   if $\mathsf{sel\_ev\_set}^*_i \cap \mathsf{event}(\Gamma^0_{i+1}) = \emptyset$ then $\mathsf{sel\_ev\_set}_{i+1} := \mathsf{fair\_select}(\Gamma^0_{i+1})$;
                                                   else $\mathsf{sel\_ev\_set}_{i+1} := \mathsf{sel\_ev\_set}^*_i$
   end if;
   $i := i + 1$;  $j := 0$;
end loop;
```

*Figure 4.10:* The Algorithm $\mathcal{SR}$ for Systematic TRS-Resolution

### 4.6.1 The Algorithm $\mathcal{SR}$

The algorithm $\mathcal{SR}$, for any input set of clauses $\Gamma$, obtains a finite *resolution proof*–called $\mathcal{D}(\Gamma)$– of the form

$$\Gamma^0_0 \mapsto \ldots \mapsto \Gamma^{h_0}_0 \Mapsto \Gamma^0_1 \mapsto \ldots \mapsto \Gamma^{h_1}_1 \Mapsto \ldots \Mapsto \Gamma^0_k \mapsto \ldots \mapsto \Gamma^{h_k}_k$$

As we will respectively show in Subsection 4.6.3 and Section 4.7, $\mathcal{D}(\Gamma)$ is always finite and $\mathcal{D}(\Gamma)$ is a refutation whenever the input set $\Gamma$ is unsatisfiable. When convenient, we represent $\mathcal{D}(\Gamma)$ by sequences of pairs

$$(\Gamma_0, \Gamma^*_0) \Mapsto (\Gamma_1, \Gamma^*_1) \Mapsto \ldots \Mapsto (\Gamma_k, \Gamma^*_k)$$

where $\Gamma_i$ and $\Gamma^*_i$ coincide with $\Gamma^0_i$ and $\Gamma^{h_i}_i$ respectively, for every $i \in \{0, \ldots, k\}$.

Derivations (refutations) obtained by means of the algorithm $\mathcal{SR}$ are called systematic derivations (refutations) and systematic TRS-derivations (refutations).

The construction of $\mathcal{D}(\Gamma)$, for any input $\Gamma$, is expressed by means of a while-program in Figure 4.10, called the algorithm $\mathcal{SR}$, which we explain next. In order to ensure that $\mathcal{D}(\Gamma)$ is finite, the rule $(\mathcal{U}\, Set)$ is applied exactly to one eventuality[5] (if there is any) between each two consecutive unnext-steps (see Subsection 4.4.1). For that purpose, the algorithm $\mathcal{SR}$ keeps two variables $\mathsf{sel\_ev\_set}_i$ and $\mathsf{sel\_ev\_set}^*_i$ for every $i \geq 0$. Both variables $\mathsf{sel\_ev\_set}_i$ and $\mathsf{sel\_ev\_set}^*_i$ take as value a set that is a singleton or empty, depending on whether $\Gamma^0_i$ contains at least one

[5] see Definition 2.2.1.

eventuality or not, respectively. The variable $\mathsf{sel\_ev\_set}_i$ stands for the selected eventuality in $\Gamma_i^0$, whereas $\mathsf{sel\_ev\_set}_i^*$ corresponds to the eventuality selected in every set of the sequence from $\Gamma_i^1$ until $\Gamma_i^{h_i}$.
The algorithm $\mathcal{SR}$ (see Figure 4.10) initializes both the set of clauses for starting the derivation $\Gamma_0^0$ to be the input set $\Gamma$ and the variable $\mathsf{sel\_ev\_set}_0$ to be either, a fairly selected eventuality in $\Gamma_0^0$ if there is any, or empty, otherwise. The expression $\mathsf{fair\_select}(\Gamma_g^\ell)$ encapsulates the *fair* selection of an eventuality in $\Gamma_g^\ell$, where fairness means that if an eventuality appears as available (from some moment onwards) for being selected whenever an eventuality must be selected, such eventuality cannot remain indefinitely unselected.
After initialization, each iteration-step of the algorithm $\mathcal{SR}$ serves to extend the derivation from $\Gamma_i^0$ to $\Gamma_i^*$ by means of the following process:

- The lines 4 to 8 serve to extend the derivation from $\Gamma_i^0$ to $\Gamma_i^*$.
  First, by lines 4-7, the rule $(\mathcal{U}\,Set)$ is applied exactly to the selected eventuality provided that $\mathsf{sel\_ev\_set}_i \neq \emptyset$. More precisely, if $\mathsf{sel\_ev\_set}_i = \{T\}$, then the rule $(\mathcal{U}\,Set)$ is applied to a partition of $\Gamma_i^0$ of the form $\Phi \cup (\Gamma_i^0 \upharpoonright \mathsf{sel\_ev\_set}_i)$,[6] producing the set $\Gamma_i^1$ in $\mathcal{D}(\Gamma)$. Additionally, as part of this application of the rule $(\mathcal{U}\,Set)$, the variable $\mathsf{sel\_ev\_set}_i^*$ gets the value $\{a\,\mathcal{U}\,P\}$ where $a\,\mathcal{U}\,P$ is the new eventuality introduced by the rule $(\mathcal{U}\,Set)$ with a fresh variable $a$. Otherwise, if $\mathsf{sel\_ev\_set}_i$ is empty, the rule $(\mathcal{U}\,Set)$ is not applied and $\mathsf{sel\_ev\_set}_i^*$ gets the value $\emptyset$.
  Second, by line 8, the remaining TRS-rules are repeatedly applied to $\Gamma_i^j$ (where $j = 0$ or $j = 1$) to construct $\Gamma_i^*$. The operation fix_close is introduced in Definition 4.4.8. Hence, $\Gamma_i^*$ is either TRS-closed (see Definition 4.4.6) or contains the empty clause. Moreover, the variable $\mathsf{sel\_ev\_set}_i^*$ is not changed by the operation fix_close. Hence, at line 11 the value of $\mathsf{sel\_ev\_set}_i^*$ is the same as at line 7.

- In line 9, the loop is exited if either the empty clause has been added to $\Gamma_i^*$ or a cycle in $\mathcal{D}(\Gamma)$ is detected according to the following definition.

  **Definition 4.6.1.** *Let* $\mathcal{D} = (\Gamma_0, \Gamma_0^*) \mapsto (\Gamma_1, \Gamma_1^*) \mapsto \ldots \mapsto (\Gamma_j, \Gamma_j^*) \mapsto \ldots \mapsto (\Gamma_k, \Gamma_k^*)$ *be a derivation (where* $0 \leq j \leq k$*), we say that* $\mathcal{D}$ *is* cycling *with respect to* $j$ *and* $k$ *iff* $\mathcal{D}$ *satisfies the following conditions*

  1. $\Box^b\bot \notin \Gamma_i^*$ *for every* $i \in \{0, \ldots, k\}$
  2. $\mathsf{now}(\mathsf{unnext}(\Gamma_k^*)) = \mathsf{now}(\Gamma_j)$
  3. *For every eventuality* $T$ *such that* $T \in \mathsf{Lits}(\mathsf{now}(\Gamma_g))$ *for all* $g \in \{j, \ldots, k\}$, *there exists* $h \in \{j, \ldots, k\}$ *such that* $\mathsf{sel\_ev\_set}_h = \{T\}$.

  The function is_cycling (line 9) is supposed to implement a test of the conditions (2) and (3) in Definition 4.6.1 on the current derivation $\mathcal{D}(\Gamma) = (\Gamma_0, \Gamma_0^*) \mapsto \ldots \mapsto (\Gamma_i, \Gamma_i^*)$.

- Otherwise, if the loop is not exited, the operator unnext (Definition 4.2.3) is applied to the TRS-closed set $\Gamma_i^*$ to yield $\Gamma_{i+1}^0$ (line 10), which will be the $\Gamma_i^0$ of the next step, after increasing $i$ (line 14).

[6] See Definition 4.2.1.

- Finally, the lines 11 to 13 serve to initialize the variable $\mathsf{sel\_ev\_set}_{i+1}$. Note that, after the application of the subsumption rule and/or of the operator $\mathsf{unnext}$, every clause that includes the selected eventuality $\mathsf{sel\_ev\_set}_i^*$ could have disappeared from the current $\Gamma_{i+1}^0$. In other words, although $\circ(a \,\mathcal{U}\, P)$ occurs in some $\Gamma_i^j$, it could happen that the selected eventuality $a \,\mathcal{U}\, P$ does not occur in $\Gamma_{i+1}^0$. The function $\mathsf{event}$ (line 11) returns the set of all eventualities occurring in an input set of clauses, that is

  **Definition 4.6.2.** *Let $\Psi$ be a set of clauses,* $\mathsf{event}(\Psi) = \{P_1 \,\mathcal{U}\, P_2 \,|\, \Box^b((P_1 \,\mathcal{U}\, P_2) \vee N) \in \Psi\}$.

  Therefore, if $\mathsf{sel\_ev\_set}_i^* \cap \mathsf{event}(\Gamma_{i+1}^0)$ is non-empty, then the selected eventuality remains selected. Otherwise, the function $\mathsf{fair\_select}$ is used to fairly select an eventuality from $\mathsf{event}(\Gamma_{i+1}^0)$.

We would like to remark the following three issues about the construction of $\mathcal{D}(\Gamma)$ by the algorithm $\mathcal{SR}$

1. Although $(Sbm)$ can be correctly applied whenever it is possible, in order to guarantee termination it suffices to apply $(Sbm)$ just before testing for a cycling derivation. This can be seen in the proof of Lemma 4.6.13.

2. For achieving completeness the operator $\mathsf{unnext}$ must always be applied to TRS-closed sets. Otherwise, equisatisfiability is not guaranteed because the operator $\mathsf{unnext}$ preserves satisfiability (Proposition 4.5.3) but, in general, does not preserve unsatisfiability.

3. In the intermediate sets $\Gamma_i^j$ of the process for obtaining $\Gamma_i^*$ from $\Gamma_i$, literals can appear that are neither in $\Gamma_i^*$ nor in $\Gamma_i$. This fact can be easily observed applying the algorithm $\mathcal{SR}$ to (e.g.) the set $\Gamma = \{p \,\mathcal{U}\, q, q\}$.

### *4.6.2 Examples*

In this subsection we apply the algorithm $\mathcal{SR}$ to some illustrative examples. As in the examples showed in Subsection 4.4.1, the clauses to which each rule affects have been underlined but we do not underline any formula in the applications of the operator $\mathsf{unnext}$. Since these TRS-derivations are built by using the algorithm $\mathcal{SR}$, in each figure we show the value of the variables $\mathsf{sel\_ev\_set}_i$ and $\mathsf{sel\_ev\_set}_i^*$.

**Example 4.6.3.** *The derivation in Figure 4.11 is a refutation of the unsatisfiable set of clauses $\{p, \Box(\neg p \vee \circ p), \Diamond\neg p\}$ that has been obtained following the algorithm $\mathcal{SR}$. First of all, in $\Gamma_0$, i.e., in $\Gamma_0^0$ the selected eventuality is $\Diamond\neg p$ and consequently* $\mathsf{sel\_ev\_set}_0 = \{\Diamond\neg p\}$. *Then, the application of the rule $(\Diamond\, Set)$ with context $\{p\}$ (always-clauses are excluded from the negation of the context) introduces a new propositional variable $a$ and transforms the clause $\Diamond\neg p$ into the last two clauses in $\Gamma_0^1$, $\neg p \vee \circ(a \,\mathcal{U}\, \neg p)$ and $\Box(\neg a \vee \neg p)$. Additionally, the value of* $\mathsf{sel\_ev\_set}_0^*$ *is set to $\{a \,\mathcal{U}\, \neg p\}$. Then, the rule applications that correspond to the operation* $\mathsf{fix\_close}$ *(line 8, Figure 4.10) are performed and the* TRS*-closed set $\Gamma_0^5$, i.e., $\Gamma_0^*$ is obtained. In order to obtain the* TRS*-closed set $\Gamma_0^5$ from $\Gamma_0^1$ the resolution rule $(Res)$ is applied three times and the subsumption rule $(Sbm)$ is applied once. In the application of the rule $(Res)$ to the set $\Gamma_0^1$, the clauses $p$ and $\Box(\neg p \vee \circ p)$ are the premises and the resolvent is the last clause $\circ p$ in $\Gamma_0^2$. Then the rule $(Res)$ is applied to the first and third clauses in $\Gamma_0^2$, giving the last clause $\circ(a \,\mathcal{U}\, \neg p)$ in $\Gamma_0^3$. Again, by*

$$
\begin{array}{lllll}
\Gamma_0 = & \Gamma_0^0 = & \{p, \Box(\neg p \vee \circ p), \underline{\Diamond \neg p}\} & (\Diamond\, Set) & \mathsf{sel\_ev\_set}_0 = \{\Diamond \neg p\} \\
& \Gamma_0^1 = & \{\underline{p}, \underline{\Box(\neg p \vee \circ p)}, \neg p \vee \circ(a\,\mathcal{U}\,\neg p), & (Res) & \mathsf{sel\_ev\_set}_0^* = \{a\,\mathcal{U}\,\neg p\} \\
& & \Box(\neg a \vee \neg p)\} & & \\
& \Gamma_0^2 = & \{\underline{p}, \Box(\neg p \vee \circ p), \underline{\neg p \vee \circ(a\,\mathcal{U}\,\neg p)}, & (Res) & \\
& & \Box(\neg a \vee \neg p), \circ p\} & & \\
& \Gamma_0^3 = & \{\underline{p}, \Box(\neg p \vee \circ p), \neg p \vee \circ(a\,\mathcal{U}\,\neg p), & (Res) & \\
& & \underline{\Box(\neg a \vee \neg p)}, \circ p, \circ(a\,\mathcal{U}\,\neg p)\} & & \\
& \Gamma_0^4 = & \{p, \Box(\neg p \vee \circ p), \underline{\neg p \vee \circ(a\,\mathcal{U}\,\neg p)}, & (Sbm) & \\
& & \Box(\neg a \vee \neg p), \circ p, \underline{\circ(a\,\mathcal{U}\,\neg p)}, \neg a\} & & \\
\Gamma_0^* = & \Gamma_0^5 = & \{p, \Box(\neg p \vee \circ p), \Box(\neg a \vee \neg p), \circ p, & (\mathsf{unnext}) & \\
& & \circ(a\,\mathcal{U}\,\neg p), \neg a\} & & \\
\Gamma_1 = & \Gamma_1^0 = & \{\Box(\neg p \vee \circ p), \Box(\neg a \vee \neg p), p, \underline{a\,\mathcal{U}\,\neg p}\} & (\mathcal{U}\, Set) & \mathsf{sel\_ev\_set}_1 = \{a\,\mathcal{U}\,\neg p\} \\
& \Gamma_1^1 = & \{\Box(\neg p \vee \circ p), \Box(\neg a \vee \neg p), \underline{p}, \underline{\neg p \vee a}, & (Res) & \mathsf{sel\_ev\_set}_1^* = \{b\,\mathcal{U}\,\neg p\} \\
& & \neg p \vee \circ(b\,\mathcal{U}\,\neg p), \Box(\neg b \vee a), & & \\
& & \Box(\neg b \vee \neg p)\} & & \\
& \Gamma_1^2 = & \{\Box(\neg p \vee \circ p), \underline{\Box(\neg a \vee \neg p)}, \underline{p}, \neg p \vee a, & (Res) & \\
& & \neg p \vee \circ(b\,\mathcal{U}\,\neg p), \Box(\neg b \vee a), & & \\
& & \Box(\neg b \vee \neg p), a\} & & \\
& \Gamma_1^3 = & \{\Box(\neg p \vee \circ p), \Box(\neg a \vee \neg p), p, \neg p \vee a, & (Res) & \\
& & \neg p \vee \circ(b\,\mathcal{U}\,\neg p), \Box(\neg b \vee a), & & \\
& & \Box(\neg b \vee \neg p), \underline{a}, \underline{\neg a}\} & & \\
\Gamma_1^* = & \Gamma_1^4 = & \{\Box(\neg p \vee \circ p), \Box(\neg a \vee \neg p), p, \neg p \vee a, & & \\
& & \neg p \vee \circ(b\,\mathcal{U}\,\neg p), \Box(\neg b \vee a), & & \\
& & \Box(\neg b \vee \neg p), a, \neg a, \bot\} & &
\end{array}
$$

*Figure 4.11:* Systematic TRS-refutation for the set of clauses $\{p, \Box(\neg p \vee \circ p), \Diamond \neg p\}$

*resolution of the first and fourth clauses in $\Gamma_0^3$, we obtain the clause $\neg a$ in $\Gamma_0^4$. By subsumption, the third clause is dropped, since it is subsumed by the sixth one, yielding $\Gamma_0^5$. Then, since no other rule can be applied, the operator* unnext *transforms the* TRS*-closed set $\Gamma_0^5$ into $\Gamma_1$. The latter represents the clauses that must be satisfied in the state $s_1$, provided that the state $s_0$ satisfies $\Gamma_0$. Since $a\,\mathcal{U}\,\neg p$ belongs to* event$(\Gamma_1^0)$*, the value of the set* sel_ev_set$_1$ *is $\{a\,\mathcal{U}\,\neg p\}$. Since the selected eventuality must be immediately handled (after the application of the operator* unnext*), the rule $(\mathcal{U}\,Set)$ is applied to $\Gamma_1 = \Gamma_1^0$. Note that, the context is again $\{p\}$. Then, $\Gamma_1^1$ contains four new clauses that substitute the clause $a\,\mathcal{U}\,\neg p$. A new propositional variable $b$ occurs in the new clauses and* sel_ev_set$_1^*$ *is $\{b\,\mathcal{U}\,\neg p\}$. Finally, by three consecutive applications of the rule $(Res)$ –which correspond to the operation* fix_close*– to the three underlined pairs of clauses, the empty clause is obtained in $\Gamma_1^4$. Note that the repeated context in $\Gamma_0$ and $\Gamma_1$ leads to find a contradiction.*

In the previous example, if we had used the rules $(\diamond\,Fix)$ and $(\mathcal{U}\,Fix)$ instead of the rules $(\diamond\,Set)$ and $(\mathcal{U}\,Set)$, we would have not obtained the empty clause. The following example illustrates this fact.

**Example 4.6.4.** *In Figure 4.12 we show a derivation whose initial set $\Gamma_0^0$ coincides with the initial unsatisfiable set considered in Example 4.6.3 (Figure 4.11). Whereas in the refutation presented in Figure 4.11 we first apply the rule $(\diamond\,Set)$, we start the derivation in Figure 4.12 by applying the rule $(\diamond\,Fix)$. Then the resolution rule $(Res)$ is applied twice and the subsumption rule $(Sbm)$ once, obtaining the* TRS*-closed set $\Gamma_0^4$. The application of the operator* unnext *to the set $\Gamma_0^4$ yields the set $\Gamma_1^0$ which contains the same clauses as $\Gamma_0^0$. By repeating this process, we could obtain an endless resolution derivation. Indeed, we will never obtain the empty clause unless we use the rules $(\diamond\,Set)$ and $(\mathcal{U}\,Set)$ in an appropriate manner. Obviously, the derivation in Figure 4.12 does not follow the algorithm $\mathcal{SR}$.*

The next example shows how the systematic TRS-resolution deals with clauses of the form $\Box\,P$.

**Example 4.6.5.** *In Figure 4.13 we provide a systematic* TRS*-refutation for the unsatisfiable set of clauses $\{\Box p, \diamond\neg p\}$. Since the procedure* fix_close *in the algorithm $\mathcal{SR}$ uses the function* BTL *(see Definitions 4.2.3 and 4.4.8) in order to decide whether a set of clauses is* TRS*-closed (Definition 4.4.6) and since* BTL *is based on the function* drop$_\Box$ *(Definition 4.2.3), clauses of the form $\Box\,P$ are considered always-clauses formed by one propositional literal and not now-clauses formed by one (basic) temporal literal. So following the algorithm $\mathcal{SR}$ we obtain the refutation in Figure 4.13. But we would like to remark that if we do not follow the algorithm $\mathcal{SR}$, it is possible to build the refutation in Figure 4.14.*

The following two examples show that the subsumption rule $(Sbm)$ is required to guarantee the termination of the algorithm $\mathcal{SR}$. In the case of Example 4.6.6 the concerned set of clauses is satisfiable, whereas in Example 4.6.7 is not.

**Example 4.6.6.** *Let us consider the derivation for the satisfiable set of clauses $\{(p\mathcal{U}\,q) \vee \Box r, \Box\neg p, \Box\neg q\}$ that is showed split (due to space reasons) in Figures 4.15 and 4.16. The derivation is only developed until the first application of the operator* unnext*, which yields the set $\Gamma_1^0$.*

*It is worth noting that if the rule $(Sbm)$ were not applied to the sets from $\Gamma_0^{11}$ to $\Gamma_0^{21}$, then the set $\Gamma_1$ would be*

$$\begin{array}{lll}
\Gamma_0^0 = & \{p, \Box(\neg p \lor \circ p), \underline{\Diamond \neg p}\} & (\Diamond \mathit{Fix}) \\
\Gamma_0^1 = & \{\underline{p}, \underline{\Box(\neg p \lor \circ p)}, \neg p \lor \circ\Diamond \neg p\} & (\mathit{Res}) \\
\Gamma_0^2 = & \{\underline{p}, \Box(\neg p \lor \circ p), \underline{\neg p \lor \circ\Diamond \neg p}, \circ p\} & (\mathit{Res}) \\
\Gamma_0^3 = & \{p, \Box(\neg p \lor \circ p), \underline{\neg p \lor \circ\Diamond \neg p}, \circ p, \underline{\circ\Diamond \neg p}\} & (\mathit{Sbm}) \\
\Gamma_0^4 = & \{p, \Box(\neg p \lor \circ p), \circ p, \circ\Diamond \neg p\} & (\mathsf{unnext}) \\
\Gamma_1^0 = & \{p, \Box(\neg p \lor \circ p), \Diamond \neg p\} & \\
 & \quad \dots &
\end{array}$$

*Figure 4.12:* Non-systematic TRS-derivation for the set of clauses $\{p, \Box(\neg p \lor \circ p), \Diamond \neg p\}$

$$\begin{array}{lllll}
\Gamma_0 = & \Gamma_0^0 = & \{\Box p, \underline{\Diamond \neg p}\} & (\Diamond \mathit{Set}) & \mathsf{sel\_ev\_set}_0 = \{\Diamond \neg p\} \\
 & \Gamma_0^1 = & \{\underline{\Box p}, \underline{\neg p \lor \circ(a \mathcal{U} \neg p)}, \Box \neg a\} & (\mathit{Res}) & \mathsf{sel\_ev\_set}_0^* = \{a \mathcal{U} \neg p\} \\
 & \Gamma_0^2 = & \{\Box p, \underline{\neg p \lor \circ(a \mathcal{U} \neg p)}, \underline{\circ(a \mathcal{U} \neg p)}, \Box \neg a\} & (\mathit{Sbm}) & \\
\Gamma_0^* = & \Gamma_0^3 = & \{\Box p, \circ(a \mathcal{U} \neg p), \Box \neg a\} & (\mathsf{unnext}) & \\
\Gamma_1 = & \Gamma_1^0 = & \{\Box p, \underline{a \mathcal{U} \neg p}, \Box \neg a\} & (\mathcal{U} \mathit{Set}) & \mathsf{sel\_ev\_set}_1 = \{a \mathcal{U} \neg p\} \\
 & \Gamma_1^1 = & \{\underline{\Box p}, \Box \neg a, \underline{\neg p \lor a}, & (\mathit{Res}) & \mathsf{sel\_ev\_set}_1^* = \{b \mathcal{U} \neg p\} \\
 & & \neg p \lor \circ(b \mathcal{U} \neg p), \Box \neg b\} & & \\
 & \Gamma_1^2 = & \{\Box p, \underline{\Box \neg a}, \neg p \lor a, \neg p \lor \circ(b \mathcal{U} \neg p), & (\mathit{Res}) & \\
 & & \Box \neg b, \underline{a}\} & & \\
\Gamma_1^* = & \Gamma_1^3 = & \{\Box p, \Box \neg a, \neg p \lor a, \neg p \lor \circ(b \mathcal{U} \neg p), & & \\
 & & \Box \neg b, a, \bot\} & &
\end{array}$$

*Figure 4.13:* Systematic TRS-refutation for the set of clauses $\{\Box p, \Diamond \neg p\}$

$$\begin{array}{lll}
\Gamma_0^0 = & \{\underline{\Box p}, \underline{\Diamond \neg p}\} & (\mathit{Res}) \\
\Gamma_0^1 = & \{\Box p, \Diamond \neg p, \bot\} &
\end{array}$$

*Figure 4.14:* Non-systematic TRS-refutation for the set of clauses $\{\Box p, \Diamond \neg p\}$

| | | | |
|---|---|---|---|
| $\Gamma_0 = \Gamma_0^0 =$ | $\{\underline{(p\,\mathcal{U}\,q) \vee \Box r}, \Box\neg p, \Box\neg q\}$ | $(\mathcal{U}\,Set)$ | $\mathsf{sel\_ev\_set}_0 = \{p\,\mathcal{U}\,q\}$ |
| $\Gamma_0^1 =$ | $\{\underline{q \vee p \vee \Box r}, q \vee \circ(a_1\,\mathcal{U}\,q) \vee \Box r, \Box\neg p, \Box\neg q, \Box\neg a_1\}$ | $(\Box Fix)$ | $\mathsf{sel\_ev\_set}_0^* = \{a_1\,\mathcal{U}\,q\}$ |
| $\Gamma_0^2 =$ | $\{q \vee p \vee r, q \vee p \vee \circ\Box r, \underline{q \vee \circ(a_1\,\mathcal{U}\,q) \vee \Box r}, \Box\neg p, \Box\neg q, \Box\neg a_1\}$ | $(\Box Fix)$ | |
| $\Gamma_0^3 =$ | $\{\underline{q \vee p \vee r}, q \vee p \vee \circ\Box r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, \Box\neg p, \underline{\Box\neg q}, \Box\neg a_1\}$ | $(Res)$ | |
| $\Gamma_0^4 =$ | $\{q \vee p \vee r, \underline{q \vee p \vee \circ\Box r}, q \vee \circ(a_1\,\mathcal{U}\,q) \vee r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, \Box\neg p, \underline{\Box\neg q}, \Box\neg a_1, p \vee r\}$ | $(Res)$ | |
| $\Gamma_0^5 =$ | $\{q \vee p \vee r, q \vee p \vee \circ\Box r, \underline{q \vee \circ(a_1\,\mathcal{U}\,q) \vee r}, q \vee \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, \Box\neg p, \underline{\Box\neg q}, \Box\neg a_1, p \vee r, p \vee \circ\Box r\}$ | $(Res)$ | |
| $\Gamma_0^6 =$ | $\{q \vee p \vee r, q \vee p \vee \circ\Box r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee r, \underline{q \vee \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r}, \Box\neg p, \underline{\Box\neg q}, \Box\neg a_1, p \vee r, p \vee \circ\Box r, \circ(a_1\,\mathcal{U}\,q) \vee r\}$ | $(Res)$ | |
| $\Gamma_0^7 =$ | $\{\underline{q \vee p \vee r}, q \vee p \vee \circ\Box r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, \underline{\Box\neg p}, \Box\neg q, \Box\neg a_1, p \vee r, p \vee \circ\Box r, \circ(a_1\,\mathcal{U}\,q) \vee r, \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r\}$ | $(Res)$ | |
| $\Gamma_0^8 =$ | $\{q \vee p \vee r, \underline{q \vee p \vee \circ\Box r}, q \vee \circ(a_1\,\mathcal{U}\,q) \vee r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, \underline{\Box\neg p}, \Box\neg q, \Box\neg a_1, p \vee r, p \vee \circ\Box r, \circ(a_1\,\mathcal{U}\,q) \vee r, \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, q \vee r\}$ | $(Res)$ | |
| $\Gamma_0^9 =$ | $\{q \vee p \vee r, q \vee p \vee \circ\Box r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, \underline{\Box\neg p}, \Box\neg q, \Box\neg a_1, \underline{p \vee r}, p \vee \circ\Box r, \circ(a_1\,\mathcal{U}\,q) \vee r, \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, q \vee r, q \vee \circ\Box r\}$ | $(Res)$ | |
| $\Gamma_0^{10} =$ | $\{q \vee p \vee r, q \vee p \vee \circ\Box r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee r, q \vee \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, \underline{\Box\neg p}, \Box\neg q, \Box\neg a_1, p \vee r, \underline{p \vee \circ\Box r}, \circ(a_1\,\mathcal{U}\,q) \vee r, \circ(a_1\,\mathcal{U}\,q) \vee \circ\Box r, q \vee r, q \vee \circ\Box r, r\}$ | $(Res)$ | |

*Figure 4.15:* Systematic TRS-derivation for the set of clauses $\{(p\,\mathcal{U}\,q) \vee \Box r, \Box\neg p, \Box\neg q\}$ (Part 1 of 2)

$$
\begin{array}{rcll}
\Gamma_0^{11} & = & \{q \lor p \lor r, q \lor p \lor \circ\Box r, & (Sbm) \\
 & & q \lor \circ(a_1 \mathcal{U}\, q) \lor r, \underline{q \lor \circ(a_1 \mathcal{U}\, q) \lor \circ\Box r}, & \\
 & & \Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, p \lor \circ\Box r, & \\
 & & \circ(a_1 \mathcal{U}\, q) \lor r, \circ(a_1 \mathcal{U}\, q) \lor \circ\Box r, & \\
 & & q \lor r, q \lor \circ\Box r, r, \underline{\circ\Box r}\} & \\[2ex]
\Gamma_0^{12} & = & \{q \lor p \lor r, q \lor p \lor \circ\Box r, & (Sbm) \\
 & & \underline{q \lor \circ(a_1 \mathcal{U}\, q) \lor r}, & \\
 & & \Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, p \lor \circ\Box r, & \\
 & & \circ(a_1 \mathcal{U}\, q) \lor r, \circ(a_1 \mathcal{U}\, q) \lor \circ\Box r, & \\
 & & q \lor r, q \lor \circ\Box r, \underline{r}, \circ\Box r\} & \\[2ex]
\Gamma_0^{13} & = & \{q \lor p \lor r, q \lor p \lor \circ\Box r, & (Sbm) \\
 & & \Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, p \lor \circ\Box r, & \\
 & & \underline{\circ(a_1 \mathcal{U}\, q) \lor r}, \circ(a_1 \mathcal{U}\, q) \lor \circ\Box r, & \\
 & & q \lor r, q \lor \circ\Box r, \underline{r}, \circ\Box r\} & \\[2ex]
\Gamma_0^{14} & = & \{q \lor p \lor r, q \lor p \lor \circ\Box r, & (Sbm) \\
 & & \Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, p \lor \circ\Box r, & \\
 & & \underline{\circ(a_1 \mathcal{U}\, q) \lor \circ\Box r}, & \\
 & & q \lor r, q \lor \circ\Box r, r, \underline{\circ\Box r}\} & \\[2ex]
\Gamma_0^{15} & = & \{q \lor p \lor r, \underline{q \lor p \lor \circ\Box r}, & (Sbm) \\
 & & \Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, p \lor \circ\Box r, & \\
 & & q \lor r, q \lor \circ\Box r, r, \underline{\circ\Box r}\} & \\[2ex]
\Gamma_0^{16} & = & \{\underline{q \lor p \lor r}, \Box\neg p, \Box\neg q, \Box\neg a_1, & (Sbm) \\
 & & p \lor r, p \lor \circ\Box r, & \\
 & & q \lor r, q \lor \circ\Box r, \underline{r}, \circ\Box r\} & \\[2ex]
\Gamma_0^{17} & = & \{\Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, \underline{p \lor \circ\Box r}, & (Sbm) \\
 & & q \lor r, q \lor \circ\Box r, r, \underline{\circ\Box r}\} & \\[2ex]
\Gamma_0^{18} & = & \{\Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, & (Sbm) \\
 & & q \lor r, \underline{q \lor \circ\Box r}, r, \underline{\circ\Box r}\} & \\[2ex]
\Gamma_0^{19} & = & \{\Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, & (Sbm) \\
 & & \underline{q \lor r}, \underline{r}, \circ\Box r\} & \\[2ex]
\Gamma_0^{20} & = & \{\Box\neg p, \Box\neg q, \Box\neg a_1, p \lor r, \underline{r}, \circ\Box r\} & (Sbm) \\[2ex]
\Gamma_0^{21} & = & \{\Box\neg p, \Box\neg q, \Box\neg a_1, \underline{q \lor r}, \underline{r}, \circ\Box r\} & (Sbm) \\[2ex]
\Gamma_0^* = \Gamma_0^{22} & = & \{\Box\neg p, \Box\neg q, \Box\neg a_1, r, \circ\Box r\} & (\mathsf{unnext}) \\[2ex]
\Gamma_1 = \Gamma_1^{0} & = & \{\Box\neg p, \Box\neg q, \Box\neg a_1, \Box r\} & \qquad\qquad \mathsf{sel\_ev\_set}_1 = \emptyset \\
 & & \ldots &
\end{array}
$$

*Figure 4.16:* Systematic TRS-derivation for the set of clauses $\{(p \mathcal{U}\, q) \lor \Box r, \Box\neg p, \Box\neg q\}$ (Part 2 of 2)

$$\{(a_1 \mathcal{U}\, q) \vee \Box r, \Box\neg p, \Box\neg q, \Box r, \Box\neg a_1\}$$

*and* $\mathsf{sel\_ev\_set}_1 = \{(a_1 \mathcal{U}\, q)\}$. *Indeed, every set* $\Gamma_i$ *(such that* $i \geq 1$*) obtained after* $i$ *unnext-steps would be of the form*

$$\{(a_i \mathcal{U}\, q) \vee \Box r, \Box\neg p, \Box\neg q, \Box r\} \cup \{\Box\neg a_h \mid 1 \leq h \leq i\}$$

*and* $\mathsf{sel\_ev\_set}_i = \{(a_i \mathcal{U}\, q)\}$. *Consequently, it would be impossible to obtain two sets* $\Gamma_j$ *and* $\Gamma_k$ *such that* $0 \leq j \leq k$ *and* $\mathsf{now}(\Gamma_j) = \mathsf{now}(\mathsf{unnext}(\Gamma_k^*))$. *Hence, the resolution process would not stop. Actually, from the eleven applications of the rule* $(Sbm)$ *in Figure 4.16, only the application of the rule* $(Sbm)$ *to the set* $\Gamma_0^{14}$ *is crucial. This application removes the clause* $\circ(a_1 \mathcal{U}\, q) \vee \circ\Box r$ *and yields the set* $\Gamma_0^{15}$. *Note that the other ten clauses removed, respectively, by the remaining ten applications of the rule* $(Sbm)$ *would also be removed by the application of the operator* unnext. *This example shows that the rule* $(Sbm)$ *is needed for completeness of the* TRS*-system.*

**Example 4.6.7.** *For the unsatisfiable set of clauses* $\{(p \mathcal{U}\, q) \vee (r \mathcal{U}\, s), \Box\neg p, \Box\neg q, \Box\neg s\}$, *if the first selected eventuality is* $p \mathcal{U}\, q$ *then the same problem as in Example 4.6.6 happens, but with* $(a_i \mathcal{U}\, q) \vee (r \mathcal{U}\, s)$ *instead of* $(a_i \mathcal{U}\, q) \vee \Box r$, *where* $a_i$ *is a fresh variable. This example shows that the rule* $(Sbm)$ *is also needed for refutational completeness of the* TRS*-system.*

One could think that if there are more than one eventuality that can be selected by the fair_select operation, then it could be that not all of the eventualities were right choices (e.g. because the program prevents the satisfaction of some of them). This view leads to the idea that wrong choices will have to be repaired by backtracking to the choice point and changing the selection. Moreover, sometimes one eventuality $\varphi$ must be necessarily fulfilled before another eventuality $\psi$. In those cases, one could think that selecting $\psi$ before selecting $\varphi$ could end up requiring backtracking. In the next example we illustrate that TRS-resolution does not need backtracking (independently of the selection strategy).

**Example 4.6.8.** *We consider the satisfiable set of clauses* $\Gamma = \{\Diamond q, \Diamond r, \Box(\neg q \vee \Box\neg r)\}$. *There are two eventualities,* $\Diamond q$ *and* $\Diamond r$, *that must be fulfilled, but the third clause* $\Box(\neg q \vee \Box\neg r)$ *states that once the eventuality* $\Diamond q$ *is fulfilled, the eventuality* $\Diamond r$ *cannot be fulfilled. So that, the eventuality* $\Diamond r$ *must be fulfilled before the eventuality* $\Diamond q$ *is fulfilled. The selection function* fair_select *could first select the eventuality* $\Diamond q$ *or could first select the eventuality* $\Diamond r$. *However, if* fair_select *first selects* $\Diamond q$, *it does not mean that* $\Diamond q$ *is fulfilled before* $\Diamond r$ *is fulfilled. Actually, since* $\Diamond r$ *must be fulfilled before* $\Diamond q$, *that is what happens. The corresponding cycling systematic* TRS*-derivation is shown in detail in Figures 4.17 and 4.18 (it is split due to space reasons).*

*After the first selection,* $\mathsf{sel\_ev\_set}_0 = \{\Diamond q\}$. *Then the application of the rule* $(\Diamond\, Set)$ *with context* $\{\Diamond r\}$ *generates the clauses* $q \vee \circ(a \mathcal{U}\, q)$ *and* $\Box(\neg a \vee \Box\neg r)$ *where* $a$ *is a fresh propositional variable. At the same time, the value of* $\mathsf{sel\_ev\_set}_0^*$ *is set to* $\{a \mathcal{U}\, q\}$. *Then, the rule applications that correspond to the* fix_close *operation (see Figure 4.10, line 8) are carried out and the* TRS*-closed set of clauses* $\Gamma_0^{12}$ *is obtained. Next, by the application of the operator* unnext, *the set* $\Gamma_1^0$ *is generated. Since the literal* $a \mathcal{U}\, q$ *belongs to* $\mathsf{event}(\Gamma_1^0)$, *it remains as the selected literal and, consequently, the rule* $(\mathcal{U}\, Set)$ *is applied to* $\Gamma_1^0$ *with* $a \mathcal{U}\, q$ *as selected literal (i.e.,* $\mathsf{sel\_ev\_set}_1 = \{a \mathcal{U}\, q\}$*) and with empty context, obtaining the set of clauses* $\Gamma_1^1$ *and setting* $\mathsf{sel\_ev\_set}_1^*$ *to* $\{b \mathcal{U}\, q\}$, *where* $b$ *is a fresh propositional variable. The operation* fix_close *that yields the* TRS*-closed set* $\Gamma_1^7$ *from* $\Gamma_1^1$, *encapsulates several applications of the rule* $(Res)$ *and the rule* $(Sbm)$. *The set* $\Gamma_2^0$ *is obtained from* $\Gamma_1^7$ *by using the operator* unnext. *Since the*

| | | |
|---|---|---|
| $\Gamma_0 = \Gamma_0^0 =$ | $\{\underline{\Diamond q}, \Diamond r, \Box(\neg q \lor \Box\neg r)\}$ | $(\Diamond Set)$ $\mathsf{sel\_ev\_set}_0 = \{\Diamond q\}$ |
| $\Gamma_0^1 =$ | $\{\underline{\Diamond r}, \Box(\neg q \lor \Box\neg r), q \lor \circ(a\,\mathcal{U}\,q), \Box(\neg a \lor \Box\neg r)\}$ | $(\Diamond Fix)$ $\mathsf{sel\_ev\_set}_0^* = \{a\,\mathcal{U}\,q\}$ |
| $\Gamma_0^2 =$ | $\{\underline{\Box(\neg q \lor \Box\neg r)}, q \lor \circ(a\,\mathcal{U}\,q), \Box(\neg a \lor \Box\neg r), r \lor \circ\Diamond r\}$ | $(\Box Fix)$ |
| $\Gamma_0^3 =$ | $\{q \lor \circ(a\,\mathcal{U}\,q), \underline{\Box(\neg a \lor \Box\neg r)}, r \lor \circ\Diamond r, \Box(\neg q \lor \neg r), \Box(\neg q \lor \circ\Box\neg r)\}$ | $(\Box Fix)$ |
| $\Gamma_0^4 =$ | $\{q \lor \circ(a\,\mathcal{U}\,q), \underline{r \lor \circ\Diamond r}, \Box(\neg q \lor \neg r), \underline{\Box(\neg q \lor \circ\Box\neg r)}, \Box(\neg a \lor \neg r), \Box(\neg a \lor \circ\Box\neg r)\}$ | $(Res)$ |
| $\Gamma_0^5 =$ | $\{q \lor \circ(a\,\mathcal{U}\,q), r \lor \circ\Diamond r, \underline{\Box(\neg q \lor \neg r)}, \Box(\neg q \lor \circ\Box\neg r), \Box(\neg a \lor \neg r), \Box(\neg a \lor \circ\Box\neg r), \underline{r \lor \neg q}\}$ | $(Res)$ |
| $\Gamma_0^6 =$ | $\{q \lor \circ(a\,\mathcal{U}\,q), \underline{r \lor \circ\Diamond r}, \Box(\neg q \lor \neg r), \Box(\neg q \lor \circ\Box\neg r), \Box(\neg a \lor \neg r), \underline{\Box(\neg a \lor \circ\Box\neg r)}, r \lor \neg q, \neg q\}$ | $(Res)$ |
| $\Gamma_0^7 =$ | $\{q \lor \circ(a\,\mathcal{U}\,q), r \lor \circ\Diamond r, \Box(\neg q \lor \neg r), \Box(\neg q \lor \circ\Box\neg r), \underline{\Box(\neg a \lor \neg r)}, \Box(\neg a \lor \circ\Box\neg r), r \lor \neg q, \neg q, \underline{r \lor \neg a}\}$ | $(Res)$ |
| $\Gamma_0^8 =$ | $\{\underline{q \lor \circ(a\,\mathcal{U}\,q)}, r \lor \circ\Diamond r, \Box(\neg q \lor \neg r), \Box(\neg q \lor \circ\Box\neg r), \Box(\neg a \lor \neg r), \Box(\neg a \lor \circ\Box\neg r), r \lor \neg q, \underline{\neg q}, r \lor \neg a, \neg a\}$ | $(Res)$ |
| $\Gamma_0^9 =$ | $\{q \lor \circ(a\,\mathcal{U}\,q), r \lor \circ\Diamond r, \Box(\neg q \lor \neg r), \Box(\neg q \lor \circ\Box\neg r), \Box(\neg a \lor \neg r), \Box(\neg a \lor \circ\Box\neg r), \underline{r \lor \neg q}, \underline{\neg q}, r \lor \neg a, \neg a, \circ(a\,\mathcal{U}\,q)\}$ | $(Sbm)$ |
| $\Gamma_0^{10} =$ | $\{q \lor \circ(a\,\mathcal{U}\,), r \lor \circ\Diamond r, \Box(\neg q \lor \neg r), \Box(\neg q \lor \circ\Box\neg r), \Box(\neg a \lor \neg r), \Box(\neg a \lor \circ\Box\neg r), \neg q, \underline{r \lor \neg a}, \underline{\neg a}, \circ(a\,\mathcal{U}\,q)\}$ | $(Sbm)$ |
| $\Gamma_0^{11} =$ | $\{\underline{q \lor \circ(a\,\mathcal{U}\,q)}, r \lor \circ\Diamond r, \Box(\neg q \lor \neg r), \Box(\neg q \lor \circ\Box\neg r), \Box(\neg a \lor \neg r), \Box(\neg a \lor \circ\Box\neg r), \neg q, \neg a, \underline{\circ(a\,\mathcal{U}\,q)}\}$ | $(Sbm)$ |
| $\Gamma_0^* = \Gamma_0^{12} =$ | $\{r \lor \circ\Diamond r, \Box(\neg q \lor \neg r), \Box(\neg q \lor \circ\Box\neg r), \Box(\neg a \lor \neg r), \Box(\neg a \lor \circ\Box\neg r), \neg q, \neg a, \circ(a\,\mathcal{U}\,q)\}$ | (unnext) |

*Figure 4.17:* Cycling systematic TRS-derivation for $\{\Diamond q, \Diamond r, \Box(\neg q \lor \Box\neg r)\}$ (Part 1 of 2)

$$
\begin{array}{rrlll}
\Gamma_1 = & \Gamma_1^0 = & \{\Box(\neg q \vee \neg r), \Box(\neg q \vee \circ\Box\neg r), & (\mathcal{U}\,Set) & \mathsf{sel\_ev\_set}_0 = \{a\,\mathcal{U}\,q\} \\
& & \Box(\neg a \vee \neg r), \Box(\neg a \vee \circ\Box\neg r), \underline{a\,\mathcal{U}\,q}\} & & \\[6pt]
& \Gamma_1^1 = & \{\Box(\neg q \vee \neg r), \Box(\neg q \vee \circ\Box\neg r), & (Res) & \\
& & \underline{\Box(\neg a \vee \neg r)}, \Box(\neg a \vee \circ\Box\neg r), & & \\
& & \underline{q \vee a}, q \vee \circ(b\,\mathcal{U}\,q), \Box\neg b\} & & \\[6pt]
& \Gamma_1^2 = & \{\underline{\Box(\neg q \vee \neg r)}, \Box(\neg q \vee \circ\Box\neg r), & (Res) & \\
& & \Box(\neg a \vee \neg r), \Box(\neg a \vee \circ\Box\neg r), & & \\
& & q \vee a, q \vee \circ(b\,\mathcal{U}\,q), \Box\neg b, \underline{q \vee \neg r}\} & & \\[6pt]
& \Gamma_1^3 = & \{\Box(\neg q \vee \neg r), \underline{\Box(\neg q \vee \circ\Box\neg r)}, & (Res) & \\
& & \Box(\neg a \vee \neg r), \Box(\neg a \vee \circ\Box\neg r), & & \\
& & \underline{q \vee a}, q \vee \circ(b\,\mathcal{U}\,q), \Box\neg b, q \vee \neg r, \neg r\} & & \\[6pt]
& \Gamma_1^4 = & \{\Box(\neg q \vee \neg r), \Box(\neg q \vee \circ\Box\neg r), & (Res) & \\
& & \Box(\neg a \vee \neg r), \underline{\Box(\neg a \vee \circ\Box\neg r)}, q \vee a, & & \\
& & q \vee \circ(b\,\mathcal{U}\,q), \Box\neg b, q \vee \neg r, \neg r, \underline{a \vee \circ\Box\neg r}\} & & \\[6pt]
& \Gamma_1^5 = & \{\Box(\neg q \vee \neg r), \Box(\neg q \vee \circ\Box\neg r), & (Sbm) & \\
& & \Box(\neg a \vee \neg r), \Box(\neg a \vee \circ\Box\neg r), q \vee a, & & \\
& & q \vee \circ(b\,\mathcal{U}\,q), \Box\neg b, \underline{q \vee \neg r}, \underline{\neg r}, a \vee \circ\Box\neg r, \circ\Box\neg r\} & & \\[6pt]
& \Gamma_1^6 = & \{\Box(\neg q \vee \neg r), \Box(\neg q \vee \circ\Box\neg r), & (Sbm) & \\
& & \Box(\neg a \vee \neg r), \Box(\neg a \vee \circ\Box\neg r), q \vee a, & & \\
& & q \vee \circ(b\,\mathcal{U}\,q), \Box\neg b, \neg r, \underline{a \vee \circ\Box\neg r}, \underline{\circ\Box\neg r}\} & & \\[6pt]
\Gamma_1^* = & \Gamma_1^7 = & \{\Box(\neg q \vee \neg r), \Box(\neg q \vee \circ\Box\neg r), & (\mathsf{unnext}) & \\
& & \Box(\neg a \vee \neg r), \Box(\neg a \vee \circ\Box\neg r), q \vee a, & & \\
& & q \vee \circ(b\,\mathcal{U}\,q), \Box\neg b, \neg r, \circ\Box\neg r\} & & \\[6pt]
\Gamma_2 = & \Gamma_2^0 = & \{\underline{\Box(\neg q \vee \neg r)}, \Box(\neg q \vee \circ\Box\neg r), & (Sbm) & \mathsf{sel\_ev\_set}_2 = \emptyset \\
& & \Box(\neg a \vee \neg r), \Box(\neg a \vee \circ\Box\neg r), \Box\neg b, \underline{\Box\neg r}\} & & \\[6pt]
& \Gamma_2^1 = & \{\Box(\neg q \vee \circ\Box\neg r), \underline{\Box(\neg a \vee \neg r)}, & (Sbm) & \mathsf{sel\_ev\_set}_2^* = \emptyset \\
& & \Box(\neg a \vee \circ\Box\neg r), \Box\neg b, \underline{\Box\neg r}\} & & \\[6pt]
\Gamma_2^* = & \Gamma_2^2 = & \{\Box(\neg q \vee \circ\Box\neg r), \Box(\neg a \vee \circ\Box\neg r), & & \\
& & \Box\neg b, \Box\neg r\} & &
\end{array}
$$

$$
\mathsf{now}(\mathsf{unnext}(\Gamma_2^2)) = \mathsf{now}(\Gamma_2^0)
$$

$$
\{\neg q, r, \neg a\} \mapsto \{q, \neg r, \neg b\} \mapsto \{\neg r, \neg b\} \mapsto \{\neg r, \neg b\} \cdots
$$

*Figure 4.18:* Cycling systematic TRS-derivation for $\{\Diamond q, \Diamond r, \Box(\neg q \vee \Box\neg r)\}$ (Part 2 of 2)

*set* $\mathsf{event}(\Gamma_2^0)$ *is empty, the value of* $\mathsf{sel\_ev\_set}_2$ *as well as the value of* $\mathsf{sel\_ev\_set}_2^*$ *is the empty set. Therefore no context-dependent rule is applied to* $\Gamma_2^0$ *and we get the* TRS*-closed set* $\Gamma_2^2$ *by applying the rule* $(Sbm)$ *twice. At this point the derivation is cycling with respect to* $j = 2$ *and* $k = 2$ *(see Definition 4.6.1). In particular this means that* $\mathsf{now}(\mathsf{unnext}(\Gamma_2^2)) = \mathsf{now}(\Gamma_2^0)$*. The sets* $\Gamma_0^* = \Gamma_0^{12}$*,* $\Gamma_1^* = \Gamma_1^7$ *and* $\Gamma_2^* = \Gamma_2^1$ *characterize a collection of models for the initial set of clauses* $\Gamma$*. All the models of such collection make true the literals* $\{\neg q, r, \neg a\}$ *in* $s_0$*, the literals* $\{\neg r, \neg b\}$ *in* $s_1$ *and also the literals* $\{\neg r, \neg b\}$ *in all the states* $s_j$ *such that* $j \geq 2$*. Additionally,* $q$ *must be true in* $s_k$ *for some* $k \geq 1$*. Therefore, if we choose to make* $q$ *true as soon as possible, i.e. in the state* $s_1$*, we can obtain an ultimately periodic model* $\mathcal{M}$ *of* $\Gamma$ *with states* $s_0, s_1, s_2, \ldots$ *such that* $V_{\mathcal{M}}(s_0) = \{r\}$*,* $V_{\mathcal{M}}(s_1) = \{q\}$ *and* $V_{\mathcal{M}}(s_j) = \emptyset$ *for every* $j \geq 2$*.*

In Example 4.6.8 we can see that the strategy for selecting eventualities does not compromise the completeness of TRS-resolution. However it can affect efficiency. In particular, if we had selected the eventuality $\diamond r$ instead of the eventuality $\diamond q$, the derivation would have been considerably longer.

**Remark 4.6.9.** *Note that when* $\Gamma$ *is a satisfiable set of (non-temporal) classical propositional clauses, the derivation* $\mathcal{D}(\Gamma)$ *obtained by the algorithm* $\mathcal{SR}$ *is of the form* $\Gamma_0^0 \mapsto \ldots \mapsto \Gamma_0^{h_0} \Mapsto \Gamma_1^0$*, and it can also be represented as* $(\Gamma_0, \Gamma_0^*) \Mapsto (\Gamma_1, \Gamma_1^*)$*, where* $\Gamma_0 = \Gamma_0^0 = \Gamma$*,* $\Gamma_0^{h_0} = \Gamma_0^*$*,* $\Gamma_1 = \Gamma_1^* = \mathsf{unnext}(\Gamma_0^*) = \emptyset$*. The set* $\Gamma_1^0$ *–which is at the same time* $\Gamma_1$ *and* $\Gamma_1^*$*– is* TRS*-closed and additionaly produces a cycle because* $\mathcal{D}(\Gamma)$ *verifies the three items of Definition 4.6.1 and, in particular the second one since* $\mathsf{now}(\mathsf{unnext}(\Gamma_1^*)) = \mathsf{now}(\Gamma_1)$*. So the cycle is from* $\Gamma_1^0$ *to* $\Gamma_1^0$*. Sets of temporal clauses, e.g. the singleton* $\{\circ P\}$*, can also give rise to this kind of cycling derivation ended in an empty set. However, the singleton* $\{\square P\}$ *produces a cycle with non-empty set of clauses. In general, every systematic derivation that is not a refutation becomes cyclic.*

Along the rest of the chapter, we will denote by $\mathcal{D}(\Gamma)$ any derivation of the form $(\Gamma_0, \Gamma_0^*) \Mapsto (\Gamma_1, \Gamma_1^*) \Mapsto \ldots \Mapsto (\Gamma_j, \Gamma_j^*) \Mapsto \ldots \Mapsto (\Gamma_k, \Gamma_k^*)$ obtained by $\mathcal{SR}$ with initial set $\Gamma_0 = \Gamma$. In particular, $\mathcal{D}(\Gamma)$ may be a refutation or a cycling derivation with respect to $j$ and $k$.

### 4.6.3 Termination

In this subsection we show that the algorithm $\mathcal{SR}$ always obtains either a refutation or a cycling derivation after a finite number of iterations. Remember that we assume that $\mathcal{SR}$ uses a fair strategy for selecting eventualities.

The rule $(\mathcal{U}\,Set)$ introduces new eventualities involving fresh variables. In order to justify that derivations that (potentially) use the rule $(\mathcal{U}\,Set)$ are finite, we have to show that, whenever a refutation is not obtained, the cycling conditions in Definition 4.6.1, in particular its third requirement, will be satisfied after a finite number of iteration steps. In other words, the termination proof of $\mathcal{SR}$ requires to show that the algorithm cannot generate an infinite number of new propositional variables. A priori, there are two ways for generating new propositional variables in $\mathcal{SR}$. The first is the translation to CNF applied in the output to the rule $(\mathcal{U}\,Set)$. However, no new variable is introduced by $\mathcal{SR}$ in this way. The reason is that the translation to CNF is applied to a formula that only needs DtNF-rules to be in CNF and DtNF-rules do not use extra variables (see Proposition 4.2.6).

The second source of new propositional variables is the explicit occurrence of a fresh variable in the consequent of the rule $(\mathcal{U}\,Set)$. However, as we will show, the sequence of new eventualities produced by successive applications of the rule $(\mathcal{U}\, Set)$ is always finite. There is a twofold reason for the latter. On one hand, the clauses defining a new variable (see function def in Figure 4.5) are always-clauses, which are excluded from the negated context. On the other hand, in the algorithm $\mathcal{SR}$, the rule $(\mathcal{U}\, Set)$ is always applied to sets where the propositional variables introduced (as fresh) by previous applications of $(\mathcal{U}\, Set)$ are also out of the context.

In order to prove the termination result, we first define the set $\mathsf{univlit}(\Gamma)$ (Definition 4.6.10) formed by all the literals that could appear in the clauses obtained from $\Gamma$ by means of all the TRS-rules with the exception of the rule $(\mathcal{U}\, Set)$ (and the derived rule $(\diamond\, Set)$). Then the closure of a set of clauses $\Gamma$ (Definition 4.6.11) is formed by all the clauses that can be generated from the literals in $\mathsf{univlit}(\Gamma)$.

**Definition 4.6.10.** *Let $\Gamma$ be a set of clauses. The set* $\mathsf{univlit}(\Gamma)$ *is the smallest set of literals defined as follows*[7]

- $\mathsf{Lits}(\Gamma) \subseteq \mathsf{univlit}(\Gamma)$
- *If* $L \in \mathsf{univlit}(\Gamma)$*, then* $\widetilde{L} \in \mathsf{univlit}(\Gamma)$
- *If* $P_1 \mathcal{U} P_2 \in \mathsf{univlit}(\Gamma)$*, then* $\{\circ(P_1 \mathcal{U} P_2), P_1, P_2\} \subseteq \mathsf{univlit}(\Gamma)$
- *If* $P_1 \mathcal{R} P_2 \in \mathsf{univlit}(\Gamma)$*, then* $\{\circ(P_1 \mathcal{R} P_2), P_1, P_2\} \subseteq \mathsf{univlit}(\Gamma)$
- *If* $\diamond P \in \mathsf{univlit}(\Gamma)$*, then* $\{\circ\diamond P, P\} \subseteq \mathsf{univlit}(\Gamma)$
- *If* $\Box P \in \mathsf{univlit}(\Gamma)$*, then* $\{\circ\Box P, P\} \subseteq \mathsf{univlit}(\Gamma)$
- *If* $\circ L \in \mathsf{univlit}(\Gamma)$*, then* $L \in \mathsf{univlit}(\Gamma)$.

The set $\mathsf{univlit}(\Gamma)$ is finite for any set of clauses $\Gamma$ since we only consider finite sets of clauses and finite clauses. Now, we define the closure of a set of clauses.

**Definition 4.6.11.** *Let $\Gamma$ be a set of clauses. The set* $\mathsf{closure}(\Gamma)$ *is the set formed by all the clauses $C$ such that* $\mathsf{Lits}(C) \subseteq \mathsf{univlit}(\Gamma)$.

As a consequence of the finiteness of $\mathsf{univlit}(\Gamma)$ and of the fact that clauses do not contain repeated literals, the set $\mathsf{closure}(\Gamma)$ is also finite.

We additionally consider the notions of *direct descendant* and *sequence of descendants*.

**Definition 4.6.12.** *Let $\mathcal{D}(\Gamma) = (\Gamma_0, \Gamma_0^*) \mapsto \ldots \mapsto (\Gamma_k, \Gamma_k^*)$ be the derivation constructed by the algorithm $\mathcal{SR}$ (Figure 4.10). We say that an eventuality $T'$ is the* direct descendant *of an eventuality $T$ in $\mathcal{D}(\Gamma)$ iff for some $i \in \{0, \ldots, k\}$:* $\mathsf{sel\_ev\_set}_i = \{T\}$ *and* $\mathsf{sel\_ev\_set}_i^* = \{T'\}$. *Let $S = T_0, T_1, \ldots, T_n$ be a sequence of eventualities. We say that $S$ is the* sequence of descendants *of $T_0$ in $\mathcal{D}(\Gamma)$ iff $T_{i+1}$ is a direct descendant of $T_i$ in $\mathcal{D}(\Gamma)$ for all $i \in \{0, \ldots, n-1\}$.*

For example, $\diamond\neg p, a\mathcal{U}\neg p, b\mathcal{U}\neg p$ is the sequence of descendants of $\diamond\neg p$ in the derivation in Example 4.6.5.

Next we first show that for all $\mathcal{D}(\Gamma)$ and every selected eventuality $T$ in $\mathcal{D}(\Gamma)$, the sequence of descendants of $T$ in $\mathcal{D}(\Gamma)$ is finite (Lemma 4.6.13). The proof is based on the fact that the algorithm $\mathcal{SR}$ follows a specific strategy with two crucial features. First, the algorithm keeps at

[7] Remember that $\mathsf{Lits}(\Box^b(L_1 \vee \ldots \vee L_n)) = \{L_1, \ldots, L_n\}$ and $\mathsf{Lits}(\Gamma) = \bigcup_{C\in\Gamma} \mathsf{Lits}(C)$.

most one selected eventuality to which the rule $(\mathcal{U}\,Set)$ can be applied and when a new eventuality is generated, by application of $(\mathcal{U}\,Set)$, that new eventuality has priority to become the selected eventuality for the next application of the rule $(\mathcal{U}\,Set)$ (after an unnext-step). Second, the rule $(\mathcal{U}\,Set)$ is applied before any other rule in each iteration step. As a consequence of these two crucial features of the strategy followed by the algorithm $\mathcal{SR}$, when the rule $(\mathcal{U}\,Set)$ is applied with selected eventuality $T$, eventualities generated by previous applications of $(\mathcal{U}\,Set)$ do not appear in the set of clauses $\Phi$ (see Figure 4.5) and the propositional variables introduced (as fresh) by previous applications of $(\mathcal{U}\,Set)$ appear only in always-clauses. Hence, the context (Definition 4.3.3) is always a subset of the closure set, which is finite. Therefore, since the number of possible different contexts is finite, if the sequence of descendants of an eventuality were infinite, some context would be repeated, but context repetition produces the end of the sequence of descendants (as shown in the proof of Lemma 4.6.13).

**Lemma 4.6.13.** *For all $\mathcal{D}(\Gamma)$ and every selected eventuality $T$ in $\mathcal{D}(\Gamma)$, the sequence of descendants of $T$ in $\mathcal{D}(\Gamma)$ is finite.*

*Proof.* Let $T$ be $P_0\,\mathcal{U}\,P$. Suppose that $T$ occurs in the set $\Gamma_0^0$ in $\mathcal{D}(\Gamma)$, $\mathsf{sel\_ev\_set}_0 = \{P_0\,\mathcal{U}\,P\}$ and the sequence of descendants of $T$ in $\mathcal{D}(\Gamma)$ is infinite. When the rule $(\mathcal{U}\,Set)$ is applied to a partition of $\Gamma_0^0$ of the form $\Phi_0 \cup \Gamma_0^0 \upharpoonright \{P_0\,\mathcal{U}\,P\}$, the set $\Gamma_0^0 \upharpoonright \{P_0\,\mathcal{U}\,P\}$ is replaced with the union of the following five disjoint sets of clauses

$$\begin{aligned}
\Psi_0^1 &= \{P \vee P_0 \vee N_0 \mid \Box^b((P_0\,\mathcal{U}\,P) \vee N_0) \in \Gamma_0\}\\
\Psi_0^2 &= \{P \vee \circ(a_1\,\mathcal{U}\,P) \vee N_0 \mid \Box^b((P_0\,\mathcal{U}\,P) \vee N_0) \in \Gamma_0\}\\
\Psi_0^3 &= \{\Box(\circ(P_0\,\mathcal{U}\,P) \vee \circ N_0) \mid \Box((P_0\,\mathcal{U}\,P) \vee N_0) \in \Gamma_0\}\\
\Psi_0^4 &= \{\Box(\neg a_1 \vee P_0)\}\\
\Psi_0^5 &= \mathsf{CNF}(\Box(\neg a_1 \vee \neg\mathsf{now}(\Phi_0)))
\end{aligned}$$

where $\Psi_0^4 \cup \Psi_0^5$ corresponds to $\mathsf{CNF}(\mathsf{def}(a_1, P_0, \mathsf{now}(\Phi_0)))$ (see Figure 4.5).

Hence, the set $\Gamma_0^1$ is the union of $\Phi_0$ and the above five sets, and the new selected eventuality is $a_1\,\mathcal{U}\,P$, i.e., $\mathsf{sel\_ev\_set}_0^* = \{a_1\,\mathcal{U}\,P\}$. The fresh variable $a_1$ only occurs in $\Psi_0^2$ and $\Psi_0^4 \cup \Psi_0^5$. The latter is a set of always-clauses, and the occurrences of $a_1$ in $\Psi_0^4 \cup \Psi_0^5$ are not preceded by $\circ$. Consequently, after the operations $\mathsf{fix\_close}$ and $\mathsf{unnext}$ (lines 8 and 10 in Figure 4.10), all the occurrences of $a_1$ in the set $\Gamma_1^0$ are either in an always-clause or in a now-clause that comes from $\Psi_0^2$. Hence, the only now-clauses where $a_1$ occurs in $\Gamma_1^0$ are of the form $N \vee a_1\,\mathcal{U}\,P$, where $a_1\,\mathcal{U}\,P$ is the new selected eventuality. Hence, the next application of the rule $(\mathcal{U}\,Set)$ does not introduce any occurrence of $a_1$ in the negated context, because always-clauses and clauses containing $a_1\,\mathcal{U}\,P$ are both excluded from the context. Moreover, $\mathsf{CNF}(\Box(\neg a_1 \vee \neg\mathsf{now}(\Phi_0)))$ does not contain any other fresh variable (apart from $a_1$). The reason is that $\mathsf{DtNF}(\Box(\neg a_1 \vee \neg\mathsf{now}(\Phi_0)))$ is already in conjunctive normal form, so the only transformation that uses new fresh variables –which is detailed in the proof of Theorem 4.2.7– is left out.

The above reasoning about the construction of $\Gamma_1^0$ from $\Gamma_0^0$ can be generalized to the construction of $\Gamma_{i+1}^0$ from $\Gamma_i^0$ with selected eventuality $a_i\,\mathcal{U}\,P$ to obtain a direct descendant $a_{i+1}\,\mathcal{U}\,P$ as follows. When the rule $(\mathcal{U}\,Set)$ is applied to a partition of $\Gamma_i^0$ of the form $\Phi_i \cup \Gamma_i^0 \upharpoonright$

$\{a_i \mathcal{U} P\}$, then the consequent $\Gamma_i^1$ is the union of $\Phi_i$ and the following five disjoint sets

$$\begin{aligned}
\Psi_i^1 &= \{P \vee a_i \vee N_i \mid \Box^b((a_i \mathcal{U} P) \vee N_i) \in \Gamma_i\}\\
\Psi_i^2 &= \{P \vee \circ(a_{i+1} \mathcal{U} P) \vee N_i \mid \Box^b((a_i \mathcal{U} P) \vee N_i) \in \Gamma_i\}\\
\Psi_i^3 &= \{\Box(\circ(a_i \mathcal{U} P) \vee \circ N_i) \mid \Box((a_i \mathcal{U} P) \vee N_i) \in \Gamma_i\}\\
\Psi_i^4 &= \{\Box(\neg a_1 \vee P_0), \Box(\neg a_2 \vee a_1), \dots, \Box(\neg a_i \vee a_{i-1}), \Box(\neg a_{i+1} \vee a_i)\}\\
\Psi_i^5 &= \mathsf{CNF}(\Box(\neg a_{i+1} \vee \neg \mathsf{now}(\Phi_i)))
\end{aligned}$$

where $(\Psi_i^4 \setminus \Psi_{i-1}^4) \cup \Psi_i^5$ corresponds to $\mathsf{CNF}(\mathsf{def}(a_{i+1}, a_i, \mathsf{now}(\Phi_i)))$ whenever $i \geq 1$ (see Figure 4.5). Now, the fresh variables $a_1, \dots, a_i, a_{i+1}$ occur in the above five sets $\Psi_i^j$. The occurrences of fresh variables in $\Psi_i^2 \cup \Psi_i^4 \cup \Psi_i^5$ are not filtered to the negated context in $\Gamma_{i+1}^0$ by the reasons explained above for $\Gamma_1^0$. Regarding the occurrences of $a_i$ in the set $\Psi_i^1$, since they are not preceded by $\circ$, no one of them can be filtered to $\Gamma_{i+1}^0$. Additionally, $\Psi_i^3$ is empty for all $i \geq 1$. To realize this fact, it suffices to check the following three facts. First, whenever the rule $(\mathcal{U}\,Set)$ is applied to the set $\Gamma_{i-1}^0$, by considering the partition $\Phi_{i-1} \cup (\Gamma_{i-1}^0 \upharpoonright \mathsf{sel\_ev\_set}_{i-1})$, the new literal $\circ(a_i \mathcal{U} P)$ appears only in now-clauses. Second, the remaining basic rules (resolution, subsumption and fixpoint rules), that are applied to obtain the TRS-closed set $\Gamma_{i-1}^*$ from $\Gamma_{i-1}^1$, cannot introduce (in $\Gamma_{i-1}^*$) an always-clause $C$ such that $\circ(a_i \mathcal{U} P) \in \mathsf{Lits}(C)$. Third, since $\Gamma_i^0$ is obtained from $\Gamma_{i-1}^*$ by $\mathsf{unnext}$, then $\Gamma_i^0$ cannot include an always-clause $C$ such that $\circ(a_i \mathcal{U} P) \in \mathsf{Lits}(C)$.
Consequently, every fresh variable $a_\ell$ is not in $\mathsf{Lits}(\mathsf{now}(\Gamma_h^0))$ for all $h \geq \ell$ and all $\ell \geq 1$. Therefore, fresh variables do not occur in any context of any application of the rule $(\mathcal{U}\,Set)$. So that, the successive contexts are exclusively formed by formulas from the closure of $\Gamma_0^0$.
Since the set $\mathsf{closure}(\Gamma_0^0)$ is finite, if the sequence of descendants of $P_0 \mathcal{U} P$ were infinite, there would necessarily be two sets $\Gamma_g^0$ and $\Gamma_h^0$ such that $g < h$ and $\mathsf{now}(\Gamma_g^0 \setminus \Gamma_g^0 \upharpoonright \mathsf{sel\_ev\_set}_g) = \mathsf{now}(\Gamma_h^0 \setminus \Gamma_h^0 \upharpoonright \{a_h \mathcal{U} P\})$[8]. Without loss of generality, we consider $g = 0$ and $h = i$. By repeatedly applying the rule $(Res)$ to $\mathsf{now}(\Gamma_0^0 \setminus \Gamma_0^0 \upharpoonright \{P_0 \mathcal{U} P\})$ and $\mathsf{CNF}(\Box(\neg a_1 \vee \neg \mathsf{now}(\Gamma_0 \setminus \Gamma_0 \upharpoonright \{P_0 \mathcal{U} P\})))$, the algorithm $\mathcal{SR}$ obtains $\neg a_1$ which resolves with $\Box(\neg a_2 \vee a_1)$ producing $\neg a_2$. Then $\neg a_2$ resolves with $\Box(\neg a_3 \vee a_2)$. At the end of this process $\neg a_{i-1}$ resolves with $\Box(\neg a_i \vee a_{i-1})$ producing $\neg a_i$. This literal resolves with every clause in $\{P \vee a_i \vee N_i \mid (a_i \mathcal{U} P) \vee N_i \in \Gamma_i\}$ producing the clauses in $\{P \vee N_i \mid (a_i \mathcal{U} P) \vee N_i \in \Gamma_i\}$ which subsume the clauses in $\{P \vee \circ(a_{i+1} \mathcal{U} P) \vee N_i \mid (a_i \mathcal{U} P) \vee N_i \in \Gamma_i\}$. Therefore, the selected temporal literal $a_{i+1} \mathcal{U} P$ disappears after the following unnext-step. Hence, $a_{i+1} \mathcal{U} P$ cannot be the selected eventuality at the next step, i.e., $\mathsf{sel\_ev\_set}_{i+1} \neq \{a_{i+1} \mathcal{U} P\}$. This is a contradiction because the sequence of descendants of $P_0 \mathcal{U} P$ has been supposed to be infinite. ■

In the above proof we have considered that $(\mathcal{U}\,Set)$ is always applied with a non-empty context. The proof for possibly empty contexts is just a especial case. Note also that the application of the subsumption rule, together with the subsequent use of the operator $\mathsf{unnext}$, is essential in the above proof.

**Theorem 4.6.14.** *The algorithm $\mathcal{SR}$, for each input $\Gamma$, terminates giving a resolution proof.*

*Proof.* Suppose that $\mathcal{SR}$ does not produce $\Box^b\bot$. On one hand, by Lemma 4.6.13, $\mathcal{SR}$ cannot generate an infinite sequence of descendants of any selected eventuality. Besides, when the sequence of descendants of one eventuality finishes because the last one, namely $T$, ceases to be

[8] $\mathsf{sel\_ev\_set}_g = \{P_0 \mathcal{U} P\}$ if $g = 0$, and $\mathsf{sel\_ev\_set}_g = \{a_g \mathcal{U} P\}$ if $g > 0$.

the selected eventuality in $\Gamma_i$ for some $i \geq 1$ (i.e. $\mathsf{sel\_ev\_set}^*_{i-1} = \{T\}$ and $\mathsf{sel\_ev\_set}_i \neq \{T\}$), then the set $\mathsf{now}(\Gamma_i)$ is included in $\mathsf{closure}(\Gamma)$ because the fresh variables introduced by $(\mathcal{U}\,Set)$ only occur in $\mathsf{alw}(\Gamma_i)$. If the process continues and the algorithm $\mathcal{SR}$ selects another eventuality, finiteness of sequences of descendants (Lemma 4.6.13) guarantees the existence of $\Gamma_g$, with $g > i$, such that $\mathsf{now}(\Gamma_g)$ is included in $\mathsf{closure}(\Gamma)$. As the closure is finite, there must exist $j$ and $k$ such that $j \leq k$ and the set of now-clauses of $\Gamma_j$ is exactly the set of now-clauses of $\mathsf{unnext}(\Gamma^*_k)$.
On the other hand, fairness ensures that the third condition in Definition 4.6.1 must be satisfied at some moment. ∎

### 4.6.4 Complexity

In order to analyze the worst case complexity of the algorithm $\mathcal{SR}$, we first consider the set $\mathsf{closure}(\Gamma)$ (see Definition 4.6.11) of all the possible clauses formed using the literals in $\mathsf{univlit}(\Gamma)$ (see Definition 4.6.10).

**Proposition 4.6.15.** *The number of clauses in* $\mathsf{closure}(\Gamma)$ *is* $2^n$*, where* $n$ *is the number of literals in* $\mathsf{univlit}(\Gamma)$. ∎

Then, the set of all possible sets of clauses that could appear as context when applying $(\mathcal{U}\,Set)$ has double-exponential size in $n$.

**Proposition 4.6.16.** *Let* $\mathsf{contexts}(\Gamma) = \{\Delta \mid \Delta \subseteq \mathsf{closure}(\Gamma)\}$*, then the number of sets in* $\mathsf{contexts}(\Gamma)$ *is* $2^{2^n}$. ∎

Therefore, the worst case complexity of the algorithm $\mathcal{SR}$ can be bounded to $\mathcal{O}(2^{\mathcal{O}(2^n)})$.

**Proposition 4.6.17.** *The number of clauses generated by the resolution method is bounded by* $\mathcal{O}(2^{\mathcal{O}(2^n)})$ *and the number of new variables is also bounded by* $\mathcal{O}(2^{\mathcal{O}(2^n)})$ *where* $n$ *is the number of literals in* $\mathsf{univlit}(\Gamma)$.

*Proof.* In the worst case, each clause in $\mathsf{closure}(\Gamma)$ contains a selected eventuality that generates a sequence of descendants with an eventuality for each possible context in $\mathsf{contexts}(\Gamma)$ plus a repeated context. That is, each of the $2^n$ initial clauses may generate $1 + 2^{2^n}$ clauses with new eventualities. So, $f(n) = 2^n \times (1 + 2^{2^n}) = 2^n + 2^{n+2^n}$ is the maximum number of different clauses (with new eventualities) that can appear in a derivation. Since, each new eventuality is associated to a new variable, $2^n + 2^{n+2^n}$ also bounds the number of fresh variables. In the worst case, the definition of each new variable generates $2^n$ new clauses. So that, $g(n) = 2^{2.n} + 2^{2.n+2^n}$ bounds the number of clauses defining new variables. To sum up, the worst case is bounded to

$$2^n + f(n) + g(n) = 2^n + 2^n + 2^{n+2^n} + 2^{2.n} + 2^{2.n+2^n}$$

where the leftmost $2^n$ stands for the size of the closure which bounds the initial set of clauses. That is, in the worst case, the number of clauses is in $\mathcal{O}(2^{\mathcal{O}(2^n)})$ and the number of new variables is in $\mathcal{O}(2^{\mathcal{O}(2^n)})$ . ∎

## *4.7 Completeness*

A resolution method is *refutationally complete* if, whenever a set of clauses $\Gamma$ is unsatisfiable, a refutation for $\Gamma$ can be constructed. In our case we prove the refutational completeness of TRS-resolution showing that there exists a model of $\Gamma$ whenever the resolution proof $\mathcal{D}(\Gamma)$ obtained by the algorithm $\mathcal{SR}$ is a cycling derivation. This result together with the proof of termination (Theorem 4.6.14) shows that our algorithm for systematic resolution (Figure 4.10) is complete and, hence, a decision procedure for PLTL.

For the rest of this section we fix the derivation

$$\mathcal{D}(\Gamma) \;\equiv\; (\Gamma_0, \Gamma_0^*) \mapsto (\Gamma_1, \Gamma_1^*) \mapsto \ldots \mapsto (\Gamma_j, \Gamma_j^*) \mapsto \ldots \mapsto (\Gamma_k, \Gamma_k^*)$$

to be cycling with respect to $j$ and $k$. In order to prove the existence of a model of $\Gamma$ from the existence of $\mathcal{D}(\Gamma)$ we will show that the sets $\Gamma_i^*$ in $\mathcal{D}(\Gamma)$ can be extended (with literals of their own clauses) preserving its local consistency. These extensions, denoted as $\widehat{\Gamma_i^*}$, are literal-closed in the sense that they contain at least one literal from each clause in $\Gamma_i^*$. Remember that the sets $\Gamma_i^*$ in $\mathcal{D}(\Gamma)$ are TRS-closed (see Definition 4.4.6) which, in particular, means that $\mathsf{BTL}(\Gamma_i^*) = \emptyset$ (Definition 4.2.3). Actually, inside the collection of all the locally consistent literal-closed (lclc, in short) extensions of each $\Gamma_i^*$, we define the subclass of the so-called *standard extensions*. In particular, standard lclc-extensions of the sets $\Gamma_i^*$ in $\mathcal{D}(\Gamma)$ allow us to ensure the model existence. We define a *successor relation* on lclc-extensions of the sets $\Gamma_i^*$ that gives rise to infinite paths of standard lclc-extensions. These infinite paths can be used to characterize or define PLTL-structures. Finally we show that at least one of those paths satisfies the suitable conditions for defining a model of $\Gamma$. Hence, this section is divided into a first subsection devoted to the notion of lclc-extensions of sets of clauses and their main properties, including the existence of a non-empty subclass of standard lclc-extensions for any locally consistent and TRS-closed set of clauses. In the second subsection, we define the notion of successor and prove the existence of infinite paths. Lastly, in the third subsection, we prove the existence of a model of $\Gamma$.

### *4.7.1 Extending Locally Consistent* TRS*-Closed Sets of Clauses*

In this subsection we show that every TRS-closed set of clauses has at least one locally consistent extension that is literal-closed and standard. We gradually define the notions and prove the results.

**Definition 4.7.1.** *A set of clauses* $\Gamma$ *is* literal-closed *iff* $\Gamma \cap \mathsf{Lits}(C) \neq \emptyset$ *for every* $C \in \Gamma$.[9] *Besides,* $\mathsf{lclc}(\Gamma)$ *denotes the collection of all locally consistent sets of clauses* $\widehat{\Gamma}$ *such that* $\Gamma \subseteq \widehat{\Gamma} \subseteq \Gamma \cup \mathsf{Lits}(\Gamma)$ *and* $\widehat{\Gamma}$ *is literal-closed. We say that each* $\widehat{\Gamma} \in \mathsf{lclc}(\Gamma)$ *is an* lclc-extension *of* $\Gamma$.

Note that if $\Box^b\bot$ is in $\Gamma$ then $\mathsf{lclc}(\Gamma) = \emptyset$ by local inconsistency. Besides, since only literals included in some clause in $\Gamma$ are used to build the elements in $\mathsf{lclc}(\Gamma)$, if no clause in $\Gamma$ includes any (basic) temporal literal (i.e. $\mathsf{BTL}(\Gamma) = \emptyset$, see Subsection 4.2.1) then every $\widehat{\Gamma} \in \mathsf{lclc}(\Gamma)$ also satisfies that $\mathsf{BTL}(\widehat{\Gamma}) = \emptyset$. In particular, if $\Gamma = \emptyset$ then $\mathsf{lclc}(\Gamma) = \{\emptyset\}$.

Next, we show that for every locally consistent set of clauses $\Gamma$ that does not contain (basic) temporal literals there exists at least one lclc-extension of $\Gamma$.

**Proposition 4.7.2.** *If* $\Gamma$ *is a locally consistent set of clauses such that* $\mathsf{BTL}(\Gamma) = \emptyset$ *then* $\mathsf{lclc}(\Gamma) \neq \emptyset$.

[9] Note that literals in $\mathsf{Lits}(C)$ are viewed as singleton clauses.

*Proof.* We will show that there exists a sequence $S = \Omega_0, \Omega_1, \Omega_2, \dots, \Omega_g$ such that $g \geq 0$, $\Omega_0 = \Gamma$ and $\Omega_{h+1} = \Omega_h \cup \{L\}$ (for every $h \in \{0, \dots, g-1\}$) for some $L \in \mathsf{Lits}(C)$ and some $C \in \Omega_h$ such that $\mathsf{Lits}(C) \cap \Omega_h = \emptyset$ and $\Omega_h \cup \{L\}$ is locally consistent. In addition, $\Omega_g \in \mathsf{lclc}(\Gamma)$ whereas $\Omega_h \notin \mathsf{lclc}(\Gamma)$ for all $h \in \{0, \dots, g-1\}$. Since the number of clauses is finite, this inductive construction is also finite and shows that $\mathsf{lclc}(\Gamma) \neq \emptyset$.

We have to show that, for every $h$ such that $\Omega_h \notin \mathsf{lclc}(\Gamma)$, there exists a locally consistent $\Omega_{h+1}$ that extends $\Omega_h$ with a new literal from some clause in $\Gamma$. Since $\Omega_h \notin \mathsf{lclc}(\Gamma)$ there exists (at least one) clause $C = \Box^b(L_1 \vee \ldots \vee L_n) \in \Omega_h$ such that $L_i \notin \Omega_h$ for all $i \in \{1, \dots, n\}$. Suppose that $\Omega_h \cup \{L_i\}$ is not locally consistent for all $i \in \{1, \dots, n\}$. Then, by Proposition 4.4.14, there exists a local refutation $\mathcal{D}_i$ for $\Omega_h \cup \{L_i\}$ that is linear with respect to $L_i$, for every $i \in \{1, \dots, n\}$. From these $n$ local refutations we are able to construct a local refutation $\mathcal{D}$ for $\Omega_h$ that is linear with respect to $C$, contradicting the assumption that $\Omega_h$ is locally consistent. Hence, $\Omega_h \cup \{L_i\}$ must be locally consistent for some $i \in \{1, \dots, n\}$. ■

**Definition 4.7.3.** *Let* $\Gamma$ *be a set of clauses such that* $\mathsf{lclc}(\Gamma) \neq \emptyset$ *and let* $\Lambda \subseteq \mathsf{Lits}(\Gamma)$*. We say that* $\Lambda$ represents $\Gamma$ *if* $\widehat{\Gamma} \cap \Lambda \neq \emptyset$ *for all* $\widehat{\Gamma} \in \mathsf{lclc}(\Gamma)$*. If, in addition, for every* $\Lambda' \subsetneq \Lambda$ *there exists* $\widehat{\Gamma} \in \mathsf{lclc}(\Gamma)$ *such that* $\widehat{\Gamma} \cap \Lambda' = \emptyset$*, then we say that* $\Lambda$ minimally represents $\Gamma$.

The following result shows that the minimal representatives of a TRS-closed set of clauses $\Gamma$ are included (as clauses) in $\Gamma$.

**Proposition 4.7.4.** *For every* $\Lambda$ *that minimally represents a non-empty locally consistent* TRS*-closed set of clauses* $\Gamma$ *there is a clause* $C \in \Gamma$ *such that* $\mathsf{Lits}(C) = \Lambda$.

*Proof.* First we will show that $\Gamma$ must contain at least one clause $C$ such that $\mathsf{Lits}(C) \subseteq \Lambda$. We partition $\Gamma$ into the following two sets:

$$\Pi_1 = \{C \in \Gamma \mid \mathsf{Lits}(C) \cap \Lambda = \emptyset\}$$
$$\Pi_2 = \{C \in \Gamma \mid \mathsf{Lits}(C) \cap \Lambda \neq \emptyset\}$$

We split the clauses in $\Pi_2$ into the sub-clauses formed by literals that do not appear in $\Lambda$ and the sub-clauses formed by literals that appear in $\Lambda$. These sets of clauses respectively are the following sets $\Sigma_1$ and $\Sigma_2$.

$$\Sigma_1 = \{N \mid \Box^b(N \vee N') \in \Pi_2, \mathsf{Lits}(N) \cap \Lambda = \emptyset \text{ and } \mathsf{Lits}(N') \subseteq \Lambda\}$$
$$\Sigma_2 = \{N' \mid \Box^b(N \vee N') \in \Pi_2, \mathsf{Lits}(N) \cap \Lambda = \emptyset \text{ and } \mathsf{Lits}(N') \subseteq \Lambda\}$$

Since $\Gamma$ is locally consistent, $\Pi_1$, $\Pi_2$ and also their proper subsets are locally consistent. In addition, $\Gamma$ is TRS-closed, hence $\mathsf{BTL}(\Gamma) = \emptyset$ and every set of clauses considered along the rest of this proof does not contain any clause that includes any (basic) temporal literal.

Now we show, by contradiction, that $\bot \in \Pi_1 \cup \Sigma_1$ and, since $\Pi_1$ is locally consistent, it follows that $\bot \in \Sigma_1$ and, consequently, there exists a clause $C \in \Gamma$ such that $\mathsf{Lits}(C) \subseteq \mathsf{Lits}(\Sigma_2)$, i.e., $\mathsf{Lits}(C) \subseteq \Lambda$.

Let us suppose that $\bot \notin \Pi_1 \cup \Sigma_1$. First, suppose that $\Pi_1 \cup \Sigma_1$ is locally consistent. By Proposition 4.7.2, the set $\mathsf{lclc}(\Pi_1 \cup \Sigma_1)$ is non-empty and for every $\Psi \in \mathsf{lclc}(\Pi_1 \cup \Sigma_1)$ the set $\Omega = \Gamma \cup \{L \mid L \in \Psi\}$ is in $\mathsf{lclc}(\Gamma)$ and satisfies $\Omega \cap \Lambda = \emptyset$. This contradicts that $\Lambda$ minimally represents $\Gamma$.

Second, suppose that $\Pi_1 \cup \Sigma_1$ is locally inconsistent, there exists some minimal locally inconsistent subset $\Phi$ of $\Pi_1 \cup \Sigma_1$ (i.e. $\Phi$ does not contain locally inconsistent proper subsets of $\Pi_1 \cup \Sigma_1$). Since every subset of $\Pi_1$ is locally consistent, then $\Phi \cap \Sigma_1 \neq \emptyset$. Let $N$ be any clause in $\Phi \cap \Sigma_1$. By Proposition 4.4.14, there exists a local refutation $\mathcal{D}$ for $\Phi$ that is linear with respect to $N$. By using the original clauses in $\Pi_2$ instead of their sub-clauses in $\Phi \cap \Sigma_1$, we can build from $\mathcal{D}$ a derivation $\mathcal{D}'$ whose last set contains a clause $C$ such that $\mathsf{Lits}(C) \subseteq \mathsf{Lits}(\Sigma_2)$. Hence, $\bot \in \Sigma_1$ and this contradicts that $\bot \notin \Pi_1 \cup \Sigma_1$.

So, since considering $\bot \notin \Pi_1 \cup \Sigma_1$ leads to a contradiction when we consider that $\Pi_1 \cup \Sigma_1$ is locally consistent and when we consider that $\Pi_1 \cup \Sigma_1$ is locally inconsistent, it follows that $\bot \in \Pi_1 \cup \Sigma_1$. Therefore $\bot \in \Sigma_1$ because $\Pi_1$ is locally consistent and, consequently, there are a clause $C \in \Gamma$ such that $\mathsf{Lits}(C) \subseteq \Lambda$.

Finally, $\mathsf{Lits}(C)$ cannot be a proper subset of $\Lambda$ because $\mathsf{Lits}(C)$ also represents $\Gamma$ and that would contradict the minimality of the representation of $\Gamma$ by $\Lambda$ (see Definition 4.7.3). Henceforth, $\mathsf{Lits}(C) = \Lambda$. ■

Next we introduce the notion of *standard* lclc-extensions of a set of clauses.

**Definition 4.7.5.** *Let $\Gamma$ be a locally consistent* TRS*-closed set of clauses. We say that $\widehat{\Gamma} \in \mathsf{lclc}(\Gamma)$ is* standard *iff it satisfies the following conditions:*

*(a) If $\circ L \in \widehat{\Gamma}$, then there exists a clause $\Box^b(\circ L \vee \circ N) \in \Gamma$*

*(b) For every propositional literal $P \in \mathsf{Lits}(\Gamma)$, if $\widehat{\Gamma} \cup \{P\}$ is locally consistent, then $P \in \widehat{\Gamma}$.*

*(c) If $\circ L \in \widehat{\Gamma}$, then $\Gamma \cup (\widehat{\Gamma} \setminus \{\circ L\})$ is not literal-closed.*

The following lemma ensures the existence of at least one standard lclc-extension of any locally consistent TRS-closed set of clauses.

**Lemma 4.7.6.** *Let $\Gamma$ be a locally consistent* TRS*-closed set of clauses. There exists at least one standard set in $\mathsf{lclc}(\Gamma)$.*

*Proof.* We first prove that there exists $\Omega \in \mathsf{lclc}(\Gamma)$ that satisfies item *(a)* in Definition 4.7.5. Second, we show that there exists $\Sigma \supseteq \Omega$ such that $\Sigma \in \mathsf{lclc}(\Gamma)$ and satisfies *(a)* and *(b)* in Definition 4.7.5. Third, we show that there exists $\Delta \subseteq \Sigma$ such that $\Delta \in \mathsf{lclc}(\Gamma)$ and satisfies *(a)*, *(b)* and *(c)* in Definition 4.7.5.

1. By Proposition 4.7.2, $\mathsf{lclc}(\Gamma)$ is non-empty. Now, let us suppose that for every set in $\mathsf{lclc}(\Gamma)$ there exists a literal of the form $\circ L$ such that $\circ L \notin \mathsf{Lits}(\Box^b \circ N)$ for every clause $\Box^b \circ N \in \Gamma$. Then, for every $\widehat{\Gamma} \in \mathsf{lclc}(\Gamma)$, there exists some $L \in \widehat{\Gamma}$ that belongs to the following set

$$\Psi = \{\circ L \in \mathsf{Lits}(\Gamma) \mid \circ L \notin \mathsf{Lits}(\Box^b \circ N) \text{ for every clause } \Box^b \circ N \in \Gamma\}$$

   Hence $\Psi$ represents $\Gamma$ and there should exist some $\Lambda \subseteq \Psi$ that minimally represents $\Gamma$. Therefore, by Proposition 4.7.4, there exists a clause $C \in \Gamma$ such that $\mathsf{Lits}(C) = \Lambda$. This is a contradiction because the literals in $\Psi$, and in particular the literals in $\Lambda$, do not belong to any clause of the form $\Box^b \circ N$ in $\Gamma$. Therefore, there exists some set $\Omega$ in $\mathsf{lclc}(\Gamma)$ that satisfies Definition 4.7.5*(a)*.

2. Since $\Omega$ is locally consistent and $\mathsf{BTL}(\Omega) = \emptyset$, the sequence $\Omega_0, \Omega_1, \Omega_2, \ldots, \Omega_g$ in the proof of Proposition 4.7.2 is easily adapted for ensuring that each $\Omega_i$ satisfies Definition 4.7.5*(a)* and that $\Omega_g$ satisfies Definition 4.7.5*(b)*. So that $\Sigma = \Omega_g$.

3. We show that $\Sigma$ should contain a subset $\Delta$ that satisfies the lemma. Since $\Sigma$ belongs to $\mathsf{lclc}(\Gamma)$, verifies Definition 4.7.5*(a)* and *(b)* and is a finite set, we can ensure the existence of a finite sequence $\Sigma_0, \Sigma_1, \Sigma_2, \ldots, \Sigma_r$ such that $r \geq 0$, $\Sigma_0 = \Sigma$, $\Sigma_r \setminus \{\circ L\} \notin \mathsf{lclc}(\Gamma)$ for all $\circ L \in \Sigma_r$, and $\Sigma_{h+1} = \Sigma_h \setminus \{\circ L_h\}$ for some $\circ L_h \in \Sigma_h$ and $\Sigma_{h+1} \in \mathsf{lclc}(\Gamma)$ for every $h \in \{0, \ldots, r-1\}$. Therefore, $\Sigma_h$ satisfies Definition 4.7.5*(a)* and *(b)* for all $h \in \{0, \ldots, r\}$ and $\Sigma_r$ additionally satisfies *(c)*. Hence, $\Sigma_r$ is the set $\Delta$ we were looking for. ∎

For locally consistent TRS-closed sets, the subclass of their standard lclc-extensions represents the whole class of their lclc-extensions with respect to sets of next-literals in the sense shown by the following proposition.

**Proposition 4.7.7.** *Let $\Gamma$ be any locally consistent* TRS*-closed set of clauses and $\Lambda \subseteq \mathsf{Lits}(\Gamma)$ be a set such that every literal in $\Lambda$ is of the form $\circ L$. If $\widehat{\Gamma} \cap \Lambda \neq \emptyset$ for every standard set $\widehat{\Gamma} \in \mathsf{lclc}(\Gamma)$, then $\Lambda$ represents $\Gamma$.*

*Proof.* Consider any $\Lambda$ that satisfies the hypothesis but does not represent $\Gamma$. Hence, there exists some non-standard set $\Psi \in \mathsf{lclc}(\Gamma)$ such that $\Psi \cap \Lambda = \emptyset$. Now, let

$$\Pi = \{N \mid \Box^b(N \vee N') \in \Gamma, \mathsf{Lits}(N) \cap \Lambda = \emptyset \text{ and } \mathsf{Lits}(N') \subseteq \Lambda\}$$
$$\Phi = \{N \in \Pi \mid \text{ no clause in } \Pi \text{ subsumes } N\}$$

Then, $\Phi$ is TRS-closed and locally consistent. The former holds because $\Gamma$ is TRS-closed. For the latter suppose that $\Phi$ is not locally consistent. By Proposition 4.4.10, $\bot \in \Phi$. Hence, by definition of $\Phi$, there exists a clause $C \in \Gamma$ such that $\mathsf{Lits}(C) \subseteq \Lambda$. But this contradicts the assumption $\Psi \cap \Lambda = \emptyset$ because $\Psi$ is an lclc-extension of $\Gamma$ and, consequently, $\mathsf{Lits}(C) \cap \Psi$ cannot be empty.

Since $\Phi$ is TRS-closed and locally consistent, by Lemma 4.7.6, there is some $\Omega \in \mathsf{lclc}(\Phi)$ that is standard. Hence, consider $\Sigma = \Gamma \cup \{L \mid L \in \Omega\}$ for some standard $\Omega \in \mathsf{lclc}(\Phi)$. First, $\Sigma$ is an lclc-extension of $\Gamma$ because $\mathsf{Lits}(\Omega) \subseteq \mathsf{Lits}(\Gamma)$ and because for every clause $C \in \Gamma$ there exists a clause $N \in \Phi$ such that $\mathsf{Lits}(N) \subseteq \mathsf{Lits}(C)$. Second, $\Sigma$ is standard because $\Omega$ is a standard lclc-extension of $\Phi$ and $\Lambda$ contains only literals of the form $\circ L$, so that $\Sigma$ satisfies Definition 4.7.5. Consequently, $\Sigma$ is a standard lclc-extension of $\Gamma$ such that $\Sigma \cap \Lambda = \emptyset$. This contradicts that $\widehat{\Gamma} \cap \Lambda \neq \emptyset$ for all standard $\widehat{\Gamma} \in \mathsf{lclc}(\Gamma)$. Therefore, $\Lambda$ represents $\Gamma$. ∎

### *4.7.2 Building Infinite Paths of Standard Lclc-Extensions*

In order to build sequences of standard lclc-extensions of the TRS-closed sets $\Gamma_i^*$ –in the cycling derivation $\mathcal{D}(\Gamma)$– that represent models of $\Gamma$, such sequences must be coherent with respect to the meaning of temporal connectives. We mean that, e.g. if $\circ p$ belongs to a set $\Omega$ in the sequence, then $p$ must belong to the set that is the successor of $\Omega$ in the sequence. Similarly, for eventualities where also the selections performed along $\mathcal{D}(\Gamma)$ are relevant. As a consequence a successor relation is defined for the lclc-extensions of the TRS-closed sets that appear in the derivation $\mathcal{D}(\Gamma)$:

$$(\Gamma_0, \Gamma_0^*) \mapsto (\Gamma_1, \Gamma_1^*) \mapsto \ldots \mapsto (\Gamma_j, \Gamma_j^*) \mapsto \ldots \mapsto (\Gamma_k, \Gamma_k^*)$$

which is cycling with respect to $j$ and $k$. This successor relation on

$$\{\mathsf{lclc}(\Gamma_i^*) \times \mathsf{lclc}(\Gamma_{i+1}^*) \mid 0 \leq i < k\} \cup (\mathsf{lclc}(\Gamma_k^*) \times \mathsf{lclc}(\Gamma_j^*))$$

is presented in Definition 4.7.8. Along the rest of this chapter, $\widehat{\Gamma_i^*}$ denotes a member of $\mathsf{lclc}(\Gamma_i^*)$.

**Definition 4.7.8.** *Let $i = h+1$ if $h \in \{0,\dots,k-1\}$ and let $i = j$ if $h = k$, we say that $\widehat{\Gamma_i^*}$ is a* successor *of $\widehat{\Gamma_h^*}$ or that $\widehat{\Gamma_h^*}$ is a* predecessor *of $\widehat{\Gamma_i^*}$ if for every $\circ L \in \widehat{\Gamma_h^*}$ there is some $S \in \mathsf{nxclo}_i(\circ L)$ such that $S \subseteq \widehat{\Gamma_i^*}$, where $\mathsf{nxclo}_i$ is defined as follows*

- $\mathsf{nxclo}_i(\circ P) = \{\{P\}\}$ *where $P$ is a propositional literal.*

- $\mathsf{nxclo}_i(\circ\circ L) = \{\{\circ L\}\}$

- $\mathsf{nxclo}_i(\circ(P_1 \,\mathcal{U}\, P_2)) = \begin{cases} \{\{P_2\},\{P_1,\circ(P_1 \,\mathcal{U}\, P_2)\}\} & \text{if } P_1 \,\mathcal{U}\, P_2 \notin \mathsf{sel\_ev\_set}_i \\ \{\{P_2\},\{P_1,\circ(a \,\mathcal{U}\, P_2)\}\} & \text{otherwise} \\ \text{where } a \,\mathcal{U}\, P_2 \in \mathsf{sel\_ev\_set}_i^* & \end{cases}$

- $\mathsf{nxclo}_i(\circ\Diamond P) = \begin{cases} \{\{P\},\{\circ\Diamond P\}\} & \text{if } \Diamond P \notin \mathsf{sel\_ev\_set}_i \\ \{\{P\},\{\circ(a \,\mathcal{U}\, P)\}\} & \text{otherwise} \\ \text{where } a \,\mathcal{U}\, P \in \mathsf{sel\_ev\_set}_i^* & \end{cases}$

- $\mathsf{nxclo}_i(\circ(P_1 \,\mathcal{R}\, P_2)) = \{\{P_2, P_1\},\{P_2,\circ(P_1 \,\mathcal{R}\, P_2)\}\}$

- $\mathsf{nxclo}_i(\circ\Box P) = \{\{P,\Box P\},\{P,\circ\Box P\}\}$.

*The set of successors of a given set $\widehat{\Gamma_h^*}$ is denoted by $\mathsf{succ}(\widehat{\Gamma_h^*})$.*

The definition of $\mathsf{nxclo}_i(\circ\Box P)$ arises from the fact that the literal $\circ\Box P$ can be either a singleton now-clause or a literal properly contained in a clause $C$. In the first case, $\Gamma_i$ contains the always-clause $\Box P$ which will not be affected by the rule $(\Box Fix)$. Consequently, in such a case $\Gamma_i^*$ contains necessarily $\Box P$. However, in the second case, the literal $\circ\Box P$ is introduced by application of the rule $(\Box Fix)$ to the clause $C$.

The existence of infinite paths of standard lclc-extensions is based on the existence of a predecessor for each standard lclc-extension of a TRS-closed set in the derivation which is a standard lclc-extension of the previous TRS-closed set in the derivation.

**Proposition 4.7.9.** *For every $i \in \{1,\dots,k\}$ and every standard $\widehat{\Gamma_i^*} \in \mathsf{lclc}(\Gamma_i^*)$, there exists a standard $\widehat{\Gamma_{i-1}^*} \in \mathsf{lclc}(\Gamma_{i-1}^*)$ such that $\widehat{\Gamma_i^*} \in \mathsf{succ}(\widehat{\Gamma_{i-1}^*})$.*

*Proof.* Let $W_\ell = \{\widehat{\Gamma_\ell^*} \in \mathsf{lclc}(\Gamma_\ell^*) \mid \widehat{\Gamma_\ell^*}$ is standard $\}$ for each $\ell \in \{0,\dots,k\}$. If there exists some $\widehat{\Gamma_{i-1}^*} \in W_{i-1}$ such that $\widehat{\Gamma_{i-1}^*}$ does not contain any clause of the form $\circ L$, then $\widehat{\Gamma_i^*} \in \mathsf{succ}(\widehat{\Gamma_{i-1}^*})$ for all $\widehat{\Gamma_i^*}$. Otherwise, every set $\widehat{\Gamma_{i-1}^*} \in W_{i-1}$ contains at least one clause of the form $\circ L$. We proceed by contradiction. Let us suppose that $\widehat{\Gamma_i^*}$ is a member of $W_i$ such that $\widehat{\Gamma_i^*} \notin \mathsf{succ}(\widehat{\Gamma_{i-1}^*})$ for all $\widehat{\Gamma_{i-1}^*} \in W_{i-1}$. Hence, there exists at least one $\circ L$ in every $\widehat{\Gamma_{i-1}^*} \in W_{i-1}$ such that $S \not\subseteq \widehat{\Gamma_i^*}$ for all $S \in \mathsf{nxclo}_i(\circ L)$. Therefore, the set

$$\Lambda = \{\circ L \mid \circ L \in \textstyle\bigcup_{\widehat{\Gamma_{i-1}^*} \in W_{i-1}} \widehat{\Gamma_{i-1}^*} \text{ such that } S \not\subseteq \widehat{\Gamma_i^*} \text{ for all } S \in \mathsf{nxclo}_i(\circ L)\}$$

satisfies that $\Lambda \cap \widehat{\Gamma^*_{i-1}} \neq \emptyset$ for all $\widehat{\Gamma^*_{i-1}} \in W_{i-1}$. Therefore, by Proposition 4.7.7, $\Lambda$ represents $\Gamma^*_{i-1}$ and, consequently there exists some set $\Omega \subseteq \Lambda$ that minimally represents $\Gamma^*_{i-1}$. By Proposition 4.7.4, there exists a clause $C = \Box^b(\circ L_1 \vee \ldots \vee \circ L_r)$ in $\Gamma^*_{i-1}$ such that $\mathsf{Lits}(C) = \Omega$ and $r \geq 1$. Since $\mathsf{unnext}(\{C\}) \subseteq \Gamma_i$, then the clause $C' = L_1 \vee \ldots \vee L_r$ is in $\Gamma_i$. Now, let

$$\{S_1, \ldots, S_n\} = \bigcup_{g=1}^{r} \mathsf{nxclo}_i(\circ L_g)$$

(note that $n \geq 1$) and let $\{C_1, \ldots, C_m\}$ be the set of all clauses of the form $L_1 \vee \ldots \vee L_n$ such that $L_h \in S_h$ for all $h \in \{1, \ldots, n\}$. By subsumption, $\Gamma^*_i$ contains a non-empty set of (non-empty) clauses $\{D_1, \ldots, D_m\}$ such that $\mathsf{Lits}(D_t) \subseteq \mathsf{Lits}(C_t)$ for all $t \in \{1, \ldots, m\}$. By construction $S \not\subseteq \widehat{\Gamma^*_i}$ for all $S \in \mathsf{nxclo}_i(\circ L_g)$ and all $g \in \{1, \ldots, r\}$. Hence, for each pair $(g, S)$ such that $g \in \{1, \ldots, r\}$ and $S \in \mathsf{nxclo}_i(\circ L_g)$, we can choose at least one literal $L$ such that $L \in S$ and $L \notin \widehat{\Gamma^*_i}$. As a consequence, there exists a clause $D_t \in \Gamma^*_i$ with $t \in \{1, \ldots, m\}$ such that $\mathsf{Lits}(D_t) \subseteq \mathsf{Lits}(C_t)$ where $D_t \cap \widehat{\Gamma^*_i} = \emptyset$. This contradicts the fact that $\widehat{\Gamma^*_i}$ contains at least one literal from each clause in $\Gamma^*_i$. ■

**Proposition 4.7.10.** *For every $i \in \{1, \ldots, k\}$ and every standard $\widehat{\Gamma^*_i}$, there exists a sequence $\widehat{\Gamma^*_0}, \widehat{\Gamma^*_1}, \ldots, \widehat{\Gamma^*_i}$ of standard sets such that $\widehat{\Gamma^*_h} \in \mathsf{succ}(\widehat{\Gamma^*_{h-1}})$ for every $h \in \{1, \ldots, i\}$.*

*Proof.* By Lemma 4.7.6 and Proposition 4.7.9. ■

**Proposition 4.7.11.** *For every standard $\widehat{\Gamma^*_j}$ there exists at least one standard $\widehat{\Gamma^*_k}$ such that $\widehat{\Gamma^*_j} = \mathsf{succ}(\widehat{\Gamma^*_k})$.*

*Proof.* The proof is very similar to the one of Proposition 4.7.9, but using that $\mathsf{now}(\Gamma_j) = \mathsf{now}(\mathsf{unnext}(\Gamma^*_k))$ instead of $\Gamma_i = \mathsf{unnext}(\Gamma^*_{i-1})$ and also using the fact that the set $\{N \mid \Box \circ N \in \Gamma^*_k\}$ is contained into the set $\mathsf{now}(\mathsf{unnext}(\Gamma^*_k))$ (by definition of the operator unnext). ■

We construct pre-models of $\Gamma$ by means of sequences of standard lclc-extensions of the sets in $\mathcal{D}(\Gamma)$ which will be ordered by the successor relation. For that, we need some notation on such sequences. For $g$ and $h$, where $0 \leq g \leq h \leq k$, we denote by $\mathcal{D}(\Gamma)_{[g..h]}$, the set of all *intervals* of standard lclc-extensions $\widehat{\Gamma^*_g}, \widehat{\Gamma^*_{g+1}}, ..., \widehat{\Gamma^*_h}$ such that $\widehat{\Gamma^*_i} \in \mathsf{succ}(\widehat{\Gamma^*_{i-1}})$ for every $i \in \{g+1, \ldots, h\}$. The functions first and last respectively return the first and the last set of a given interval. We use superscripts notation to denote subsequences of an interval $s \in \mathcal{D}(\Gamma)_{[g..h]}$ as follows. For $n$ and $m$ such that $g \leq n \leq m \leq h$, the subsequence $s^{n..m}$ denotes the subsequence formed by the sets $\widehat{\Gamma^*_n}, \widehat{\Gamma^*_{n+1}}, \ldots, \widehat{\Gamma^*_m}$ of $s$. In particular, if $n = m$ we write $s^n$ instead of $s^{n..n}$ and intentionally confuse the sequence of one set with the set itself. For $s \in \mathcal{D}(\Gamma)_{[g..h]}$, we denote by $\mathsf{range}(s)$ the set of natural numbers $\{n \mid g \leq n \leq h\}$. Since $\mathcal{D}(\Gamma)$ is cycling with respect to $j$ and $k$, the two sets of intervals $\mathcal{D}(\Gamma)_{[0..j-1]}$ and $\mathcal{D}(\Gamma)_{[j..k]}$ are respectively called *initial* and *inner*. Note that, since $j$ could be 0, the set $\mathcal{D}(\Gamma)_{[0..j-1]}$ could be empty, but $\mathcal{D}(\Gamma)_{[j..k]}$ is non-empty for any $\mathcal{D}(\Gamma)$.

**Proposition 4.7.12.** *For each standard $\widehat{\Gamma^*_j}$ there exists $s \in \mathcal{D}(\Gamma)_{[j..k]}$ such that $\widehat{\Gamma^*_j} \in \mathsf{succ}(\mathsf{last}(s))$.*

*Proof.* By Propositions 4.7.10 and 4.7.11. ■

Note that in the above proposition $\widehat{\Gamma_j^*}$ and $\mathsf{first}(s)$ can be different.

Now, we define when a sequence of elements from $\mathcal{D}(\Gamma)_{[j..k]}$ forms a cycle, which is called a $\mathcal{D}(\Gamma)$-cycle. Then we prove that there exists at least one $\mathcal{D}(\Gamma)$-cycle.

**Definition 4.7.13.** *A $\mathcal{D}(\Gamma)$-cycle is a finite non-empty sequence $s_0, s_1, \ldots, s_n$ such that*

*(i)* $s_i \in \mathcal{D}(\Gamma)_{[j..k]}$ *for all* $i \in \{0, \ldots, n\}$

*(ii)* $\mathsf{first}(s_{i+1}) \in \mathsf{succ}(\mathsf{last}(s_i))$ *for all* $i \in \{0, \ldots, n-1\}$ *and*

*(iii)* $\mathsf{first}(s_0) \in \mathsf{succ}(\mathsf{last}(s_n))$.

**Proposition 4.7.14.** *There exists at least one $\mathcal{D}(\Gamma)$-cycle.*

*Proof.* By Lemma 4.7.6, there exists at least one standard set in $\mathsf{lclc}(\Gamma_j^*)$. Let us consider any standard $\widehat{\Gamma_j^*}$ in $\mathsf{lclc}(\Gamma_j^*)$. By Proposition 4.7.12, there exists an interval $r_0 \in \mathcal{D}(\Gamma)_{[j..k]}$ such that $\widehat{\Gamma_j^*} \in \mathsf{succ}(\mathsf{last}(r_0))$. Additionally, by repeatedly applying Proposition 4.7.12, we can build an infinite sequence of intervals $r_0, r_1, \ldots$ in $\mathcal{D}(\Gamma)_{[j..k]}$ such that $\mathsf{first}(r_{i-1}) \in \mathsf{succ}(\mathsf{last}(r_i))$ for every $i \geq 1$. Since $\mathcal{D}(\Gamma)_{[j..k]}$ is finite, $r_g = r_h$ must hold for some $g$ and $h$ such that $0 \leq g < h$. Then, the reverse of the sequence $r_g, \ldots, r_{h-1}$, i.e. the sequence $r_{h-1}, \ldots, r_g$ is a $\mathcal{D}(\Gamma)$-cycle. ■

Note that the minimal cycles consist of exactly one interval $s \in \mathcal{D}(\Gamma)_{[j..k]}$ such that $\mathsf{first}(s) \in \mathsf{succ}(\mathsf{last}(s))$.

### 4.7.3 Model Existence

In this subsection we prove that there exists at least one model of $\Gamma$ on the basis of the cycling derivation $\mathcal{D}(\Gamma)$. First, we define a graph structure $\mathcal{G}_{\mathcal{D}(\Gamma)}$ whose nodes are intervals in $\mathcal{D}(\Gamma)_{[0..j-1]}$ and $\mathcal{D}(\Gamma)_{[j..k]}$. There is a (directed) edge $(s, s')$ in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ whenever $\mathsf{first}(s') \in \mathsf{succ}(\mathsf{last}(s))$. Note that every node in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ is related to a node from $\mathcal{D}(\Gamma)_{[j..k]}$. Second, we define a notion of self-fulfilling path in this graph. Then, we prove that $\mathcal{G}_{\mathcal{D}(\Gamma)}$ contains at least one strongly connected component (a $\mathcal{D}(\Gamma)$-cycle) that is self-fulfilling. Finally, we define a model of $\Gamma$ on the basis of this strongly connected component in $\mathcal{G}_{\mathcal{D}(\Gamma)}$.

**Definition 4.7.15.** *We associate to $\mathcal{D}(\Gamma)$ the graph $\mathcal{G}_{\mathcal{D}(\Gamma)}$ that is formed by the following set of nodes $S_{\mathcal{D}(\Gamma)}$ and the following edge-relation $R_{\mathcal{D}(\Gamma)}$ on $S_{\mathcal{D}(\Gamma)}$:*

- $S_{\mathcal{D}(\Gamma)} = \mathcal{D}(\Gamma)_{[0..j-1]} \cup \mathcal{D}(\Gamma)_{[j..k]}$
- $sR_{\mathcal{D}(\Gamma)}s'$ *iff* $s' \in \mathcal{D}(\Gamma)_{[j..k]}$ *and* $\mathsf{first}(s') \in \mathsf{succ}(\mathsf{last}(s))$.

Paths and strongly connected components in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ are defined as usual in graph theory. The notion of $\mathcal{D}(\Gamma)$-cycle (see Definition 4.7.13) has an obvious extension to $\mathcal{G}_{\mathcal{D}(\Gamma)}$. Therefore, by Proposition 4.7.14, the graph $\mathcal{G}_{\mathcal{D}(\Gamma)}$ has at least one cycle. The minimal graphs $\mathcal{G}_{\mathcal{D}(\Gamma)}$ consist of exactly one node $n$ with one edge from $n$ to $n$.

We would like to remark that, from a locally consistent literal-closed set, interleaved unnext-steps and TRS-steps could yield a TRS-refutation. As a consequence, there could exist some interval $s$ in $S_{\mathcal{D}(\Gamma)}$ such that no $s' \in S_{\mathcal{D}(\Gamma)}$ satisfies $sR_{\mathcal{D}(\Gamma)}s'$ and, hence, there could exist lclc-extensions that do not belong to any interval in $S_{\mathcal{D}(\Gamma)}$.

The paths in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ are formed by standard lclc-extensions of TRS-closed sets which do not include any (basic) temporal literal. Consequently, any occurrence of an eventuality in the states of $\mathcal{G}_{\mathcal{D}(\Gamma)}$ must be preceded by a connective $\circ$. This fact leads us to define the following notion of eventuality fulfillment in the paths of $\mathcal{G}_{\mathcal{D}(\Gamma)}$.

**Definition 4.7.16.** *Let $\pi = s_0, s_1, \dots$ be a path in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ such that $\circ(P_1 \mathcal{U} P_2) \in s_g^i$ for some $g \geq 0$ and $i \in \mathsf{range}(s_g)$. We say that $\pi$* fulfills $\circ(P_1 \mathcal{U} P_2)$ *iff either*

- *there exists $h \in \mathsf{range}(s_g)$ such that $h > i$, $P_2 \in s_g^h$ and $P_1 \in s_g^\ell$ for all $\ell \in \{i+1, \dots, h-1\}$, or*
- *there exist $r > g$ and $h \in \mathsf{range}(s_r)$ such that $P_2 \in s_r^h$ and $P_1 \in s_z^\ell$ for all $(z, \ell)$ such that $g < z < r$ and $\ell \in \mathsf{range}(s_z)$ and $P_1 \in s_r^\ell$ for all $\ell \in \{j, \dots, h-1\}$ and $P_1 \in s_g^\ell$ for all $\ell \in \{i+1, \dots, m\}$ where $m$ is the maximum in $\mathsf{range}(s_g)$.*

*A path $\pi$ is self-fulfilling iff $\pi$ fulfills every $\circ(P_1 \mathcal{U} P_2)$ that occurs in any of its sets. Besides, a $\mathcal{D}(\Gamma)$-cycle $\sigma$ in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ is* self-fulfilling *if the path $\sigma^\omega$ is self-fulfilling.*

Since $\circ\diamond P$ and $\circ(\widetilde{P} \mathcal{U} P)$ are equivalent, the fulfillment notion for $\circ\diamond P$ is a particular case of Definition 4.7.16.

The next three propositions are auxiliary results about the fulfillment of eventualities, which are useful for proving the Lemma 4.7.20.

**Proposition 4.7.17.** *Let $s$ be an interval in $\mathcal{D}(\Gamma)_{[g..k]}$ for some $g \in \{0, \dots, k-1\}$. If $\circ(P_g \mathcal{U} P) \in s^g$ and $P_g \mathcal{U} P \in \mathsf{sel\_ev\_set}_{g+1}$, then $P \in s^i$ for some $i \in \{g+1, \dots, k\}$.*

*Proof.* Let us suppose that $P \notin s^i$ for every $i \in \{g+1, \dots, k\}$. Then, since $s$ is an interval, $s^i \in \mathsf{succ}(s^{i-1})$ for every $i \in \{g+1, \dots, k\}$. Hence, by Definition 4.7.8, there exists a sequence of literals of the form $P_{g+1} \mathcal{U} P, \dots, P_k \mathcal{U} P$ such that $\mathsf{sel\_ev\_set}_h^* = \{P_h \mathcal{U} P\}$ for every $h \in \{g+1, \dots, k\}$ and $P_h \mathcal{U} P$ is the direct descendant of $P_{h-1} \mathcal{U} P$ in $\mathcal{D}(\Gamma)$ for every $h \in \{g+1, \dots, k\}$. Since $s^k$ is standard, by item *(a)* in Definition 4.7.5, there exists a clause of the form $\circ N \in \Gamma_k^*$ such that $\circ(P_k \mathcal{U} P) \in \mathsf{Lits}(\circ N)$. Consequently, since $\mathcal{D}(\Gamma)$ is a cycling derivation with respect to $j$ and $k$, there exists $N \in \Gamma_j$ such that $P_k \mathcal{U} P \in \mathsf{Lits}(N)$. This contradicts the fact that $P_k$ is (according to the rule $(\mathcal{U}\, Set)$) a fresh variable that cannot appear in the set $\Gamma_j$. ■

**Proposition 4.7.18.** *Let $s$ be an interval in $\mathcal{D}(\Gamma)_{[g..h]}$ for some $g$ and $h$ such that $0 \leq g < h \leq k-1$. If $\circ(P_g \mathcal{U} P) \in s^g$, $P_g \mathcal{U} P \in \mathsf{sel\_ev\_set}_{g+1}$ and $P \notin s^i$ for all $i \in \{g+1, \dots, h\}$, then $P_g \in s^i$ for all $i \in \{g+1, \dots, h\}$.*

*Proof.* If $h = g+1$ then $P_g \in s^h$ because $s^h$ is a successor of $s^g$ (see Definition 4.7.8). Now, in the case of $h \geq g+2$, let us suppose that there exists some $r \in \{g+2, \dots, h\}$ such that $P_g \notin s^r$. Since $s$ is an interval, $s^\ell \in \mathsf{succ}(s^{\ell-1})$ for every $\ell \in \{g+1, \dots, h\}$. Hence, by Definition 4.7.8, there exists a sequence of literals of the form $P_{g+1} \mathcal{U} P, \dots, P_h \mathcal{U} P$ such that $P_\ell \mathcal{U} P$ is the direct descendant of $P_{\ell-1} \mathcal{U} P$ in $\mathcal{D}(\Gamma)$, $\mathsf{sel\_ev\_set}_\ell^* = \{P_\ell \mathcal{U} P\}$ and $\{P_{\ell-1}, \circ(P_\ell \mathcal{U} P)\} \subseteq s^\ell$ for every $\ell \in \{g+1, \dots, h\}$. Then, $P_{r-1} \in s^r$. Additionally, by construction of $\mathcal{D}(\Gamma)$, there exists either a clause of the form $C_i = \square(\neg P_i \vee P_{i-1})$ or $C_i = \square\neg P_i$ in $s^r$ for every

$i \in \{g+1, \dots, r\}$.[10] Since we are supposing that $P_g \notin s^r$, then $\{\neg P_{g+1}, \dots, \neg P_r\} \subseteq s^r$ must hold because $s^r$ is literal-closed. Then, $\neg P_{r-1}$ is also in $s^r$. Therefore $\{P_{r-1}, \neg P_{r-1}\} \subseteq s^r$, which contradicts the fact that $s^r$ is locally consistent. ∎

**Proposition 4.7.19.** *Let $\pi = s_0, s_1, \dots, s_n$ be a $\mathcal{D}(\Gamma)$-cycle. If there exists a literal $\circ(P_0 \mathcal{U} P) \in$ $\mathsf{univlit}(\Gamma)$ such that $\circ(P_0 \mathcal{U} P) \in s^i_\ell$ for some $\ell \in \{0, \dots, n\}$ and some $i \in \{j, \dots, k\}$, and the path $\pi^\omega$ does not fulfill $\circ(P_0 \mathcal{U} P)$, then $P_0 \mathcal{U} P \notin \mathsf{sel\_ev\_set}_g$ and $\{P_0, \circ(P_0 \mathcal{U} P)\} \subseteq s^g_h$ for every $h \in \{0, \dots, n\}$ and every $g \in \{j, \dots, k\}$.*

*Proof.* Since $\pi$ is a $\mathcal{D}(\Gamma)$-cycle and $\pi^\omega$ does not fulfill $\circ(P_0 \mathcal{U} P)$, we can ensure, by Definitions 4.7.13, 4.7.8 and 4.7.16 that $P_0 \in s^g_h$ and $P \notin s^g_h$ for every $h \in \{0, \dots, n\}$ and every $g \in \{j, \dots, k\}$. Therefore, by using Proposition 4.7.17 and Proposition 4.7.18, we can ensure that $P_0 \mathcal{U} P \notin \mathsf{sel\_ev\_set}_g$ for every $g \in \{j, \dots, k\}$, since otherwise $\pi^\omega$ would fulfill $\circ(P_0 \mathcal{U} P)$. Consequently, by Definition 4.7.8 and Definition 4.7.13, we can ensure that $\{P_0, \circ(P_0 \mathcal{U} P)\} \subseteq s^g_h$ for every $h \in \{0, \dots, n\}$ and every $g \in \{j, \dots, k\}$. ∎

Next, we prove that every $\mathcal{D}(\Gamma)$-cycle in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ is self-fulfilling. As a consequence, we know that there exists at least one self-fulfilling infinite path in the graph $\mathcal{G}_{\mathcal{D}(\Gamma)}$.

**Lemma 4.7.20.** *For any cycling derivation $\mathcal{D}(\Gamma)$, the graph $\mathcal{G}_{\mathcal{D}(\Gamma)}$ contains at least one self-fulfilling $\mathcal{D}(\Gamma)$-cycle.*

*Proof.* By Proposition 4.7.14 there is at least one $\mathcal{D}(\Gamma)$-cycle in $\mathcal{G}_{\mathcal{D}(\Gamma)}$. We show, by contradiction, that every $\mathcal{D}(\Gamma)$-cycle in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ is self-fulfilling. For that, let us suppose that there is a $\mathcal{D}(\Gamma)$-cycle $\pi = s_0, s_1, \dots, s_n$ in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ that is non-self-fulfilling, i.e., the path $\pi^\omega$ does not fulfill a literal $\circ(P_0 \mathcal{U} P) \in s^i_\ell$ for some $\ell \in \{0, \dots, n\}$ and some $i \in \{j, \dots, k\}$. Then, by Proposition 4.7.19, $P_0 \mathcal{U} P \notin \mathsf{sel\_ev\_set}_g$ for every $g \in \{j, \dots, k\}$ and $\{P_0, \circ(P_0 \mathcal{U} P)\} \subseteq s^i_\ell$ for every $\ell \in \{0, \dots, n\}$ and every $i \in \{j, \dots, k\}$. Since $s^g_h$ is standard for every $\ell \in \{0, \dots, n\}$ and every $i \in \{j, \dots, k\}$, we conclude that, for every $i \in \{j, \dots, k\}$, the set $\Gamma^*_i$ contains a clause $C = \Box^b \circ N$ such that $\circ(P_0 \mathcal{U} P) \in \mathsf{Lits}(C)$ and, consequently, $P_0 \mathcal{U} P \in \mathsf{Lits}(\mathsf{now}(\Gamma_i))$ for every $i \in \{j, \dots, k\}$. Therefore, by Definition 4.6.1(3), $\mathcal{D}(\Gamma)$ is not a cycling derivation, which is a contradiction. ∎

The particular case of Propositions 4.7.17, 4.7.18 and 4.7.19 and Lemma 4.7.20 for eventualities of the form $\diamond P$ follows easily.

Next, we introduce pre-models as a kind of paths along $\mathcal{G}_{\mathcal{D}(\Gamma)}$.

**Definition 4.7.21.** $\mathsf{PMod}(\mathcal{G}_{\mathcal{D}(\Gamma)})$ *is the collection of all finite paths $\pi = s_0, s_1, s_2, \dots, s_n$ in $\mathcal{G}_{\mathcal{D}(\Gamma)}$ such that*

*(a)* $s_0 \in \mathcal{D}(\Gamma)_{[0..j-1]}$ *and* $\sigma = s_1, s_2, \dots, s_n \in \mathsf{cycles}(\mathcal{G}_{\mathcal{D}(\Gamma)})$, *if* $\mathcal{D}(\Gamma)_{[0..j-1]} \neq \emptyset$

*(b)* $\pi = s_0, s_1, \dots, s_n \in \mathsf{cycles}(\mathcal{G}_{\mathcal{D}(\Gamma)})$, *if* $\mathcal{D}(\Gamma)_{[0..j-1]} = \emptyset$

*where* $\mathsf{cycles}(\mathcal{G}_{\mathcal{D}(\Gamma)})$ *is the collection of all the self-fulfilling cycles in* $\mathcal{G}_{\mathcal{D}(\Gamma)}$.

[10] The form of the clause respectively depends on whether the context is empty or not when the rule $(\mathcal{U}\, Set)$ is applied to $\Gamma_i$.

As a direct consequence of Propositions 4.7.10 and 4.7.14 and Lemma 4.7.20, there exists at least one pre-model in the graph $\mathcal{G}_{\mathcal{D}(\Gamma)}$.

**Proposition 4.7.22.** $\mathsf{PMod}(\mathcal{G}_{\mathcal{D}(\Gamma)})$ *is non-empty.* ■

Finally, the above pre-model allows us to construct a model of $\Gamma$. This proves the completeness of our TRS-resolution system.

**Theorem 4.7.23.** *For any set of clauses* $\Gamma$*, if* $\Gamma$ *is unsatisfiable then there exists a* TRS*-refutation for* $\Gamma$.

*Proof.* Suppose that there is no TRS-refutation for $\Gamma$, then the algorithm $\mathcal{SR}$ in Figure 4.10 produces a cycling derivation $\mathcal{D}(\Gamma)$. By Proposition 4.7.22, there exists a pre-model $\pi = s_0, s_1, s_2, \dots, s_n$ in $\mathsf{PMod}(\mathcal{G}_{\mathcal{D}(\Gamma)})$. If $\mathcal{D}(\Gamma)_{[0..j-1]} = \emptyset$ we define $\sigma$ as the infinite path $\pi^\omega$. Otherwise $\sigma = s_0 \cdot \rho^\omega$ where $\rho = s_1, s_2, \dots, s_n$. Now, we define the PLTL-structure $\mathcal{M}_\sigma = (\sigma, V_{\mathcal{M}_\sigma})$ where the states are the standard lclc-extensions that form the intervals in $\sigma$ which can be seen as

$$\Omega_0^0, \dots, \Omega_0^r, \Omega_1^j, \dots, \Omega_1^k, \Omega_2^j, \dots, \Omega_2^k, \dots, \Omega_n^j, \dots, \Omega_n^k, \Omega_\ell^j, \dots, \Omega_\ell^k, \dots$$

where $r = j-1$ and $\ell = 1$ if $\mathcal{D}(\Gamma)_{[0..j-1]} \neq \emptyset$, whereas $r = k$ and $\ell = 0$ if $\mathcal{D}(\Gamma)_{[0..j-1]} = \emptyset$. Additionally, $\Omega_h^g$ is in $\mathsf{lclc}(\Gamma_g^*)$ and $V_{\mathcal{M}_\sigma}(\Omega_h^g) = \{p \in \mathsf{Prop} \mid p \in \Omega_h^g\}$ for every $g \in \{0, \dots, k\}$ and every $h \in \{0, \dots, n\}$. It is routine to see that $\langle \mathcal{M}_\sigma, \Omega_h^i \rangle \models C$ holds for all $C \in \Gamma_i^*$. Since any lclc-extension contains at least one literal of $C$, this is made by structural induction on the form of the literal and using Definition 4.7.8 and the fact that $\sigma$ is self-fulfilling (by Lemma 4.7.20). In particular, $\mathcal{M}_\sigma$ is a model of $\Gamma_0^*$ and, by Propositions 4.5.1 and 4.5.2, the set $\Gamma_0$ is satisfiable. Hence, since $\Gamma = \Gamma_0$, the set of clauses $\Gamma$ is satisfiable. ■

## 4.8 Related Work

In this section we describe the contributions in the literature that are more closely related to our approach to clausal temporal resolution. First, we explain the relation with the tableau method TTM (presented in the previous chapter) that inspired TRS-resolution. And then, we discuss and compare the four clausal resolution methods ([29, 1, 126, 40]) that are more similar to TRS-resolution.

### 4.8.1 The TTM Tableau Method [58, 61]

The TRS-resolution method is strongly inspired in the TTM tableau method introduced in the previous chapter (see also [58, 61]). Indeed, the TRS-rule $(\mathcal{U}\,Set)$ is a clausal variant of the TTM-rule $(\mathcal{U})_2$. In Chapter 3 (see also [60, 61]), the idea behind the rule $(\mathcal{U})_2$ is used for achieving cut-freeness (in particular, invariant-freeness) in the framework of sequent calculi for PLTL. In particular, the cut-free sequent calculus GTC that is dual to the one-pass tableau method TTM is presented.

The crucial point –in both rules $(\mathcal{U})_2$ and $(\mathcal{U}\,Set)$– is the fact that whenever a set of formulas $\Delta \cup \{\varphi \mathcal{U} \psi\}$ is satisfiable, there must exist a model $\mathcal{M}$ (with states $s_0, s_1, \dots$) that is minimal in the following sense:

$$\mathcal{M} \textit{ satisfies either } \Delta \cup \{\psi\} \textit{ or } \Delta \cup \{\varphi, \circ((\varphi \wedge \neg\Delta)\,\mathcal{U}\,\psi)\}$$

In other words, in a minimal model $\mathcal{M}$ such that $\langle \mathcal{M}, s_0 \rangle \not\models \psi$, the so-called *context* $\Delta$ cannot be true from the state $s_1$ until the state where $\psi$ is true. Regarding tableaux, the rule $(\mathcal{U})_2$ –which is crucial in our approach for getting a one-pass method– allows to split a branch containing a node labelled by $\Delta \cup \{\varphi\,\mathcal{U}\,\psi\}$ into two branches respectively labelled by $\Delta \cup \{\psi\}$ and $\Delta \cup \{\varphi, \circ((\varphi \wedge \neg\Delta)\,\mathcal{U}\,\psi)\}$. Hence, the negation of the successive contexts $\Delta$ will be required by the postponed eventuality. Provided that the number of possible contexts $\Delta$ is finite, the fulfillment of $\varphi\,\mathcal{U}\,\psi$ cannot be indefinitely postponed, without getting a contradiction. Of course, the procedure must fairly select an eventuality to ensure termination. Tableau rules handle general formulas, whereas resolution needs a preliminary transformation to the clausal language before the rules can be applied. The rule $(\mathcal{U}\,Set)$ introduced in this chapter is an adaptation –to the clausal language setting– of the tableau rule $(\mathcal{U})_2$, in the sense that $(\mathcal{U}\,Set)$ is applied to a set of clauses and the eventuality is inside a clause whereas in $(\mathcal{U})_2$ the eventuality is itself a formula.
Regarding worst-case complexity, the upper bound given for TTM in Proposition 3.4.10 coincides with the one for TRS-resolution (see Proposition 4.6.17). The computational cost of introducing the negation of the context in postponed eventualities not only depends on the size of the context but also on its form. As pointed out in Subsection 3.4.5, there are syntactically detectable classes of formulas that can be disregarded when negating the context. In particular the most remarkable class is formed by formulas of the form $\Box\,\varphi$. The rule $(\mathcal{U}\,Set)$, by definition, does not consider the always-clauses when negating the context. Since often most of the clauses are always-clauses, i.e. formulas of the form $\Box\,\varphi$ where $\varphi$ is in clausal normal form, the rule $(\mathcal{U}\,Set)$ is specifically well suited for clausal resolution.

### 4.8.2 The Resolution Method of Cavali & Fariñas del Cerro [29]

The complete resolution method presented in [29] deals with a language that is strictly less expressive than full PLTL since only the temporal connectives $\circ$, $\Box$ and $\Diamond$ are allowed. The normal form is based only on distribution laws, and renaming is not used to remove any nesting of operators. Consequently, their translation into the normal form does not introduce new variables, at the price of achieving little reduction of nesting of classical and temporal connectives. A formula in Conjunctive Normal Form is a conjunction of clauses $C_1 \wedge \ldots \wedge C_r$ where every clause $C_j$ has the following recursive structure

$$L_1 \vee \ldots \vee L_n \vee \Box\,\delta_1 \vee \ldots \vee \Box\,\delta_m \vee \Diamond\,\kappa_1 \vee \ldots \vee \Diamond\,\kappa_h$$

Here each $L_j$ is of the form $\circ^i p$ or $\circ^i \neg p$ with $p$ being a propositional atom, each $\delta_j$ is a clause and each $\kappa_j$ is a conjunction where every conjunct is a clause. The resolution method is based on considering different cases in order to check whether formulas that must be satisfied at the same state are contradictory or not. For instance, for deciding whether $\{\Box\,\varphi, \Diamond\,\psi\}$ is unsatisfiable, the unsatisfiability of $\{\Diamond\,(\varphi \wedge \psi)\}$ is analyzed. Similarly, in order to decide whether $\{\Diamond\,\varphi, \Diamond\,\psi\}$ is unsatisfiable, the unsatisfiability of $\{\Diamond\,\varphi, \psi\}$ and $\{\varphi, \Diamond\,\psi\}$ is analyzed. Also formulas of the form $\varphi \vee \circ\varphi \vee \ldots \vee \circ^i\varphi$ and of the form $\neg\varphi \wedge \circ\neg\varphi \wedge \ldots \wedge \circ^{i-1}\neg\varphi \wedge \circ^i\varphi$ are considered for dealing with $\Diamond\,\varphi$ and formulas of the form $\varphi \wedge \circ\varphi \wedge \ldots \wedge \circ^i\varphi$ for dealing with $\Box\,\varphi$, with $i$ ranging in a finite set of the form $\{0, \ldots, g\}$ where $g \geq 0$. These latter cases represent an attempt to decide whether there exists a future state (in a finite scope) in which the involved formula (the formula $\varphi$ from $\diamond\varphi$ or from $\Box\varphi$) does not generate an inconsistency. However, there is not a clear algorithm to construct derivations and, therefore, complexity cannot be analyzed. In our approach, the nesting of connectives in the normal form is much more restricted. Our resolution method is based on reasoning "forwards in time" state by state. And, finally, our method is complete for full PLTL and we provide a terminating algorithm to construct derivations. In [28] an extension of the resolution method presented in [29] is shown and the full expressiveness of PLTL is achieved by means of the connectives $\circ$ and $\mathcal{P}$ ("precedes") such that $\varphi\,\mathcal{P}\,\psi$ is equivalent to the until-formula $(\neg\psi)\,\mathcal{U}\,(\varphi \wedge \neg\psi)$, but the completeness result for the extended method is not provided.

### *4.8.3 The Nonclausal Resolution Method of Abadi & Manna [1]*

A nonclausal resolution method for full PLTL is presented in [1] (see also [4]). Eventualities are expressed by means of the connectives $\diamond$ and $\mathcal{P}$ ("precedes"). Since they deal with general formulas (instead of clauses), the provided rules enable the manipulation and simplification of subformulas at any level but with some restrictions for preserving soundness. The resolution rule is of the form

$$\varphi[\chi], \psi[\chi] \longmapsto \varphi[true] \vee \psi[false]$$

where the occurrences of the subformula $\chi$ in $\varphi$ and $\psi$ that are replaced with $true$ and $false$, respectively, are all in the scope of the same number of $\circ$'s and are not in the scope of any other modal operator in either $\varphi$ or $\psi$. They also use modality rules, such as e.g. $\Box\varphi, \diamond\psi \longmapsto \diamond((\Box\varphi) \wedge \psi)$ and $\diamond\varphi, \diamond\psi \longmapsto \diamond((\diamond\varphi) \wedge \psi) \vee \diamond(\varphi \wedge \diamond\psi)$, that makes this non-clausal method very different from our proposal. However, they also introduce *induction rules* for dealing with eventualities. These induction rules are very close to our rule $(\mathcal{U}\,Set)$. Here, for simplicity and clarity, we only describe the induction rule for $\diamond$, which in terms of the present thesis says

$$\Delta, \Delta', \diamond\varphi \longmapsto \Delta, \Delta', \diamond(\neg\varphi \wedge \circ(\varphi \wedge \neg\Delta)) \text{ if } \vdash \neg(\Delta \wedge \varphi)$$

where $\Delta$ and $\Delta'$ are set of formulas. This rule states that if $\Delta$ and $\varphi$ cannot hold at the same time but $\varphi$ eventually holds, then there must be a sate $s_j$ where $\varphi$ does not hold and at the next state $s_{j+1}$ the formulas $\varphi$ and $\neg\Delta$ hold. Hence, the above $\Delta$ (called a *fringe* in [1]) resembles our context, but the technical handling of fringes in [1] is quite different from our treatment of contexts. The first important difference is that induction rules use an aside condition (see $\vdash \neg(\Delta \wedge \varphi)$ above) for choosing the fringe $\Delta$. In our approach, contexts are syntactically determined without any auxiliary derivation. Second, in $(\mathcal{U}\,Set)$ accumulation of the contexts is made in the non-eventuality part of the until-formula, i.e. the left-hand subformula of the until-formula. Indeed, the consequent of the TRS-rule $(\diamond\,Set)$ introduces an until-formula with the negated context in the left-hand subformula. In contrast, negated fringes are accumulated in the eventuality part. Third, the method in [1] does not impose any deterministic or systematic strategy to apply the induction rules although the completeness proof outlines a strategy based on the finiteness of the set of possible fringes. We provide, by means of the algorithm $\mathcal{SR}$, a systematic method. Additionally, in our method when a context is repeated, the derivation of a refutation is straightforward, whereas in [1] obtaining a refutation after a repetition is not so direct. The reason is that our forward reasoning approach keeps a better structure for detecting the contradiction between a context and its negation. This fact can be seen by looking at the following example $\{p, \Box(\neg p \vee \circ p), \diamond\neg p\}$. In our method a refutation is easily achieved when the context $\{p\}$ is repeated (see Example 4.6.3). However, by using the induction rule in [1] with $\Delta = \{p\}$ and $\Delta' = \{\Box(\neg p \vee \circ p)\}$, they get

$$\{p, \Box(\neg p \vee \circ p), \Diamond(\neg\neg p \wedge \circ(\neg p \wedge \neg p))\}.$$

Applying some other rules, which we cannot detail here, this set is transformed into

$$\{p, \circ p, \circ\Box(\neg p \vee \circ p), \Diamond(p \wedge \circ\neg p)\}.$$

The resolution rule is not enough for achieving a contradiction from the latter set. Fourth, [1] does not address the problem of satisfiable input sets, whereas we ensure the existence of a model for any satisfiable input through the notion of cycling derivation. Finally, complexity is not discussed in [1, 4] and is difficult to assess due to the lack of a clear strategy for applying the rules.

### *4.8.4 Venkatesh's Temporal Resolution [126]*

The resolution method presented in [126] is very similar to ours in everything but the way of dealing with eventualities. The normal form and even the way in which the new variables are used during the translation process are the same as ours. The resolution rule and the way of unwinding temporal literals –in the case of our rules $(\mathcal{U}\,Fix)$ and $(\mathcal{R}\,Fix)$– follow the same idea. Also the approach of reasoning forwards, i.e., jumping to the next state carrying the clauses that must be necessarily satisfied in the next state, appears in both methods. However, in sharp contrast to our TRS-resolution, the method in [126] needs invariant property generation for dealing with eventualities that can unwind indefinitely (or whose fulfillment can be delayed indefinitely). More precisely, cyclic sequences of sets of clauses that contain the so-called *persistent eventualities* –eventualities that can be unwound indefinitely and cannot be satisfied– must be detected and the persistent eventualities must be removed. Detecting those cycles can be seen as finding an invariant property $\chi$ that ensures that a given eventuality $\varphi\,\mathcal{U}\,\psi$ cannot be fulfilled because $\Box\neg\psi$ follows from $\chi$. Finding the invariant property requires an additional process whose development is not tackled in [126], therefore the complexity of the method cannot be directly assessed. Instead of invariant properties, we use the concept of context –in the applications of the rule $(\mathcal{U}\,Set)$– for preventing indefinite unwinding of eventualities.

### *4.8.5 Fisher's Temporal Resolution [40]*

The resolution method presented in [40] is also for full PLTL. The structure of a formula in the Separated Normal Form (SNF) is $\Box C_1 \wedge \ldots \wedge \Box C_r$ and since it is equivalent to $\Box(C_1 \wedge \ldots \wedge C_r)$, the calculations are made using only the so-called PLTL-clauses $C_1, \ldots, C_r$, without $\Box$. Each $C_j$ is of one of the following three forms

$$\mathbf{start} \to \delta \qquad\qquad \kappa \to \circ\,\delta \qquad\qquad \kappa \to \Diamond\,\lambda$$

where $\to$ is the classical connective of implication (i.e. $\chi \to \gamma \equiv \neg\chi \vee \gamma$), $\mathbf{start}$ is a nullary connective that is only true in the initial state, $\delta$ is a disjunction of propositional literals, $\kappa$ is a conjunction of propositional literals and $\lambda$ is a propositional literal. The use of $\mathbf{start}$ makes possible to differentiate the clauses that refer only to the first state and the clauses that refer to all the states. Additionally, in SNF only the temporal connectives $\circ$ and $\Diamond$ are kept, since any clause involving one of the remaining connectives ($\mathcal{U}$, $\Box$, etc.) is expressed by a set of new clauses whose only temporal connectives are $\circ$ and $\Diamond$. A formula and the corresponding set of clauses in SNF are equisatisfiable but, in general, they are not logically equivalent. The three kinds of clauses are called, respectively, *initial* PLTL-clauses, *step* PLTL-clauses and *sometime* PLTL-clauses. Resolution between the former two kinds of clauses is a straightforward generalization of classical resolution but the so-called *temporal resolution rule* for sometime PLTL-clauses is more complicated:

$$\frac{\kappa_0 \to \circ\,\delta_0, \ldots, \kappa_n \to \circ\,\delta_n, \kappa_{n+1} \to \diamond\,\lambda}{\text{SNF}(\kappa_{n+1} \to (\neg\kappa_0 \wedge \ldots \wedge \neg\kappa_n)\mathcal{W}\lambda)}$$

where the *unless* or *weak until* connective $\mathcal{W}$ is defined as $\varphi\mathcal{W}\psi \equiv (\varphi\,\mathcal{U}\,\psi) \vee \Box\varphi$. Additionally the following *loop side conditions* must be valid

$$\delta_j \to \neg\lambda \text{ and } \delta_j \to (\kappa_0 \vee \ldots \vee \kappa_n) \text{ for every } j \in \{0, \ldots, n\}$$

The idea is that if the set $\Omega = \{\kappa_0 \to \circ\,\delta_0, \ldots, \kappa_n \to \circ\,\delta_n\}$ satisfies the loop side conditions, then it follows that $(\kappa_0 \vee \ldots \vee \kappa_n) \to \circ\Box\neg\lambda$. In such a case $\Omega$ is called a loop in $\diamond\,\lambda$ and $\kappa_0 \vee \ldots \vee \kappa_n$ is called a loop formula (also called invariant) in $\neg\lambda$. So the method is based on searching for the existence of these invariant properties. This task requires specialized graph search algorithms (see [45, 33]) and is the most intricate part of this approach. The worst-case complexity is discussed in [45], where the translation to SNF is proved to be linear in the length of the input, whereas resolution is doubly exponential in the number of proposition symbols. An improved and simplified version of the resolution method in [40] can be found in [32]. The main differences with respect to TRS-resolution method are three. First, although the technique of renaming complex subformulas by a new proposition symbol is used in both approaches, in our normal form the temporal connectives $\mathcal{U}$ and $\mathcal{R}$ are kept. Second, we follow the approach of reasoning forwards and jumping to the next state when necessary, whereas the method presented in [40] involves reasoning backwards. Actually, contradictions are achieved at the initial state. Third, the most remarkable difference is the way of dealing with eventualities, since we dispense with invariant generation by means of the rule $(\mathcal{U}\,Set)$ and the strategy presented in the algorithm $\mathcal{SR}$.

# 5. LOGICAL FOUNDATIONS FOR MORE EXPRESSIVE DECLARATIVE TEMPORAL LOGIC PROGRAMMING LANGUAGES

## *5.1 Introduction*

Temporal Logic Programming (TLP) deals with the direct execution of temporal logic formulas. Hence TLP provides a single framework in which dynamic systems can be specified, developed, validated and verified by means of executable specifications that make possible to prototype, debug and improve systems before their final use. In TLP, the direct execution of a formula corresponds to building a model for that formula. The idea of directly executing logic formulas has been thoroughly studied in (classical) Logic Programming (LP). Given a program $\Pi$, the computation of a goal $\bot \leftarrow \gamma$ with respect to $\Pi$ in an LP system is a search for a refutation proof of $\Pi \cup \{\neg\gamma\}$. However, this proof search can also be seen as an attempt to build a model of $\Pi \cup \{\gamma\}$. This model is (in general) partially specified, because it only determines the truth value of the atoms (from $\Pi$) that are involved in the refutation proof. We illustrate this view (of LP) in the next example.

**Example 5.1.1.** *Let us consider the following (classical) logic program:*

$$\begin{array}{l} q(0) \leftarrow \top \\ q(X) \leftarrow q(Y) \wedge X = Y + 1 \\ r(X) \leftarrow q(Y) \wedge X = Y + 2 \\ w(X) \leftarrow q(Y) \wedge X = Y + 3 \end{array}$$

*The computation of the goal $\bot \leftarrow r(Z)$ gives rise to the infinite sequence of answer substitutions $\{Z \leftarrow 2\}, \{Z \leftarrow 3\}, \{Z \leftarrow 4\}, \ldots$ that partially shows the implicit step by step construction of the infinite minimal model $\{q(j), r(j+2) \mid j \in I\!N\}$ for the body of the goal (i.e. $r(Z)$) and the subprogram that contains the first three program clauses. However, this model does not specify which instances of $w(X)$ are true.*

TLP, in a broad sense, means programming in any language based on temporal logic. In TLP two different approaches have arisen: the *imperative future* approach and the *declarative* approach. In the imperative future approach a program is a set of rules of the form $\varphi \rightarrow \circ\psi$ asserting that whenever the formula $\varphi$ is true in a state $s$, the next state $s'$ must make true the formula $\psi$. The imperative future approach tries to construct a model of the whole input program by using a forward chaining process. By contrast, the declarative approach to TLP is based on extending classical resolution for dealing with temporal connectives. Hence the (implicit) attempt of constructing a model is driven by the goal. As the above Example 5.1.1 shows, such model determines only the predicates involved in the refutational process. Next, we briefly review the most significant proposals in the literature for both approaches. More discussion and references about programming languages with capabilities for reasoning about time can be found e.g. in [67, 44, 98, 100].

**Imperative future TLP languages.** The most significant representatives of this approach are Tempura [94] and MetateM [9]. The language Tempura is based on a fragment of Interval Temporal Logic with a restricted use of eventualities. The Tempura approach has been continued ([27, 95]) and extended to Framed Tempura and Projection Temporal Logic Programming [37, 38, 129].
The language MetateM develops the methodology outlined in [54]. MetateM is based on First-order Linear-time Temporal Logic (FLTL) and formulas are written in the Separated Normal Form (SNF) presented in [40, 41]. The propositional fragment of MetateM is complete, however, since FLTL is incomplete ([92, 122, 121]), the execution of a first-order MetateM program attempts to build a model, but the success of such construction is not guaranteed (see Example 5.1.4). In MetateM disjunctions are seen as choices and one disjunct is selected from each disjunction as part of the process of building a model. If a choice is later shown to be inappropriate, because it leads to inconsistency, then backtracking is used to return to the last point where a choice was made. In propositional MetateM the termination is addressed by explicitly considering the small model property, which allows to calculate an upper bound of forward chaining steps. If a model is not obtained bellow this upper bound, then the attempt is given up and the procedure backtracks. MetateM was extended to Concurrent MetateM in [42]. Among its applications we can mention, e.g., the development of agent systems ([43, 47]). More references on MetateM, Concurrent MetateM and their applications can be found in [44]. A fragment of Linear-time Temporal Logic is presented as imperative future TLP language in [93]. This language, for efficiency, restricts the use of eventualities (and also disjunctions). The clausal normal form and the idea of forward chaining construction of models introduced in MetateM are used in [6, 7] to obtain a temporal extension of the Answer Set Programming paradigm (non-monotonic reasoning).
Finally, we also mention the assembly-like TLP language XYZ/E that was presented in [124, 125] as a vehicle for providing temporal semantics to programs written in conventional imperative programming languages. An imperative program is expressed in XYZ/E on the basis of the execution sequences that it generates along the timeline. A similar approach can be found in Chapter 3 of [44].

**Declarative TLP languages.** There are several works on extending classical LP (in particular Prolog) for reasoning about time. Some proposals are purely based on temporal logic and extensions of SLD resolution, but the incompleteness of FLTL becomes a delicate issue for using fragments of FLTL as TLP languages. Also the complexity result is a drawback even for the propositional fragment (see [119]). Additionally, the interaction between the $\Box$ ("always") and the $\circ$ ("next") connectives makes possible to encode the so-called *induction on time* by means of loops or hidden invariants (see Section 2.4) that, in an indirect way, state that a formula is satisfied in every moment in time. The presence of these loops or hidden invariants makes necessary to consider quite intricate mechanisms for detecting (un)satisfiable eventualities (Definition 2.2.1). Many temporal extensions of LP are not purely founded on temporal logic due to their extra-logical features for handling eventualities. Next, we summarize representative published work concerning the variety of proposals in declarative TLP languages (including some approaches that are not purely based on temporal logic).
The language Tokio [52, 82, 83, 96] extends Prolog by adding temporal reasoning capabilities inspired by both Linear-time Temporal Logic and Interval Temporal Logic. In Tokio there are

restrictions regarding the use of temporal connectives and, unlike Prolog variables, the so-called temporal variables used in Tokio have state, what makes possible to express properties like $\circ Y = Y + 1$ stating that the value of the variable $Y$ in the next time instant will be its present value plus one. Obviously, this kind of expressions are no supported by conventional temporal logic.

A different temporal extension of Prolog was introduced by Hrycej in [74, 75] where time intervals are considered as conceptual primitives. The Hrycej's language is a non-modal approach based on first-order logic with capabilities to deal with time intervals. More precisely, the first-order "reified" logic ([108, 118]) is considered as the basis for the implementation of the language.

Metric temporal operators and dense time are considered in [21, 22, 24, 23, 25, 26] where execution is based on translating temporal logic programs into Constraint Logic Programming. Temporal Annotated Constraint Logic Programming is presented in [50, 49, 51, 107].

The Temporal Prolog presented in [114] extends Prolog by introducing linear-time temporal connectives. Programs are transformed into a normal form that is similar to the Separated Normal Form used in MetateM. This transformation removes most temporal connectives by introducing fresh predicates. The transformation of eventualities yields negated atoms. If negated atoms (i.e., eventualities) are involved in a program, then the Herbrand universe must be finite and, in this case, computation is performed on the basis of a nondeterministic finite automaton that corresponds to the program. Two implementation options are devised: first, by translating programs into Prolog (if the program contains negation, then a pure Prolog program is not obtained) and second, asserting the facts which are true at each point in time (although this implementation option is not explained in detail, it resembles, at first sight, the imperative future approach).

A sequent-based proposal for establishing logical foundation for declarative TLP is presented in [106]. This approach considers a complete fragment of $\mathsf{FLTL}$ where eventualities are allowed. In order to handle eventualities, the sequent system contains an invariant-based rule.

We finally review the three existing declarative TLP languages that are based on pure extensions of classical logic programming languages and resolution, which are Chronolog [127, 99], Templog [2, 3, 10, 11, 12, 13, 14] and Gabbay's Temporal Prolog [55]. Chronolog and Templog are the most studied and the most representative languages in the purely declarative approach. The underlying logic for the languages Templog and Chronolog is $\mathsf{FLTL}$. In the case of Gabbay's Temporal Prolog, the presented system is intended for both branching-time and linear-time temporal logic. In Chronolog, the connectives $\mathsf{first}$ (to refer to the state $s_0$) and $\mathsf{next}$ (to refer to the next state) are the only temporal connectives. Templog's syntax allows the always connective ($\Box$) to occur in clause heads and the eventually connective ($\Diamond$) in clause bodies. However, Templog programs are expressible by using $\circ$ as the unique temporal connective in clause heads and bodies ([12, 14]) and consequently it has the same expressive power as Chronolog. This restriction is so strong that it allows reducing any temporal program to a (possibly infinite) classical logic program. Templog and Chronolog have also the same metalogical properties of existence of minimal model and fixpoint characterization. Gabbay's Temporal Prolog is a more expressive language that allows eventualities in clause heads (although it does not allow $\Box$ in clause bodies). The resolution-based computation procedure outlined in [55] is proved to be sound, however its completeness has not been addressed. The development of these three declarative languages was mainly done in the early nineties, in contrast to the imperative future approach (e.g. Tempura and MetateM) which has been evolving until present days. During the last two decades, no other clausal sublanguage of linear-time temporal logic has been proposed as declarative TLP language. Hence, nowadays, Templog, Chronolog and Gabbay's Temporal Prolog remain as the most expressive proposals of declarative TLP languages. Later extensions of Chronolog (e.g. [103, 102, 112, 113, 68]) did not add significant temporal expressiveness. In the case of Gabbay's Temporal Prolog, although the expressive power was considerably high, it seems that the lack of completeness was a handicap for further study and development.

In general, it seems that the troublesome solving (in the resolution sense) of the so-called eventualities has been blocking the steps toward more expressive resolution-based declarative TLP languages. Indeed, even in the propositional fragment –i.e. in PLTL– the solving of eventualities is the most intricate part that often requires techniques such as invariant generation ([40, 45]).

In this thesis, we contribute to the effort of increasing the temporal expressiveness of declarative TLP languages on the basis of the temporal resolution-based mechanism presented in the previous chapter (see also [62]) that is complete (in the propositional setting). As already explained in Chapter 4, the main novelty of this temporal resolution lies in a new approach to handle eventualities. We introduce a purely declarative propositional TLP language, called TeDiLog, that allows both $\Box$ and $\Diamond$ in clause heads and bodies. Hence, TeDiLog is strictly more expressive than the propositional fragments of the above mentioned purely declarative proposals: Templog [3, 12], Chronolog [127, 99] and Gabbay's Temporal Prolog [55]. Additionally TeDiLog is as expressive as propositional MetateM [9]. However, MetateM follows the imperative future approach and is not based on resolution. Two crucial differences of our proposal with MetateM are that TeDiLog does not need backtracking and the resolution mechanism of TeDiLog directly manages unsatisfiable eventualities, hence upper bounds are not needed.

A very preliminary version of the content provided in this chapter was presented at the Spanish Workshop PROLE 2009 (see [64]).

Along the chapter, we compare TeDiLog with its most closely related proposals: Templog, Chronolog, the linear-time Gabbay's Temporal Prolog and MetateM. The technical content of this chapter is focused on the propositional language TeDiLog. However, for a better illustration of the aim of our proposal, we next discuss some first-order program examples. They are written in the natural extension of TeDiLog with predicates and variables.

**Example 5.1.2.** *Consider the following program (on Fibonacci numbers):*

$$\begin{array}{l} fib(0) \leftarrow \top \\ \circ fib(1) \leftarrow \top \\ \Box(\circ^2 fib(V) \leftarrow fib(X) \wedge \circ fib(Y) \wedge V = X + Y) \end{array}$$

*The goal $\bot \leftarrow \circ^3 fib(Z)$ yields the answer substitution $\{Z \leftarrow 2\}$. The goal $\bot \leftarrow \Diamond fib(Z)$ produces an infinite sequence of answer substitutions $\{Z \leftarrow 0\}$, $\{Z \leftarrow 1\}$, $\{Z \leftarrow 1\}$, $\{Z \leftarrow 2\}$, ..., that is, the sequence of Fibonacci numbers. Now, consider the goal $\bot \leftarrow \Box fib(Z)$ which is not expressible in Templog, Chronolog and Gabbay's Temporal Prolog. The* TeDiLog *computation does not finish and does not produce any answer. Note that $\Box fib(j)$ is not a logical consequence of the program for any $j \in I\!N$.*
*The above program is expressible in MetateM through a simple transformation. The MetateM program execution, which does not need a goal, builds the infinite model*

$$\{fib(0), \circ fib(1), \circ^2 fib(1), \circ^3 fib(2), \ldots\}$$

*for the above program.*

**Example 5.1.3.** *The following program encodes the so-called induction on time (for $q(a)$):*

$$\begin{array}{l} q(a) \leftarrow \top \\ \square\,(\circ q(X) \leftarrow q(X)) \end{array}$$

*Hence, $q(a)$ is true at every instant along the time. The goal $\bot \leftarrow \circ^3 q(Z)$ yields the answer substitution $\{Z \leftarrow a\}$. The goal $\bot \leftarrow \diamond\, q(Z)$ generates the infinite sequence of answer substitutions $\{Z \leftarrow a\}$, $\{Z \leftarrow a\}$, $\{Z \leftarrow a\}$, $\{Z \leftarrow a\}$, .... The goal $\bot \leftarrow \square\, q(Z)$ also yields the answer substitution $\{Z \leftarrow a\}$. The latter goal is neither expressible in Templog, nor Chronolog, nor Gabbay's Temporal Prolog. The MetateM system builds the infinite model $\{q(a), \circ q(a), \circ^2 q(a), \ldots\}$ for the above program.*

**Example 5.1.4.** *The following program shows that, as expected, the natural first-order extension of* TeDiLog *gives rise to an incomplete system:*

$$\begin{array}{l} q(0) \leftarrow \top \\ \square\,(\circ q(X) \leftarrow q(X)) \\ \square\,(\circ q(X) \leftarrow q(Y) \wedge X = Y + 1) \\ \square\,(w(X) \leftarrow \square\, q(X)) \end{array}$$

*This fact is due to the interaction between the infinite domain and the connective $\square$ in the body of the last clause. By means of the first three clauses, for every $i \in I\!N$, $q(i)$ holds in all states $s_j$ such that $j \geq i$. As a consequence, $w(i)$ holds in a state $s_j$ if $i \geq j$. Indeed, the atoms $w(0), \circ w(0), \circ w(1), \circ^2 w(0), \circ^2 w(1), \circ^2 w(2), \ldots$ are logical consequences of the program. However, the first-order extension of our resolution method will neither yield any answer for the goal $\bot \leftarrow \diamond\, w(Z)$ nor for any goal $\bot \leftarrow \circ^k w(Z)$ where $k \geq 0$. The reason is that, by contrast with the previous Example 5.1.3, here the goal $\bot \leftarrow \square\, q(V)$ does not give any answer (due to the infinite domain), and consequently the last program clause cannot be used to produce $w(V)$.*

*In order to obtain a MetateM program, the last program clause above is translated into SNF giving rise to two clauses: $\square\,(\circ r(X) \leftarrow q(X) \wedge \neg w(X))$ and $\square\,(\diamond \neg q(X) \leftarrow r(X))$, where $r$ is a fresh predicate symbol. Consequently MetateM attempts to construct a model for the following program*[1]*:*

$$\begin{array}{l} q(0) \leftarrow \top \\ \square\,(\circ q(X) \leftarrow q(X)) \\ \square\,(\circ q(X) \leftarrow q(Y) \wedge X = Y + 1) \\ \square\,(\circ r(X) \leftarrow q(X) \wedge \neg w(X)) \\ \square\,(\diamond \neg q(X) \leftarrow r(X)) \end{array}$$

*Then, the atoms in $\{q(0), \circ q(0), \circ q(1), \circ^2 q(0), \circ^2 q(1), \circ^2 q(2), \ldots\}$ are successively obtained. In addition, since there is no clause with head $w(Z)$, we can suppose that $\neg w(X)$ succeeds in a time instant for any $X$ such that $q(X)$ is true at that time instant. Therefore, the atoms*

$$\{\circ r(0), \circ^2 r(0), \circ^2 r(1), \circ^3 r(0), \circ^3 r(1), \circ^3 r(2), \ldots\}$$

[1] Actually this program is not in pure SNF yet (see e.g. [41]). Some minor syntactical changes are still needed, but they are irrelevant for our discussion.

*are also generated. According to the last program clause, the system attempts to satisfy $\diamond \neg q(X)$, however at each step the system must delay this task for the next step. Therefore, MetateM (as* TeDiLog*) is not able to generate a model for this program.*

In the rest of the chapter we restrict ourselves to the propositional setting. Hence, the logic that underlies TeDiLog is the well-known Propositional Linear-time Temporal Logic (PLTL), which is complete and decidable. We endow TeDiLog with logical and operational semantics and prove their equivalence. The logical semantics is given by the set of all the (finite) formulas of the form $\alpha_1 \vee \ldots \vee \alpha_n$ that are logical consequences (in PLTL) of the program and where each $\alpha_j$ is either a body or a body prefixed by the connective $\diamond$. The operational semantics of TeDiLog is based on the *invariant-free resolution method* that is presented in detail in Chapter 4 of this thesis (see also [62]) and dispenses with invariant generation. We cannot expect to have the classical *Minimal Model Property* (MMP in short) that assigns to any program a minimal model, which is the intersection of all its models. The reason for this is twofold. First, the non-conjunctive temporal connective $\diamond$ appearing in clause heads, and also the non-finitary connective $\square$ appearing in clause bodies, both (separately) prevent from holding the MMP (see [101, 99]). For Gabbay's Temporal Prolog the MMP does not hold because of the use of the connective $\diamond$ in clause heads. The second reason is that our resolution mechanism produces (in computation time) disjunctive clauses, so TeDiLog is located in the disjunctive logic programming (DLP) paradigm, which does not enjoy the MMP even in the classical (non-temporal) case. In the DLP framework, the semantics of a program consists of the collection of all its minimal models (see e.g. [89]). Temporal disjunctive logic programming has previously been addressed in [68] where Chronolog is extended with DLP features. The satisfiability of a Templog/Chronolog program can be reduced to the satisfiability of a classical logic program. As a consequence, the minimal model characterization of Templog and (Disjunctive) Chronolog (see [12, 68, 127, 99]) is a straightforward adaptation of the classical (disjunctive) case. In the case of TeDiLog, due to the fact that syntactical cut elimination seems to be unfeasible in PLTL (indeed, it is an open problem in [20] and [61]), the collection of minimal models associated to a program should be related to every possible goal. This results in a too intricate (hence, unseemly) model-theoretic characterization to be used as declarative semantics for TeDiLog. Indeed, although a continuous immediate consequence operator can be associated to every program, there are great difficulties (related to cut elimination) for using this operator in a customary completeness proof. Hence, we prove completeness with respect to the logical semantics through a particular model construction.

Our resolution system requires the expressive power of full temporal logic. That is, the resolution of a $\diamond$-goal, necessarily generates subgoals involving the strictly more expressive connective $\mathcal{U}$. Hence, we directly formulate our language in terms of the temporal connectives $\mathcal{U}$ and its dual: the connective $\mathcal{R}$. We present a complete algorithm which performs resolution of a goal with respect to a program. This algorithm is based on a natural extension of the classical LP rule for (binary) resolution in two senses: temporal ($\square$ in front of clauses) and disjunctive (disjunction in clause heads). The algorithm not only performs the standard (linear) resolution between the current goal and a selected program clause, but also a controlled kind of resolution called *nx-resolution*. This nx-resolution is performed to infer (from program clauses) all the (program) clauses that have a connective $\circ$ in front of every literal. Intuitively, nx-resolution allows to extract all the implicit information about the next state that is crucial to achieve completeness.

$$L ::= p \mid \neg p \qquad\qquad H ::= \bot \mid A \vee H$$
$$T ::= L\,\mathcal{U}\,p \mid L\,\mathcal{R}\,p \mid \Diamond p \mid \Box p \qquad\qquad B ::= \top \mid A \wedge B$$
$$A ::= \circ^i p \mid \circ^i T \qquad\qquad D ::= \Box^b(A \vee H \leftarrow B)$$
$$G ::= \Box^b(\bot \leftarrow B)$$

where $p \in \mathsf{Prop}$, $i \in I\!N$, $\bot$ is the empty disjunction, $\top$ is the empty conjunction and $b \in \{0,1\}$.

*Figure 5.1:* Syntax of TeDiLog

*Outline of the chapter*. In Section 5.2 we introduce the syntax of TeDiLog, some preliminary definitions and a sample TeDiLog specification of a reactive system. In Section 5.3 we present the system of rules that are the basis for the operational semantics of TeDiLog. Section 5.4 is devoted to the operational and logical semantics and their equivalence. In Subsection 5.4.1 we present the operational semantics of TeDiLog. Then, in Subsection 5.4.2 we detail some sample derivations. The logical semantics is described in Subsection 5.4.3. We prove the equivalence between both semantics in Subsection 5.4.4. Finally, we discuss relevant related work in Section 5.5.

## 5.2 *The Language* TeDiLog

In this section we introduce the syntax of TeDiLog along with an illustrative example of a TeDiLog specification for a reactive system.

The syntax of TeDiLog (Figure 5.1) is an adaptation, to the usual logic programming style, of the clausal normal form previously presented for clausal temporal resolution (Section 4.2). The programming language TeDiLog is a twofold extension of propositional Horn clauses that incorporates temporal connectives in atoms and disjunctions in clause heads. It is the Temporal Disjunctive Logic programming language given in Figure 5.1, where the metavariable $A$ denotes *atom*, $L$ stands for (classical) literal, $T$ for temporal atom, $H$ for head, $B$ for body, $D$ for (disjunctive) program clause, and $G$ for goal clause. As in the previous chapter, we use the superscript $b$ varying in $\{0,1\}$ to represent a formula with or without a prefixed unary connective (in particular for the connectives $\Box$ and $\Diamond$). So that, along the rest of the chapter superscripts $b$ (from bit) range in $\{0,1\}$. These kinds of superscripts are notation, hence they are not part of the syntax. Due to the superscript $b$, the metavariable $D$ represents two kinds of clauses. The expression $\Box^b(H \leftarrow B)$, for $b = 0$, represents $H \leftarrow B$, which is called a *now-clause*, whereas for $b = 1$, it represents $\Box(H \leftarrow B)$, which is called an *always-clause*. The same classification applies to the goal clauses denoted by $G$. In particular, $\Box^b(\bot \leftarrow \top)$ represents the two possible syntactic forms of the empty clause, as now- or always-clause.

**Definition 5.2.1.** *Given a set of clauses $\Phi$, the set* $\mathsf{alw}(\Phi)$ *is formed by all the always-clauses in $\Phi$, i.e. all the clauses of the form $\Box(H \leftarrow B)$. In addition, the set* $\mathsf{now}(\Phi)$ *is* $\Phi \setminus \mathsf{alw}(\Phi)$.

A program is a set of program clauses and a goal is a set of goal clauses.

The set of atoms of a clause $C = \Box^b(A_1 \vee \ldots \vee A_m \leftarrow A'_1 \wedge \ldots \wedge A'_n)$ is the set $\{A_1, \ldots, A_m, A'_1, \ldots, A'_n\}$. We assume that there is neither repetitions nor established order in the atoms of a head or a body. An atom is said to be $\circ$-free if it is a temporal atom or a classical propositional atom. The connective $\circ$ is distributive over every other connective and, consequently, $\circ\Box\,(A_1 \vee \ldots \vee A_m \leftarrow A'_1 \wedge \ldots \wedge A'_n)$ is equivalent to $\Box\,(\circ A_1 \vee \ldots \vee \circ A_m \leftarrow \circ A'_1 \wedge \ldots \wedge \circ A'_n)$. Given a head, body, program clause or goal clause $\psi$, we denote by $\circ\psi$ the head, body, program clause or goal clause that is obtained by adding one connective $\circ$ to every atom in $\psi$. For instance, $\circ\Box\,(p \vee q \leftarrow \circ r)$ denotes $\Box\,(\circ p \vee \circ q \leftarrow \circ\circ r)$ and $\circ\Box\,(\bot \leftarrow \circ r)$ denotes $\Box\,(\bot \leftarrow \circ\circ r)$. Note that $\circ\bot$ is written just $\bot$ and $\circ\top$ is written $\top$.

A clause $\Box^b(H \leftarrow B)$ is semantically equivalent to the formula $\Box^b(H \vee \neg B)$. Consequently, not only the temporal atoms of the form $\Diamond p$ and $L\,\mathcal{U}\,p$ that occur in the head $H$ of the clause behave as eventualities, but also the temporal atoms $\Box p$ and $L\,\mathcal{R}\,p$ in the body $B$, which respectively correspond to (temporal) literals $\neg\Box p$ and $\neg(L\,\mathcal{R}\,p)$. Hence, we define the eventuality literals of a clause, on the basis of the notion of eventuality (see Definition 2.2.1).

**Definition 5.2.2.** *Let $C$ be a clause $\Box^b(A_1 \vee \ldots \vee A_m \leftarrow A'_1 \wedge \ldots \wedge A'_n)$.* $\mathsf{Lits}(C)$ *denotes the set $\{A_1, \ldots, A_m, \neg A'_1, \ldots, \neg A'_n\}$ whose elements are called the* temporal literals *of $C$. Additionally,* $\mathsf{EventLits}(C)$ *denotes the set of all the* eventuality literals *in $C$, i.e. $\{N \mid N \in$* $\mathsf{Lits}(C)$ *and $N$ is an eventuality$\}$.*
*Both notations are extended to a set of clauses $\Psi$ in the obvious manner:*

$$\mathsf{Lits}(\Psi) = \textstyle\bigcup_{C\in\Psi} \mathsf{Lits}(C) \textit{ and } \mathsf{EventLits}(\Psi) = \bigcup_{C\in\Psi} \mathsf{EventLits}(C).$$

Note that eventuality literals from clauses have one of the following four forms: $\Diamond p$, $L\,\mathcal{U}\,p$, $\neg\Box p$ and $\neg(L\,\mathcal{R}\,p)$, where $p$ is a propositional variable and $L$ a classical literal.

TeDiLog is syntactically a sublanguage of PLTL, but every PLTL-formula can be translated into TeDiLog by using, in general, new propositional variables. The translation yields an equisatisfiable set of (program and goal) clauses. For example, the PLTL-formula $\Box\neg p \leftarrow q$ (i.e. $\Box\neg p \vee \neg q$) can be translated into TeDiLog as the goal $\bot \leftarrow q \wedge \Diamond p$ but also as the set formed by the program clause $\Box r \leftarrow q$ and the goal clause $\Box\,(\bot \leftarrow r \wedge p)$ where $r$ is a fresh propositional variable. For the PLTL-formula $\Box\,(x \vee y) \leftarrow z$ we obtain the program clauses $\Box w \leftarrow z$ and $\Box\,(x \vee y \leftarrow w)$ where $w$ is a fresh propositional variable. A detailed translation method is presented in Subsection 4.2.2.

To finish this section, let us illustrate (with an example) how TeDiLog can be used to specify reactive systems and to verify properties that are satisfied by these systems. We also use the next example to compare the expressiveness of TeDiLog with the more closely related proposals in the literature.

**Example 5.2.3.** *Let us consider a system where a device ($dv$) and a system manager ($sm$) interact with each other. When the device $dv$ needs to execute a process, it sends a request $req\_dv$ to the system manager $sm$ to get permission and goes into waiting-state until the system manager $sm$ sends the acknowledgement signal $ack\_sm$ giving permission to execute the process.*

$$\Box\,(waiting\_dv\,\mathcal{U}\,ack\_sm \leftarrow req\_dv) \tag{5.1}$$

*Whenever $dv$ asks for permission, the system manager $sm$ will eventually give permission by sending the acknowledgement signal $ack\_sm$ in a later state.*

$$\Box(\circ\diamond ack\_sm \leftarrow req\_dv) \tag{5.2}$$

*Once the system manager produces the signal $ack\_sm$ (giving permission), the device $dv$ goes into working-state until it communicates the end of the process by means of the $eop\_dv$ signal.*

$$\Box(working\_dv\,\mathcal{U}\,eop\_dv \leftarrow ack\_sm) \tag{5.3}$$

*Whenever the device generates the $eop\_dv$ signal, then it will not be in working-state until it receives the $ack\_sm$ signal giving permission to execute another process.*

$$\Box(\neg working\_dv\,\mathcal{U}\,ack\_sm \leftarrow eop\_dv) \tag{5.4}$$

*From time to time, the system manager generates a control signal $ctr\_sm$*

$$\Box(\diamond ctr\_sm \leftarrow \top) \tag{5.5}$$

*The interaction generated after the control signal $ctr\_sm$ corresponds to the fact that the system manager has to regularly control whether the device is correctly connected to the system. This signal $ctr\_sm$ is always eventually followed by the signal $conn\_sm$ which is received by the device.*

$$\Box(\diamond conn\_sm \leftarrow ctr\_sm) \tag{5.6}$$

*After receiving the signal $conn\_sm$, the device $dv$ answers by sending the signal $conn\_dv$ to the system manager.*

$$\Box(\circ\diamond conn\_dv \leftarrow conn\_sm) \tag{5.7}$$

*The device $dv$ is considered to be in communicating-state ($com\_dv$) while the arising of the $conn\_dv$ signal (now or in a future moment) is guaranteed.*

$$\Box(com\_dv \leftarrow \diamond conn\_dv) \tag{5.8}$$

*We would like to remark that the clauses (5.2) and (5.5)-(5.7) cannot be expressed neither in Chronolog nor in Templog because of the eventualities in their heads. However, all of them are syntactically correct in Gabbay's Temporal Prolog. As for the clauses (5.1), (5.3) and (5.4), they contain the connective $\mathcal{U}$ which is not allowed in the above mentioned three declarative TLP languages.*

*Now, we can check whether the system specified by the* TeDiLog *clauses (5.1)-(5.8) verifies some properties such as fairness, liveness, safety, mutual exclusion, etc. This is made by writing the intended property as a* TeDiLog *goal and then checking if that goal can be inferred from the program. For example we would be interested in checking whether the device $dv$ will always keep in communicating-state. The corresponding goal would be $\{\bot \leftarrow \Box com\_dv\}$. Actually, the refutational mechanism of* TeDiLog *checks the unsatisfiability of the eventuality $\diamond\neg com\_dv$ with respect to the specification.*
*None of the just above mentioned three languages (Chronolog, Templog and Gabbay's Temporal Prolog) allows always-atoms in clause bodies, hence the previous goal is not expressible in any of these declarative TLP languages.*

$$(Res)\quad \frac{\Box^{b}(A \vee H \leftarrow B) \qquad \Box^{b'}(H' \leftarrow A \wedge B')}{\Box^{b\times b'}(H \vee H' \leftarrow B \wedge B')} \qquad b, b' \in \{0,1\}$$

*Figure 5.2:* The Resolution Rule

*The program clauses (5.1)-(5.8) can be expressed in propositional MetateM, although some translation into SNF is needed. For the resulting specification, the MetateM execution system builds a model step by step in the imperative future style. The process will stop when a loop that gives rise to an ultimately periodic model for the program is detected. If we add to the specification the SNF clauses that correspond to the goal $\bot \leftarrow \Box com\_dv$, then MetateM finitely detects the unsatisfiability of the extended specification.*

## 5.3 The Rule System

In this section, we introduce the rule system that constitutes the basis of the operational semantics of TeDiLog. This rule system is a straightforward adaptation ot the TRS-system presented in Section 4.3. Hence our system includes a *Resolution Rule*, a collection of *Temporal Rules* for decomposing temporal atoms, and two auxiliary rules respectively for *jumping to the next state* and for *subsumption*. We explain these four kinds of rules in the following four subsections.

### 5.3.1 The Resolution Rule

The TeDiLog's resolution rule $(Res)$ is a natural generalization of the classical rule for binary resolution. It is depicted in Figure 5.2 in the usual format of premises and resolvent separated by an horizontal line. The rule $(Res)$ applies to two temporal clauses such that one of the atoms in the head of one clause is in the body of the other clause. The premises can be headed or not by an always connective. By means of the product $b \times b'$ in the superscript of the resolvent, the resolvent is an always-clause if and only if both premises are always-clauses. Note that the resolvent is in general a program clause, but in particular when the premises respectively are a single-headed program clause and a goal clause, the resolvent is a goal clause.

### 5.3.2 The Temporal Rules

The temporal rules serve to transform the set of clauses according to the inductive definitions of temporal atoms. We write them as transformation rules $\Phi \mapsto \Psi$ where $\Phi$ and $\Psi$ are sets of clauses, respectivelly called the antecedent and the consequent. Temporal rules are grouped into two classes. On the one hand, the *context-free rules* are based on the usual inductive definitions of the temporal connectives. The antecedent and consequent of any context-free rule are logically equivalent. On the other hand, the *context-dependent* rules come up from a more complex inductive definition of the connective $\mathcal{U}$ (already presented in the previous chapters), and their antecedent and consequent are equisatisfiable.

$$
\begin{array}{ll}
(\mathcal{U}\, H_+) & \Box^b((p_1 \,\mathcal{U}\, p_2) \vee H \leftarrow B) \\
 & \longmapsto \{\Box^b(p_2 \vee p_1 \vee H \leftarrow B),\;\; \Box^b(p_2 \vee \circ(p_1 \,\mathcal{U}\, p_2) \vee H \leftarrow B)\}
\end{array}
$$

*Figure 5.3:* The Context-Free Rule $(\mathcal{U}\, H_+)$

$$
\begin{array}{ll}
(\mathcal{U}\, H_-) & \Box^b((\neg p_1 \,\mathcal{U}\, p_2) \vee H \leftarrow B) \\
 & \longmapsto \{\Box^b(p_2 \vee H \leftarrow p_1 \wedge B),\;\; \Box^b(p_2 \vee \circ(\neg p_1 \,\mathcal{U}\, p_2) \vee H \leftarrow B)\} \\
 & \\
(\mathcal{U}\, B_+) & \Box^b(H \leftarrow (p_1 \,\mathcal{U}\, p_2) \wedge B) \\
 & \longmapsto \{\Box^b(H \leftarrow p_2 \wedge B),\;\; \Box^b(H \leftarrow p_1 \wedge \circ(p_1 \,\mathcal{U}\, p_2) \wedge B)\} \\
 & \\
(\mathcal{U}\, B_-) & \Box^b(H \leftarrow (\neg p_1 \,\mathcal{U}\, p_2) \wedge B) \\
 & \longmapsto \{\Box^b(H \leftarrow p_2 \wedge B),\;\; \Box^b(p_1 \vee H \leftarrow \circ(\neg p_1 \,\mathcal{U}\, p_2) \wedge B)\}
\end{array}
$$

*Figure 5.4:* The Context-Free Rules $(\mathcal{U}\, H_-)$, $(\mathcal{U}\, B_+)$ and $(\mathcal{U}\, B_-)$

### ***Context-Free Rules***

In the context-free rules, the antecedent $\Phi$ is a singleton and we write directly its unique clause. The context-free rule $(\mathcal{U}\, H_+)$ –depicted in Figure 5.3– deals with an atom of the form $p_1 \,\mathcal{U}\, p_2$ that appears in the head of a clause. This rule replaces a clause of the form $\Box^b((p_1 \,\mathcal{U}\, p_2) \vee H \leftarrow B)$ with a logically equivalent set of (two) clauses according to the well-known inductive definition $p_1 \,\mathcal{U}\, p_2 \equiv p_2 \vee (p_1 \wedge \circ(p_1 \,\mathcal{U}\, p_2))$, from which the distribution law guarantees the equivalence

$$p_1 \,\mathcal{U}\, p_2 \equiv (p_2 \vee p_1) \wedge (p_2 \vee \circ(p_1 \,\mathcal{U}\, p_2)) \tag{5.9}$$

which justifies that the antecedent $(p_1 \,\mathcal{U}\, p_2) \vee H \leftarrow B$ of the rule $(\mathcal{U}\, H_+)$ is logically equivalent to the conjunction of the two clauses in its consequent: $p_2 \vee p_1 \vee H \leftarrow B$ and $p_2 \vee \circ(p_1 \,\mathcal{U}\, p_2) \vee H \leftarrow B$.

Our system also includes (see Figure 5.4) the rules $(\mathcal{U}\, H_-)$, $(\mathcal{U}\, B_+)$ and $(\mathcal{U}\, B_-)$ for the respective occurrences of $\neg p_1 \,\mathcal{U}\, p_2$ in the clause head and $p_1 \,\mathcal{U}\, p_2$ and $\neg p_1 \,\mathcal{U}\, p_2$ in the clause body. The rules $(\mathcal{U}\, H_-)$, $(\mathcal{U}\, B_+)$ and $(\mathcal{U}\, B_-)$ are respectively obtained by using the inductive definition $L \,\mathcal{U}\, p \equiv p \vee (L \wedge \circ(L \,\mathcal{U}\, p))$ for $\neg p_1 \,\mathcal{U}\, p_2$ in the clause head, and $p_1 \,\mathcal{U}\, p_2$ and $\neg p_1 \,\mathcal{U}\, p_2$ in the clause body. Additionally, the rules $(\mathcal{R}\, H_+)$, $(\mathcal{R}\, H_-)$, $(\mathcal{R}\, B_+)$ and $(\mathcal{R}\, B_-)$ in Figure 5.5 are obtained from the inductive definition $L \,\mathcal{R}\, p \equiv p \wedge (L \vee \circ(L \,\mathcal{R}\, p))$ by considering the same four kinds of occurrences of the release connective $\mathcal{R}$ in a clause.

### ***Context-Dependent Rules***

The context-dependent rules are based on an inductive definition of $\mathcal{U}$ that takes into account, not only the clauses where the temporal atom occurs, but also the remaining now-clauses in the antecedent of the rule. The rule $(\mathcal{U}\, C_+)$ in Figure 5.7 is the context-dependent rule that deals with atoms of the form $p_1 \,\mathcal{U}\, p_2$ in clause heads. This rule is obtained by a direct adaptation,

$$
\begin{array}{ll}
(\mathcal{R}\,H_+) & \Box^b((p_1\,\mathcal{R}\,p_2) \vee H \leftarrow B) \\
& \longmapsto \{\Box^b(p_2 \vee H \leftarrow B),\ \ \Box^b(p_1 \vee \circ(p_1\,\mathcal{R}\,p_2) \vee H \leftarrow B)\} \\[1em]
(\mathcal{R}\,H_-) & \Box^b((\neg p_1\,\mathcal{R}\,p_2) \vee H \leftarrow B) \\
& \longmapsto \{\Box^b(p_2 \vee H \leftarrow B),\ \ \Box^b(\circ(\neg p_1\,\mathcal{R}\,p_2) \vee H \leftarrow p_1 \wedge B)\} \\[1em]
(\mathcal{R}\,B_+) & \Box^b(H \leftarrow (p_1\,\mathcal{R}\,p_2) \wedge B) \\
& \longmapsto \{\Box^b(H \leftarrow p_2 \wedge p_1 \wedge B),\ \ \Box^b(H \leftarrow p_2 \wedge \circ(p_1\,\mathcal{R}\,p_2) \wedge B)\} \\[1em]
(\mathcal{R}\,B_-) & \Box^b(H \leftarrow (\neg p_1\,\mathcal{R}\,p_2) \wedge B) \\
& \longmapsto \{\Box^b(p_1 \vee H \leftarrow p_2 \wedge B),\ \ \Box^b(H \leftarrow p_2 \wedge \circ(\neg p_1\,\mathcal{R}\,p_2) \wedge B)\}
\end{array}
$$

*Figure 5.5:* The Context-Free Rules $(\mathcal{R}\,H_+)$, $(\mathcal{R}\,H_-)$, $(\mathcal{R}\,B_+)$ and $(\mathcal{R}\,B_-)$

$$
\begin{array}{l}
\mathsf{def}(a, L, \emptyset) = \{\Box(\bot \leftarrow a)\} \\
\mathsf{def}(a, p, \Delta) = \{\Box(p \leftarrow a)\} \cup \{\Box(H \leftarrow B \wedge a) \mid H \leftarrow B \in \neg\Delta\} \text{ if } \Delta \neq \emptyset \\
\mathsf{def}(a, \neg p, \Delta) = \{\Box(\bot \leftarrow p \wedge a)\} \cup \{\Box(H \leftarrow B \wedge a) \mid H \leftarrow B \in \neg\Delta\} \text{ if } \Delta \neq \emptyset
\end{array}
$$

*Figure 5.6:* The set of clauses $\mathsf{def}(a, L, \Delta)$

to the syntax of TeDiLog, of the rule $(\mathcal{U}\,Set)$ in Figure 4.5. The antecedent of $(\mathcal{U}\,C_+)$ must be interpreted as a partition of the whole set of clauses (on which we are applying temporal resolution) into two sets. The second set $\{\Box^{b_i}((p_1\,\mathcal{U}\,p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\}$ in the antecedent is a non-empty set of clauses that contain the same temporal atom $p_1\,\mathcal{U}\,p_2$ in the head. The first set, $\Omega$, is formed by all the remaining clauses. The now-clauses that belong to $\Omega$ form what we call *context* (see Definition 4.3.3 and Subsection 4.3.2). The crucial idea behind the context-dependent rule $(\mathcal{U}\,C_+)$ (and, hence, behind the resolution mechanism of TeDiLog) is based on the equisatisfiability result in Proposition 4.3.2. The transformation of such proposition into the syntax of TeDiLog is trivial because a TeDiLog clause of the form $\Box^b(A_1 \vee \ldots \vee A_m \leftarrow A'_1 \wedge \ldots \wedge A'_n)$ corresponds to the clause $\Box^b(A_1 \vee \ldots \vee A_m \vee \neg A'_1 \vee \ldots \vee \neg A'_n)$ in the clausal language presented in Chapter 4.

All the rules used in TeDiLog are straightforward adaptations of the rules used in the TRS resolution system. For instance, the transformation of the antecedent of $(\mathcal{U}\,C_+)$ into its consequent follows the same steps as the transformation of the antecedent of the rule $(\mathcal{U}\,Set)$ into its consequent, showed in detail in Subsection 4.3.2.

Our system also includes a similar context-dependent rule $(\mathcal{U}\,C_-)$ for $\neg p_1\,\mathcal{U}\,p_2$ in the head, which is depicted in Figure 5.8. The context-dependent rules $(\mathcal{R}\,C_+)$ and $(\mathcal{R}\,C_-)$ in Figure 5.8 are due to the fact that a release atom appearing in the body of a clause $C$ is an eventuality literal of $C$ (see Definition 5.2.2). The rules for $\mathcal{R}$ are explained by its duality with $\mathcal{U}$. Additionally, by using the definitions $\Diamond\varphi \equiv \neg\varphi\,\mathcal{U}\,\varphi$ and $\Box\varphi \equiv \neg\varphi\,\mathcal{R}\,\varphi$, the context-free rules $(\Diamond H_+)$, $(\Diamond B_+)$, $(\Box H_+)$ and $(\Box B_+)$ and the context-dependent rules $(\Diamond C_+)$ and $(\Box C_+)$ are derived.

$$\begin{array}{lll}
(\mathcal{U}C_+) & \Omega \cup \{\Box^{b_i}((p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\} & \\
& \longmapsto\ \Omega\ \cup & \{p_2 \vee p_1 \vee H_i \leftarrow B_i,\ p_2 \vee \circ(a \mathcal{U} p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\} \\
& \qquad\quad \cup & \mathsf{def}(a, p_1, \mathsf{now}(\Omega)) \\
& \qquad\quad \cup & \{\Box^{b_i}(\circ(p_1 \mathcal{U} p_2) \vee \circ H_i \leftarrow \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}$$

where $n \geq 1$, $a \in \mathsf{Prop}$ is fresh and $\mathsf{def}(a, p_1, \mathsf{now}(\Omega))$ is defined in Figure 5.6.

*Figure 5.7:* The Context-Dependent Rule $(\mathcal{U}C_+)$

$$\begin{array}{lll}
(\mathcal{U}C_-) & \Omega \cup \{\Box^{b_i}((\neg p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\} & \\
& \longmapsto\ \Omega\ \cup & \{p_2 \vee H_i \leftarrow p_1 \wedge B_i,\ p_2 \vee \circ(a \mathcal{U} p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\} \\
& \qquad\quad \cup & \mathsf{def}(a, \neg p_1, \mathsf{now}(\Omega)) \\
& \qquad\quad \cup & \{\Box^{b_i}(\circ(\neg p_1 \mathcal{U} p_2) \vee \circ H_i \leftarrow \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\} \\
& & \\
(\mathcal{R}C_+) & \Omega \cup \{\Box^{b_i}(H_i \leftarrow (p_1 \mathcal{R} p_2) \wedge B_i) \mid 1 \leq i \leq n\} & \\
& \longmapsto\ \Omega\ \cup & \{H_i \leftarrow p_2 \wedge p_1 \wedge B_i,\ H_i \leftarrow p_2 \wedge \circ(\neg a \mathcal{R} p_2) \wedge B_i \mid 1 \leq i \leq n\} \\
& \qquad\quad \cup & \mathsf{def}(a, \neg p_1, \mathsf{now}(\Omega)) \\
& \qquad\quad \cup & \{\Box^{b_i}(\circ H_i \leftarrow \circ(p_1 \mathcal{R} p_2) \wedge \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\} \\
& & \\
(\mathcal{R}C_-) & \Omega \cup \{\Box^{b_i}(H_i \leftarrow (\neg p_1 \mathcal{R} p_2) \wedge B_i) \mid 1 \leq i \leq n\} & \\
& \longmapsto\ \Omega\ \cup & \{p_1 \vee H_i \leftarrow p_2 \wedge B_i,\ H_i \leftarrow p_2 \wedge \circ(\neg a \mathcal{R} p_2) \wedge B_i \mid 1 \leq i \leq n\} \\
& \qquad\quad \cup & \mathsf{def}(a, p_1, \mathsf{now}(\Omega)) \\
& \qquad\quad \cup & \{\Box^{b_i}(\circ H_i \leftarrow \circ(\neg p_1 \mathcal{R} p_2) \wedge \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}$$

where $n \geq 1$, $a \in \mathsf{Prop}$ is fresh and $\mathsf{def}(a, L, \mathsf{now}(\Omega))$ is defined is in Figure 5.6.

*Figure 5.8:* The Context-Dependent Rules $(\mathcal{U}C_-)$, $(\mathcal{R}C_+)$ and $(\mathcal{R}C_-)$

These derived rules are depicted in Figure 5.10.

### *5.3.3 The Rule for Jumping to the Next State*

The rule $(Unx)$ in Figure 5.11 applies to a pair formed by a program and a goal, giving a new pair of program and goal. The expression $\mathsf{unnext}(\Psi)$ stands for the set of all clauses that should be satisfied at the next state of a state that satisfies the set of clauses $\Psi$. Note that the definition of the function unnext implicitly uses the equivalence $\Box\varphi \equiv \varphi \wedge \Box\circ\varphi$ and also that the unnext target of a program (resp. goal) is also a program (resp. goal). It is worth remembering that $\top$ and $\bot$ respectively represent the empty body and the empty head, and it holds that every atom in $\top$ and $\bot$ is of the form $\circ A$. For example, $\mathsf{unnext}(\{\Box(\circ r \leftarrow \top), \Box(q \leftarrow \top)\})$ is the set $\{\Box(\circ r \leftarrow \top), \Box(q \leftarrow \top), r \leftarrow \top\}$.

$$\mathsf{def}(a, \mathsf{now}(\Omega)) = \begin{cases} \{\Box(\bot \leftarrow a)\} & \text{if } \mathsf{now}(\Omega) = \emptyset \\ \{\Box(H \leftarrow B \wedge a) \mid H \leftarrow B \in \neg\mathsf{now}(\Omega)\} & \text{otherwise} \end{cases}$$

*Figure 5.9:* The set of clauses $\mathsf{def}(a, \mathsf{now}(\Omega))$

$$\begin{array}{ll}
(\Diamond H_+) & \Box^b(\Diamond p \vee H \leftarrow B) \longmapsto \{\Box^b(p \vee \circ\Diamond p \vee H \leftarrow B)\} \\
(\Diamond B_+) & \Box^b(H \leftarrow \Diamond p \wedge B) \longmapsto \{\Box^b(H \leftarrow p \wedge B),\ \Box^b(H \leftarrow \circ\Diamond p \wedge B)\} \\
(\Box H_+) & \Box^b(\Box p \vee H \leftarrow B) \longmapsto \{\Box^b(p \vee H \leftarrow B),\ \Box^b(\circ\Box p \vee H \leftarrow B)\} \\
(\Box B_+) & \Box^b(H \leftarrow \Box p \wedge B) \longmapsto \{\Box^b(H \leftarrow p \wedge \circ\Box p \wedge B)\}
\end{array}$$

$$\begin{array}{lllll}
(\Diamond C_+) & \multicolumn{4}{l}{\Omega \cup \{\Box^{b_i}(\Diamond p \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\}} \\
& \longmapsto & \Omega & \cup & \{p \vee \circ(a\,\mathcal{U}\,p) \vee H_i \leftarrow B_i \ \mid\ 1 \leq i \leq n\} \\
& & & \cup & \mathsf{def}(a, \mathsf{now}(\Omega)) \\
& & & \cup & \{\Box^{b_i}(\circ\Diamond p \vee \circ H_i \leftarrow \circ B_i) \ \mid\ b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}$$

$$\begin{array}{lllll}
(\Box C_+) & \multicolumn{4}{l}{\Omega \cup \{\Box^{b_i}(H_i \leftarrow \Box p \wedge B_i) \mid 1 \leq i \leq n\}} \\
& \longmapsto & \Omega & \cup & \{H_i \leftarrow p \wedge \circ(\neg a\,\mathcal{R}\,p) \wedge B_i \mid 1 \leq i \leq n\} \\
& & & \cup & \mathsf{def}(a, \mathsf{now}(\Omega)) \\
& & & \cup & \{\Box^{b_i}(\circ H_i \leftarrow \circ\Box p \wedge \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}$$

where $n \geq 1$, $a \in \mathsf{Prop}$ is fresh and $\mathsf{def}(a, \mathsf{now}(\Omega))$ is defined in Figure 5.9

*Figure 5.10:* Derived Rules for $\Diamond$ and $\Box$

### *5.3.4 The Subsumption Rule*

The rule $(Sbm)$ is formulated in Figure 5.12. Regarding the clauses in the antecedent, it is said that the clause $\Box^b(H \leftarrow B)$ is subsumed by the clause $\Box^b(H' \leftarrow B')$.
Our resolution mechanism requires $(Sbm)$ for completeness. Actually, subsumption is used in Lemma 5.4.11, which is used in the proof of Proposition 5.4.24 and allows us to prove Theorem 5.4.28.

## *5.4* TeDiLog *Semantics*

In this section we summarize our results on TeDiLog semantics. The first subsection is devoted to the operational semantics that is formalized by means of the algorithm in Figure 5.13. The second subsection shows three sample derivations. In the third subsection we define the logical semantics. Finally, in the last subsection we prove the equivalence between the operational and the logical semantics.

$$(Unx) \quad (\Pi, \Gamma) \longmapsto (\mathsf{unnext}(\Pi), \mathsf{unnext}(\Gamma))$$
$$\text{where } \mathsf{unnext}(\Psi) = \mathsf{alw}(\Psi) \cup \{H \leftarrow B \mid \Box^b(\circ H \leftarrow \circ B) \in \Psi\}$$

*Figure 5.11:* The Rule $(Unx)$

$$(Sbm) \quad \{\Box^b(H \leftarrow B), \Box^b(H' \leftarrow B')\} \longmapsto \{\Box^b(H' \leftarrow B')\}$$
$$\text{where } H' \subseteq H \text{ and } B' \subseteq B.$$

*Figure 5.12:* The Rule $(Sbm)$

### 5.4.1 Operational Semantics

In this subsection we formulate the operational semantics of TeDiLog. We refer to the refutation procedure underlying TeDiLog as IFT-resolution (for *Invariant-Free Temporal* resolution). Every step of an IFT-derivation consists in applying one of the rules presented in Section 5.3. However, as in the tableau method TTM and the resolution system TRS, the nondeterministic application of those rules does not guarantee completeness. In Figure 5.13 we show the IFT-resolution procedure that applies the rules in Section 5.3 in a more (not fully) deterministic way that is complete. The algorithm in Figure 5.13 is an adaptation of the algorithm $\mathcal{SR}$ in Figure 4.10 to the language TeDiLog. Consequently, this subsection is an adaptation of Subsection 4.6.1 into the language TeDiLog.

The IFT-resolution procedure constructs an IFT-*derivation* from an input program $\Pi$ and an input goal $\Gamma$ that we call $\mathcal{D}(\Pi, \Gamma)$ and consists of a (possibly infinite) sequence

$$S_0 \mapsto\!\!\!\Rightarrow S_1 \mapsto\!\!\!\Rightarrow S_2 \mapsto\!\!\!\Rightarrow \dots$$

where each $S_j$ is a finite sequence of pairs

$$(\Pi_j^0, \Gamma_j^0) \mapsto (\Pi_j^1, \Gamma_j^1) \mapsto \dots \mapsto (\Pi_j^{h_j}, \Gamma_j^{h_j})$$

such that

*(a)* $(\Pi_0^0, \Gamma_0^0) = (\Pi, \Gamma)$

*(b)* $(\Pi_k^0, \Gamma_k^0) = (\mathsf{unnext}(\Pi_{k-1}^{h_{k-1}}), \mathsf{unnext}(\Gamma_{k-1}^{h_{k-1}}))$ for every $k \geq 1$

*(c)* Every pair $(j, i)$ such that $j \geq 0$ and $i \in \{1, \dots, h_j\}$ satisfies one of the following two conditions

*(i)* $\Pi_j^i \cup \Gamma_j^i = \Pi_j^{i-1} \cup \Gamma_j^{i-1} \cup \{\Box^b(H \leftarrow B)\}$ where $\Box^b(H \leftarrow B)$ is the resolvent obtained by applying the rule $(Res)$ to some pair of clauses in $\Pi_j^{i-1} \cup \Gamma_j^{i-1}$

*(ii)* $\Pi_j^i \cup \Gamma_j^i = ((\Pi_j^{i-1} \cup \Gamma_j^{i-1}) \backslash \Sigma) \cup \Psi$ where $\Sigma \subseteq (\Pi_j^{i-1} \cup \Gamma_j^{i-1})$ and $\Sigma \mapsto \Psi$ according to a temporal rule or the subsumption rule.

```
(Π_0^0, Γ_0^0) := (Π, Γ);  i := 0;  j := 0;
sel_ev_set_0 := fair_select(Π_0^0, Γ_0^0);
loop
    if sel_ev_set_i ≠ ∅
      then (Π_i^1, Γ_i^1, sel_ev_set_i^*) := apply_ctx_dep(Π_i^0, Γ_i^0, sel_ev_set_i);  j := 1;
      else sel_ev_set_i^* := ∅;
    end if;
    (Π_i^*, Γ_i^*) := supported_free_close(Π_i^j, Γ_i^j);
    if □^b(⊥ ← ⊤) ∈ Π_i^* ∪ Γ_i^* then exit; end if;
    (Π_{i+1}^0, Γ_{i+1}^0) := (unnext(Π_i^*), unnext(Γ_i^*));
    if sel_ev_set_i^* ∩ EventLits(Π_{i+1}^0 ∪ Γ_{i+1}^0) = ∅
      then sel_ev_set_{i+1} := fair_select(Π_{i+1}^0, Γ_{i+1}^0);
      else sel_ev_set_{i+1} := sel_ev_set_i^*;
    end if;
    i := i + 1; j := 0;
end loop;
```

*Figure 5.13:* The IFT-Resolution Procedure

Note that we use two different symbols ($\mapsto$ and $\Mapsto$) to highlight the difference between applying the rule $(Unx)$ and any other rule. We say that an IFT-derivation is a *local derivation* if it does not contain any application of the rule $(Unx)$. Each sequence $S_j$ is a local derivation and $(Unx)$ serves to jump from each $S_j$ to the next sequence $S_{j+1}$. In other words, the application of $(Unx)$ yields each $(\Pi^0_{j+1}, \Gamma^0_{j+1})$ from each $(\Pi^{h_j}_j, \Gamma^{h_j}_j)$.

The IFT-resolution procedure first initializes (see line 1 in Figure 5.13) the pair $(\Pi^0_0, \Gamma^0_0)$ with the input pair $(\Pi, \Gamma)$. Then, the procedure iterates extending the derivation $\mathcal{D}(\Pi, \Gamma)$ with new pairs and stopping only if the empty clause is obtained (line 9). In this case, the resolution proof $\mathcal{D}(\Pi, \Gamma)$ is called an IFT-*refutation*. The IFT-resolution procedure uses a marking strategy for applying exactly one context-dependent rule between each two consecutive applications of the rule $(Unx)$.[2] For that, it keeps two variables $\mathsf{sel\_ev\_set}_i$ and $\mathsf{sel\_ev\_set}^*_i$ for every $i \geq 0$. Both variables, $\mathsf{sel\_ev\_set}_i$ and $\mathsf{sel\_ev\_set}^*_i$, take as value the empty set or a singleton that contains an eventuality literal, depending on whether $\mathsf{EventLits}(\Pi^0_i \cup \Gamma^0_i)$ – see Definition 5.2.2– is empty or not, respectively. The variable $\mathsf{sel\_ev\_set}_i$ stands for the selected eventuality literal $N$ in $(\Pi^0_i, \Gamma^0_i)$, whereas $\mathsf{sel\_ev\_set}^*_i$ corresponds to the eventuality literal obtained from $N$ by the application of the corresponding context-dependent rule, which remains selected in all pairs from $(\Pi^1_i, \Gamma^1_i)$ to $(\Pi^{h_i}_i, \Gamma^{h_i}_i)$. Consequently, in line 2 (Figure 5.13), the variable $\mathsf{sel\_ev\_set}_0$ is initialized with a singleton that contains a fairly selected temporal literal from $\mathsf{EventLits}(\Pi^0_0 \cup \Gamma^0_0)$ whenever $\mathsf{EventLits}(\Pi^0_0 \cup \Gamma^0_0)$ is non-empty. On the contrary, if $\mathsf{EventLits}(\Pi^0_0 \cup \Gamma^0_0)$ is empty, the variable $\mathsf{sel\_ev\_set}_0$ is initialized with the empty set (line 2). The expression $\mathsf{fair\_select}(\Pi^0_h, \Gamma^0_h)$ encapsulates the fair selection of a literal from $\mathsf{EventLits}(\Pi^0_h \cup \Gamma^0_h)$, where fairness means that a literal that belongs to every set in a sequence of the form

[2] Whenever there is at least one eventuality literal, exactly one is selected as the designated eventuality of the corresponding context-dependent rule. Otherwise, no context-dependent rule is applicable.

$$\mathsf{EventLits}(\Pi^0_g \cup \Gamma^0_g), \mathsf{EventLits}(\Pi^0_{g+1} \cup \Gamma^0_{g+1}), \mathsf{EventLits}(\Pi^0_{g+2} \cup \Gamma^0_{g+2}), \ldots$$

cannot remain indefinitely unselected in the derivation $S_g \mapsto S_{g+1} \mapsto S_{g+2} \mapsto \ldots$.

In addition to the above explained marking strategy, IFT-resolution requires a controlled kind of saturation (with respect to the rules introduced in Section 5.3) before jumping from a sequence $S_j$ to the next sequence $S_{j+1}$, which is also needed for completeness. Actually, every pair $(\Pi_j^{h_j} \cup \Gamma_j^{h_j})$ is IFT-closed (or saturated) in the sense given by the following definition.

**Definition 5.4.1.** *Let $\Pi$ be a program and $\Gamma$ a goal. The pair $(\Pi, \Gamma)$ is IFT-closed if and only if it satisfies the following four conditions:*

*(a) The set of atoms of the clauses in $\Pi \cup \Gamma$ is exclusively formed by atoms in $\mathsf{Prop}$ and atoms of the form $\circ A$.*

*(b) The subsumption rule $(Sbm)$ cannot be applied to $(\Pi, \Gamma)$.*

*(c) Every clause that can be obtained by applying the rule $(Res)$ to a clause in $\Pi$ and a clause in $\Gamma$, is already in $(\Pi, \Gamma)$ or it is subsumed by some clause in $(\Pi, \Gamma)$.*

*(d) Every clause of the form $\Box^b(\circ H \leftarrow \circ B)$ that can be obtained by means of a local derivation where in each derivation step the rule $(Res)$ is applied to two program clauses, is already in $(\Pi, \Gamma)$ or it is subsumed by some clause in $(\Pi, \Gamma)$.*

Items *(c)* and *(d)* represent two particular forms of the well-known set-of-support restriction of resolution[3] (see e.g. Section 2.6 in [115] and [34]). Note that, by *(c)*, the pair $(\Pi \cup \Gamma)$ is saturated with all the resolvents that can be obtained from a program clause and a goal clause. We call *goal-resolution* to every application of the rule $(Res)$ related to *(c)*. However, by *(d)*, the program $\Pi$ is saturated with all the resolvents $R$ of two program clauses such that every atom in $R$ is preceded by the connective $\circ$. We call *nx-resolution* to every application of the rule $(Res)$ related to *(d)*. The need of *nx-resolution* is illustrated in Example 5.4.5.

**Definition 5.4.2.** *Let $\Pi$ be a program and $\Gamma$ a goal. We denote by $(\Pi^*, \Gamma^*)$ any pair such that there exists a local derivation $(\Pi, \Gamma) \mapsto \ldots \mapsto (\Pi^*, \Gamma^*)$ and either $\Box^b(\bot \leftarrow \top) \in \Gamma^*$ or $(\Pi^*, \Gamma^*)$ is IFT-closed.*

Consequently, the lines 4 to 8 (in Figure 5.13) serve to extend the derivation from $(\Pi_i^0, \Gamma_i^0)$ to $(\Pi_i^*, \Gamma_i^*)$. First, by lines 4-7, if there is a selected eventuality literal, i.e., if $\mathsf{sel\_ev\_set}_i \neq \emptyset$, then the corresponding context-dependent rule is applied considering that $\Omega = (\Pi_i^0 \cup \Gamma_i^0) \setminus \{C \in (\Pi_i^0 \cup \Gamma_i^0) \mid \mathsf{EventLits}(C) \cap \mathsf{sel\_ev\_set}_i \neq \emptyset\}$. The value of $\mathsf{sel\_ev\_set}_i^*$ is the singleton that contains the new eventuality literal that is introduced by the applied context-dependent rule (i.e. the eventuality literal that appears in the consequent of the applied context-dependent rule preceded by a $\circ$ connective). If there is no selected literal, none of the context-dependent rules is applicable in the current iteration step and, additionally, the value of $\mathsf{sel\_ev\_set}_i^*$ is the empty set. Then, in line 8, denoted as $\mathsf{supported\_free\_close}$, the context-free rules, the resolution rule and the subsumption rule are repeatedly and nondeterministically applied until either an IFT-refutation or an IFT-closed pair (see Definition 5.4.1) is obtained.

[3] Also known as *set-of-support strategy for resolution.*

**Definition 5.4.3.** *Let* $(\Pi, \Gamma)$ *be a pair where* $\Pi$ *is a program and* $\Gamma$ *a goal, the non-deterministic operation that yields* $(\Pi^*, \Gamma^*)$ *from* $(\Pi, \Gamma)$ *without any application of the context-dependent rules is denoted by* supported_free_close.

In the algorithm presented in Figure 5.13 we use the procedure supported_free_close that implements the operation supported_free_close during the construction of a derivation.

Once an IFT-closed pair is obtained –if a refutation is not found in line 9– the rule $(Unx)$ is applied (line 10). Then, an eventuality literal that belongs to EventLits$(\Pi^0_{i+1} \cup \Gamma^0_{i+1})$, is fairly selected for the next iteration step (lines 11-14). If sel_ev_set$^*_i$ is empty or if the literal in sel_ev_set$^*_i$ does not appear in EventLits$(\Pi^0_{i+1} \cup \Gamma^0_{i+1})$ (line 11), then a new literal that belongs to EventLits$(\Pi^0_{i+1} \cup \Gamma^0_{i+1})$ is fairly selected for the next iteration step (line 12). Otherwise, the literal in sel_ev_set$^*_i$ is kept as the selected one for the next iteration step (line 13).

### *5.4.2 Examples*

In this section we present three detailed examples that illustrate the IFT-resolution procedure. In Example 5.4.4 we simply show how IFT-resolution deals with eventualities. The Example 5.4.5 illustrates the need of nx-resolution (Definition 5.4.1 *(d)*). Finally, Example 5.4.7 shows that the order in which eventuality literals a selected –by means of the fair_select operation– does not necessarily determine the order in which eventuality literals are fulfilled. Moreover, this example also serves to illustrate that the fulfillment of eventualities is handled by IFT-resolution without backtracking. The three sample derivations are showed in the respective figures, where we indicate which rule is applied and we underline the clauses designated by the rule application, except for the rule $(Unx)$. The values of sel_ev_set$_i$ and sel_ev_set$^*_i$ are pointed out too.

**Example 5.4.4.** *We consider the program* $\Pi = \{q \mathcal{U} r \leftarrow \top\}$ *and the goal* $\Gamma = \{\Box(\bot \leftarrow r)\}$. *The goal clause is equivalent to the formula* $\Box\neg r$ *and* $\Pi \cup \Gamma$ *is unsatisfiable. In Figure 5.14 we show an* IFT*-refutation for* $(\Pi, \Gamma)$. *First,* $\Pi^0_0$ *and* $\Gamma^0_0$ *are respectively initialized as* $\Pi$ *and* $\Gamma$. *Since* $q \mathcal{U} r$ *is the only eventuality literal in a clause that belongs to* $\Pi \cup \Gamma$, *it is selected. Therefore* sel_ev_set$_0 = \{q \mathcal{U} r\}$. *We apply the rule* $(\mathcal{U} C_+)$ *to* $\Pi^0_0 \cup \Gamma^0_0$ *with selected literal* $q \mathcal{U} r$ *and empty context. Hence, we obtain the new program clauses* $r \vee q \leftarrow \top$ *and* $r \vee \circ(a \mathcal{U} r) \leftarrow \top$ *and the goal clause* $\Box(\bot \leftarrow a)$, *where* $a$ *is a fresh variable. Since the context is empty (its negation is* $\bot$*), the goal clause* $\Box(\bot \leftarrow a)$ *gives meaning to the fresh variable* $a$. *The new atom* $a \mathcal{U} r$ *is the new selected literal, i.e,* sel_ev_set$^*_0 = \{a \mathcal{U} r\}$. *Then the resolution rule and the subsumption rule are applied twice, and the* IFT*-closed pair* $(\Pi^5_0, \Gamma^5_0)$ *is obtained. These applications of the rules* $(Res)$ *and* $(Sbm)$ *correspond to the operation* supported_free_close *(Definition 5.4.3). Since a refutation cannot be obtained in this state, the application of the rule* $(Unx)$ *serves to jump to the next state, generating* $\Pi^0_1$ *and* $\Gamma^0_1$. *Since the atom* $a \mathcal{U} r$ *appears as eventuality literal in a clause that belongs to* $\Pi^0_1 \cup \Gamma^0_1$, *it is kept as selected literal, i.e.,* sel_ev_set$_1 = \{a \mathcal{U} r\}$. *Now the rule* $(\mathcal{U} C_+)$ *is applied to the set* $\Pi^0_1 \cup \Gamma^0_1$ *with selected literal* $a \mathcal{U} r$ *and empty context. Then, we obtain two new program clauses,* $r \vee a \leftarrow \top$ *and* $r \vee \circ(b \mathcal{U} r) \leftarrow \top$, *and the goal clause* $\Box(\bot \leftarrow b)$, *where* $b$ *is a fresh variable. Now* sel_ev_set$^*_1 = \{b \mathcal{U} r\}$. *Two additional applications of the rule* $(Res)$ *–that correspond to the operation* supported_free_close– *yield the empty clause* $\bot \leftarrow \top$.

In the next example we illustrate why *nx-resolution* (Definition 5.4.1 *(d)*) is necessary for completeness. This example is an adaptation, to TeDiLog, of the Example 4.6.3 (Figure 4.11).

| | | | |
|---|---|---|---|
| $\Pi_0^0 = \{\underline{q\,\mathcal{U}\,r \leftarrow \top}\}$ | $\Gamma_0^0 = \{\Box(\bot \leftarrow r)\}$ | $(\mathcal{U}C_+)$ | $\mathsf{sel\_ev\_set}_0 = \{q\,\mathcal{U}\,r\}$ |
| $\Pi_0^1 = \{\underline{r \vee q \leftarrow \top}$,<br>$r \vee \circ(a\,\mathcal{U}\,r) \leftarrow \top\}$ | $\Gamma_0^1 = \{\underline{\Box(\bot \leftarrow r)}$,<br>$\Box(\bot \leftarrow a)\}$ | $(Res)$ | $\mathsf{sel\_ev\_set}_0^* = \{a\,\mathcal{U}\,r\}$ |
| $\Pi_0^2 = \{\underline{r \vee q \leftarrow \top}$,<br>$r \vee \circ(a\,\mathcal{U}\,r) \leftarrow \top$,<br>$\underline{q \leftarrow \top}\}$ | $\Gamma_0^2 = \{\Box(\bot \leftarrow r)$,<br>$\Box(\bot \leftarrow a)\}$ | $(Sbm)$ | |
| $\Pi_0^3 = \{\underline{r \vee \circ(a\,\mathcal{U}\,r) \leftarrow \top}$,<br>$q \leftarrow \top\}$ | $\Gamma_0^3 = \{\underline{\Box(\bot \leftarrow r)}$,<br>$\Box(\bot \leftarrow a)\}$ | $(Res)$ | |
| $\Pi_0^4 = \{\underline{r \vee \circ(a\,\mathcal{U}\,r) \leftarrow \top}$,<br>$q \leftarrow \top$,<br>$\underline{\circ(a\,\mathcal{U}\,r) \leftarrow \top}\}$ | $\Gamma_0^4 = \{\Box(\bot \leftarrow r)$,<br>$\Box(\bot \leftarrow a)\}$ | $(Sbm)$ | |
| $\Pi_0^5 = \{q \leftarrow \top$,<br>$\circ(a\,\mathcal{U}\,r) \leftarrow \top\}$ | $\Gamma_0^5 = \{\Box(\bot \leftarrow r)$,<br>$\Box(\bot \leftarrow a)\}$ | $(Unx)$ | |
| $\Pi_1^0 = \{\underline{a\,\mathcal{U}\,r \leftarrow \top}\}$ | $\Gamma_1^0 = \{\Box(\bot \leftarrow r)$,<br>$\Box(\bot \leftarrow a)\}$ | $(\mathcal{U}C_+)$ | $\mathsf{sel\_ev\_set}_1 = \{a\,\mathcal{U}\,r\}$ |
| $\Pi_1^1 = \{\underline{r \vee a \leftarrow \top}$,<br>$r \vee \circ(b\,\mathcal{U}\,r) \leftarrow \top\}$ | $\Gamma_1^1 = \{\underline{\Box(\bot \leftarrow r)}$,<br>$\Box(\bot \leftarrow a)$,<br>$\Box(\bot \leftarrow b)\}$ | $(Res)$ | $\mathsf{sel\_ev\_set}_1^* = \{b\,\mathcal{U}\,r\}$ |
| $\Pi_1^2 = \{r \vee a \leftarrow \top$,<br>$r \vee \circ(b\,\mathcal{U}\,r) \leftarrow \top$,<br>$\underline{a \leftarrow \top}\}$ | $\Gamma_1^2 = \{\Box(\bot \leftarrow r)$,<br>$\underline{\Box(\bot \leftarrow a)}$,<br>$\underline{\Box(\bot \leftarrow b)}\}$ | $(Res)$ | |
| $\Pi_1^3 = \{r \vee a \leftarrow \top$,<br>$r \vee \circ(b\,\mathcal{U}\,r) \leftarrow \top$,<br>$a \leftarrow \top\}$ | $\Gamma_1^3 = \{\Box(\bot \leftarrow r)$,<br>$\Box(\bot \leftarrow a)$,<br>$\Box(\bot \leftarrow b)$,<br>$\bot \leftarrow \top\}$ | | |

*Figure 5.14:* IFT-Refutation for $\Pi = \{q\,\mathcal{U}\,r \leftarrow \top\}$ and $\Gamma = \{\Box(\bot \leftarrow r)\}$

| | | | | | |
|---|---|---|---|---|---|
| $\Pi_0^0 =$ | $\{q \leftarrow \top,$<br>$\Box(\circ q \leftarrow q)\}$ | $\Gamma_0^0 =$ | $\{\underline{\bot \leftarrow \Box q}\}$ | $(\Box C_+)$ | $\mathsf{sel\_ev\_set}_0 = \{\neg\Box q\}$ |
| $\Pi_0^1 =$ | $\{\underline{q \leftarrow \top},$<br>$\Box(\circ q \leftarrow q)\}$ | $\Gamma_0^1 =$ | $\{\underline{\bot \leftarrow q \wedge \circ(\neg a\,\mathcal{R}\,q)},$<br>$\Box(\bot \leftarrow q \wedge a)\}$ | $(Res)$ | $\mathsf{sel\_ev\_set}_0^* = \{\neg(\neg a\,\mathcal{R}\,q)\}$ |
| $\Pi_0^2 =$ | $\{q \leftarrow \top,$<br>$\Box(\circ q \leftarrow q)\}$ | $\Gamma_0^2 =$ | $\{\underline{\bot \leftarrow q \wedge \circ(\neg a\,\mathcal{R}\,q)},$<br>$\Box(\bot \leftarrow q \wedge a),$<br>$\underline{\bot \leftarrow \circ(\neg a\,\mathcal{R}\,q)}\}$ | $(Sbm)$ | |
| $\Pi_0^3 =$ | $\{\underline{q \leftarrow \top},$<br>$\Box(\circ q \leftarrow q)\}$ | $\Gamma_0^3 =$ | $\{\underline{\Box(\bot \leftarrow q \wedge a)},$<br>$\bot \leftarrow \circ(\neg a\,\mathcal{R}\,q)\}$ | $(Res)$ | |
| $\Pi_0^4 =$ | $\{\underline{q \leftarrow \top},$<br>$\underline{\Box(\circ q \leftarrow q)}\}$ | $\Gamma_0^4 =$ | $\{\Box(\bot \leftarrow q \wedge a),$<br>$\bot \leftarrow \circ(\neg a\,\mathcal{R}\,q),$<br>$\bot \leftarrow a\}$ | $(Res)$ | |
| $\Pi_0^5 =$ | $\{q \leftarrow \top,$<br>$\Box(\circ q \leftarrow q),$<br>$\circ q \leftarrow \top\}$ | $\Gamma_0^5 =$ | $\{\Box(\bot \leftarrow q \wedge a),$<br>$\bot \leftarrow \circ(\neg a\,\mathcal{R}\,q),$<br>$\bot \leftarrow a\}$ | $(Unx)$ | |
| $\Pi_1^0 =$ | $\{\Box(\circ q \leftarrow q),$<br>$q \leftarrow \top\}$ | $\Gamma_1^0 =$ | $\{\Box(\bot \leftarrow q \wedge a),$<br>$\underline{\bot \leftarrow \neg a\,\mathcal{R}\,q}\}$ | $(\mathcal{R}\,C_-)$ | $\mathsf{sel\_ev\_set}_1 = \{\neg(\neg a\,\mathcal{R}\,q)\}$ |
| $\Pi_1^1 =$ | $\{\Box(\circ q \leftarrow q),$<br>$\underline{q \leftarrow \top},$<br>$a \leftarrow q,$<br>$\Box(a \leftarrow b)\}$ | $\Gamma_1^1 =$ | $\{\underline{\Box(\bot \leftarrow q \wedge a)},$<br>$\bot \leftarrow q \wedge \circ(\neg b\,\mathcal{R}\,q),$<br>$\Box(\bot \leftarrow q \wedge b)\}$ | $(Res)$ | $\mathsf{sel\_ev\_set}_1^* = \{\neg(\neg b\,\mathcal{R}\,q)\}$ |
| $\Pi_1^2 =$ | $\{\Box(\circ q \leftarrow q),$<br>$q \leftarrow \top,$<br>$\underline{a \leftarrow q},$<br>$\Box(a \leftarrow b)\}$ | $\Gamma_1^2 =$ | $\{\Box(\bot \leftarrow q \wedge a),$<br>$\bot \leftarrow q \wedge \circ(\neg b\,\mathcal{R}\,q),$<br>$\Box(\bot \leftarrow q \wedge b),$<br>$\underline{\bot \leftarrow a}\}$ | $(Res)$ | |
| $\Pi_1^3 =$ | $\{\Box(\circ q \leftarrow q),$<br>$\underline{q \leftarrow \top},$<br>$a \leftarrow q,$<br>$\Box(a \leftarrow b)\}$ | $\Gamma_1^3 =$ | $\{\Box(\bot \leftarrow q \wedge a),$<br>$\bot \leftarrow q \wedge \circ(\neg b\,\mathcal{R}\,q),$<br>$\Box(\bot \leftarrow q \wedge b),$<br>$\bot \leftarrow a,$<br>$\underline{\bot \leftarrow q}\}$ | $(Res)$ | |
| $\Pi_1^4 =$ | $\{\Box(\circ q \leftarrow q),$<br>$q \leftarrow \top,$<br>$a \leftarrow q,$<br>$\Box(a \leftarrow b)\}$ | $\Gamma_1^4 =$ | $\{\Box(\bot \leftarrow q \wedge a),$<br>$\bot \leftarrow q \wedge \circ(\neg b\,\mathcal{R}\,q),$<br>$\Box(\bot \leftarrow q \wedge b),$<br>$\bot \leftarrow a,$<br>$\bot \leftarrow q,$<br>$\bot \leftarrow \top\}$ | | |

*Figure 5.15:* IFT-refutation for $\Pi = \{q \leftarrow \top, \Box(\circ q \leftarrow q)\}$ and $\Gamma = \{\bot \leftarrow \Box q\}$

Now we use the propositional variable $q$ instead of the variable $p$ that is used in Example 4.6.3.

**Example 5.4.5.** *Let us consider the program* $\Pi = \{q \leftarrow \top, \Box(\circ q \leftarrow q)\}$ *and the goal* $\Gamma = \{\bot \leftarrow \Box q\}$. *The set of clauses* $\Pi \cup \Gamma$ *is unsatisfiable. The* IFT*-refutation for* $(\Pi, \Gamma)$ *is shown in Figure 5.15. The goal clause* $\bot \leftarrow \Box q$ *contains the only eventuality literal,* $\neg\Box q$, *in* $(\Pi, \Gamma)$. *Hence* $\mathsf{sel\_ev\_set}_0 = \{\neg\Box q\}$ *and the application of the rule* $(\Box C_+)$ *with context* $\{q \leftarrow \top\}$ *generates the goal clauses* $\bot \leftarrow q \wedge \circ(\neg a\, \mathcal{R}\, q)$ *and* $\Box(\bot \leftarrow q \wedge a)$, *where* $a$ *is a new propositional variable. Additionally, we have that* $\mathsf{sel\_ev\_set}^*_0 = \{\neg(\neg a\, \mathcal{R}\, q)\}$. *Now the operation* supported_free_close *is carried out, which consists in three applications of* $(Res)$ *and one application of* $(Sbm)$. *By applying the resolution rule to the program clause* $q \leftarrow \top \in \Pi^1_0$ *and the goal clause* $\bot \leftarrow q \wedge \circ(\neg a\, \mathcal{R}\, q) \in \Gamma^1_0$, *the goal clause* $\bot \leftarrow \circ(\neg a\, \mathcal{R}\, q)$ *is obtained as resolvent. Then, by* $(Sbm)$, *the goal clause* $\bot \leftarrow q \wedge \circ(\neg a\, \mathcal{R}\, q)$ *is subsumed by* $\bot \leftarrow \circ(\neg a\, \mathcal{R}\, q)$. *The second application of* $(Res)$, *this time with the program clause* $q \leftarrow \top$ *and the goal clause* $\Box(\bot \leftarrow q \wedge a)$ *as premises, yields the goal clause* $\bot \leftarrow a$. *Then the rule* $(Res)$ *is applied to the two program clauses* $q \leftarrow \top$ *and* $\Box(\circ q \leftarrow q)$ *and the program clause* $\circ q \leftarrow \top$ *is obtained as resolvent before jumping to the next state by applying the rule* $(Unx)$ *to the* IFT*-closed pair* $(\Pi^5_0, \Gamma^5_0)$.

**Remark 5.4.6.** *Note that* $(\Pi^5_0, \Gamma^5_0)$ *is obtained by nx-resolution from* $(\Pi^4_0, \Gamma^4_0)$. *Let us explain that this step is essential. In* $(\Pi^4_0, \Gamma^4_0)$ *goal-resolution is not applicable. If instead of applying nx-resolution to the clauses* $q \leftarrow \top$ *and* $\Box(\circ q \leftarrow q)$ *in* $\Pi^4_0$, *we applied the rule* $(Unx)$ *to the pair* $(\Pi^4_0, \Gamma^4_0)$, *then we would obtain the program* $\Pi' = \{\Box(\circ q \leftarrow q)\}$ *and the goal* $\Gamma' = \{\bot \leftarrow \neg a\, \mathcal{R}\, q, \Box(\bot \leftarrow a)\}$. *Since* $\Pi' \cup \Gamma'$ *is satisfiable, the refutation of* $\Pi \cup \Gamma$ *would never be found.*

*By applying the rule* $(Unx)$ *to* $(\Pi^5_0, \Gamma^5_0)$, *we obtain the pair* $(\Pi^0_1, \Gamma^0_1)$. *Then, we apply the context-dependent rule* $(\mathcal{R}\, C_-)$ *with respect to the selected eventuality literal* $\neg(\neg a\, \mathcal{R}\, q)$ *and the clause* $q \leftarrow \top$ *as context. The pair* $(\Pi^1_1, \Gamma^1_1)$ *is obtained by replacing the goal clause* $\bot \leftarrow \neg a\, \mathcal{R}\, q$ *in* $\Gamma^0_1$ *with the program clauses* $a \leftarrow q$ *and* $\Box(a \leftarrow b)$ *and the goal clauses* $\bot \leftarrow q \wedge \circ(\neg b\, \mathcal{R}\, q)$ *and* $\Box(\bot \leftarrow q \wedge b)$, *where* $b$ *is a fresh propositional variable. The value of* $\mathsf{sel\_ev\_set}^*_1$ *is* $\{\neg(\neg b\, \mathcal{R}\, q)\}$. *In* $(\Pi^1_1, \Gamma^1_1)$ *the resolution rule is applied with the program clause* $q \leftarrow \top$ *and the goal clause* $\Box(\bot \leftarrow q \wedge a)$ *as premises, obtaining the goal clause* $\bot \leftarrow a$ *as resolvent. In* $(\Pi^2_1, \Gamma^2_1)$ *the resolution between the program clause* $a \leftarrow q$ *and the goal clause* $\bot \leftarrow a$ *yields the goal clause* $\bot \leftarrow q$. *Finally, since* $\Pi^3_1$ *contains the program clause* $q \leftarrow \top$ *and* $\Gamma^3_1$ *contains the goal clause* $\bot \leftarrow q$, *the empty clause is obtained by applying the resolution rule* $(Res)$ *to these two clauses.*

Next we straightforwardly adapt Example 4.6.8 (Figures 4.17 and 4.18) to the syntax of TeDiLog. Let us recall that this example illustrates that our resolution mechanism does not need backtracking (independently of the selection strategy carried out by the operation fair_select).

**Example 5.4.7.** *We consider the program* $\Pi = \{\Diamond q \leftarrow \top, \Diamond r \leftarrow \top\}$ *and the goal* $\Gamma = \{\Box(\bot \leftarrow q \wedge \Diamond r)\}$. *The set* $\Pi \cup \Gamma$ *is satisfiable. There are two eventualities,* $\Diamond q$ *and* $\Diamond r$, *that must be fulfilled, but the goal clause states that once the eventuality* $\Diamond q$ *is fulfilled, the eventuality* $\Diamond r$ *cannot be fulfilled. An infinite* IFT*-derivation for* $\Pi \cup \Gamma$ *is shown in detail in Figures 5.16, 5.17 and 5.18 (it is split due to space reasons). Although the eventuality* $\Diamond q$ *is*

| | | | | | |
|---|---|---|---|---|---|
| $\Pi_0^0 =$ | $\{\underline{\Diamond q \leftarrow \top}, \Diamond r \leftarrow \top\}$ | $\Gamma_0^0 =$ | $\{\Box(\bot \leftarrow q \wedge \Diamond r)\}$ | $(\Diamond C_+)$ | $\mathsf{sel\_ev\_set}_0 = \{\Diamond q\}$ |
| $\Pi_0^1 =$ | $\{\underline{\Diamond r \leftarrow \top},$<br>$q \vee \circ(a \mathcal{U} q) \leftarrow \top\}$ | $\Gamma_0^1 =$ | $\{\Box(\bot \leftarrow q \wedge \Diamond r),$<br>$\Box(\bot \leftarrow a \wedge \Diamond r)\}$ | $(\Diamond H_+)$ | $\mathsf{sel\_ev\_set}_0^* = \{a \mathcal{U} q\}$ |
| $\Pi_0^2 =$ | $\{q \vee \circ(a \mathcal{U} q) \leftarrow \top,$<br>$r \vee \circ\Diamond r \leftarrow \top\}$ | $\Gamma_0^2 =$ | $\{\underline{\Box(\bot \leftarrow q \wedge \Diamond r)},$<br>$\Box(\bot \leftarrow a \wedge \Diamond r)\}$ | $(\Diamond B_+)$ | |
| $\Pi_0^3 =$ | $\{q \vee \circ(a \mathcal{U} q) \leftarrow \top,$<br>$r \vee \circ\Diamond r \leftarrow \top\}$ | $\Gamma_0^3 =$ | $\{\underline{\Box(\bot \leftarrow a \wedge \Diamond r)},$<br>$\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r)\}$ | $(\Diamond B_+)$ | |
| $\Pi_0^4 =$ | $\{q \vee \circ(a \mathcal{U} q) \leftarrow \top,$<br>$\underline{r \vee \circ\Diamond r \leftarrow \top}\}$ | $\Gamma_0^4 =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\underline{\Box(\bot \leftarrow q \wedge \circ\Diamond r)},$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r)\}$ | $(Res)$ | |
| $\Pi_0^5 =$ | $\{q \vee \circ(a \mathcal{U} q) \leftarrow \top,$<br>$r \vee \circ\Diamond r \leftarrow \top,$<br>$\underline{r \leftarrow q}\}$ | $\Gamma_0^5 =$ | $\{\underline{\Box(\bot \leftarrow q \wedge r)},$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r)\}$ | $(Res)$ | |
| $\Pi_0^6 =$ | $\{q \vee \circ(a \mathcal{U} q) \leftarrow \top,$<br>$\underline{r \vee \circ\Diamond r \leftarrow \top},$<br>$r \leftarrow q\}$ | $\Gamma_0^6 =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\underline{\Box(\bot \leftarrow a \wedge \circ\Diamond r)},$<br>$\bot \leftarrow q\}$ | $(Res)$ | |
| $\Pi_0^7 =$ | $\{q \vee \circ(a \mathcal{U} q) \leftarrow \top,$<br>$r \vee \circ\Diamond r \leftarrow \top,$<br>$r \leftarrow q,$<br>$\underline{r \leftarrow a}\}$ | $\Gamma_0^7 =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\underline{\Box(\bot \leftarrow a \wedge r)},$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\bot \leftarrow q\}$ | $(Res)$ | |
| $\Pi_0^8 =$ | $\{\underline{q \vee \circ(a \mathcal{U} q) \leftarrow \top},$<br>$r \vee \circ\Diamond r \leftarrow \top,$<br>$r \leftarrow q,$<br>$r \leftarrow a\}$ | $\Gamma_0^8 =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\underline{\bot \leftarrow q}, \bot \leftarrow a\}$ | $(Res)$ | |
| $\Pi_0^9 =$ | $\{q \vee \circ(a \mathcal{U} q) \leftarrow \top,$<br>$r \vee \circ\Diamond r \leftarrow \top,$<br>$\underline{r \leftarrow q},$<br>$r \leftarrow a,$<br>$\circ(a \mathcal{U} q) \leftarrow \top\}$ | $\Gamma_0^9 =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\underline{\bot \leftarrow q}, \bot \leftarrow a\}$ | $(Sbm)$ | |

*Figure 5.16:* IFT-derivation for $\Pi = \{\Diamond q \leftarrow \top, \Diamond r \leftarrow \top\}$ and $\Gamma = \{\Box(\bot \leftarrow q \wedge \Diamond r)\}$ (Part 1 of 3)

| | | | | | |
|---|---|---|---|---|---|
| $\Pi_0^{10} =$ | $\{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top,$<br>$r \vee \circ\Diamond r \leftarrow \top,$<br>$\underline{r \leftarrow a},$<br>$\circ(a\,\mathcal{U}\,q) \leftarrow \top\}$ | $\Gamma_0^{10} =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\bot \leftarrow q, \underline{\bot \leftarrow a}\}$ | $(Sbm)$ | |
| $\Pi_0^{11} =$ | $\{\underline{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top,}$<br>$r \vee \circ\Diamond r \leftarrow \top,$<br>$\underline{\circ(a\,\mathcal{U}\,q) \leftarrow \top}\}$ | $\Gamma_0^{11} =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\bot \leftarrow q, \bot \leftarrow a\}$ | $(Sbm)$ | |
| $\Pi_0^{12} =$ | $\{r \vee \circ\Diamond r \leftarrow \top,$<br>$\circ(a\,\mathcal{U}\,q) \leftarrow \top\}$ | $\Gamma_0^{12} =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\bot \leftarrow q, \bot \leftarrow a\}$ | $(Unx)$ | |
| $\Pi_1^{0} =$ | $\{\underline{a\,\mathcal{U}\,q \leftarrow \top}\}$ | $\Gamma_1^{0} =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r)\}$ | $(\mathcal{U}C_+)$ | $\mathsf{sel\_ev\_set}_1 = \{a\,\mathcal{U}\,q\}$ |
| $\Pi_1^{1} =$ | $\{\underline{q \vee a \leftarrow \top},$<br>$q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top\}$ | $\Gamma_1^{1} =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\underline{\Box(\bot \leftarrow a \wedge r)},$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow b)\}$ | $(Res)$ | $\mathsf{sel\_ev\_set}_1^* = \{b\,\mathcal{U}\,q\}$ |
| $\Pi_1^{2} =$ | $\{q \vee a \leftarrow \top,$<br>$q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top,$<br>$\underline{q \leftarrow r}\}$ | $\Gamma_1^{2} =$ | $\{\underline{\Box(\bot \leftarrow q \wedge r)},$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow b)\}$ | $(Res)$ | |
| $\Pi_1^{3} =$ | $\{\underline{q \vee a \leftarrow \top},$<br>$q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top,$<br>$q \leftarrow r\}$ | $\Gamma_1^{3} =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\underline{\Box(\bot \leftarrow q \wedge \circ\Diamond r)},$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\Box(\bot \leftarrow a \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow b), \bot \leftarrow r\}$ | $(Res)$ | |
| $\Pi_1^{4} =$ | $\{q \vee a \leftarrow \top,$<br>$q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top,$<br>$q \leftarrow r,$<br>$\underline{a \leftarrow \circ\Diamond r}\}$ | $\Gamma_1^{4} =$ | $\{\Box(\bot \leftarrow q \wedge r),$<br>$\Box(\bot \leftarrow q \wedge \circ\Diamond r),$<br>$\Box(\bot \leftarrow a \wedge r),$<br>$\underline{\Box(\bot \leftarrow a \wedge \circ\Diamond r)},$<br>$\Box(\bot \leftarrow b), \bot \leftarrow r\}$ | $(Res)$ | |

*Figure 5.17:* IFT-derivation for $\Pi = \{\Diamond q \leftarrow \top, \Diamond r \leftarrow \top\}$ and $\Gamma = \{\Box(\bot \leftarrow q \wedge \Diamond r)\}$ (Part 2 of 3)

$$\begin{array}{llllll}
\Pi_1^5 = & \{q \lor a \leftarrow \top, & \Gamma_1^5 = & \{\Box(\bot \leftarrow q \land r), & (Sbm) & \\
& q \lor \circ(b\,\mathcal{U}\,q) \leftarrow \top, & & \Box(\bot \leftarrow q \land \circ\Diamond r), & & \\
& \underline{q \leftarrow r}, & & \Box(\bot \leftarrow a \land r), & & \\
& a \leftarrow \circ\Diamond r\} & & \Box(\bot \leftarrow a \land \circ\Diamond r), & & \\
& & & \Box(\bot \leftarrow b), & & \\
& & & \underline{\bot \leftarrow r}, \bot \leftarrow \circ\Diamond r\} & & \\
\Pi_1^6 = & \{q \lor a \leftarrow \top, & \Gamma_1^6 = & \{\Box(\bot \leftarrow q \land r), & (Sbm) & \\
& q \lor \circ(b\,\mathcal{U}\,q) \leftarrow \top, & & \Box(\bot \leftarrow q \land \circ\Diamond r), & & \\
& \underline{a \leftarrow \circ\Diamond r}\} & & \Box(\bot \leftarrow a \land r), & & \\
& & & \Box(\bot \leftarrow a \land \circ\Diamond r), & & \\
& & & \Box(\bot \leftarrow b), & & \\
& & & \bot \leftarrow r, \underline{\bot \leftarrow \circ\Diamond r}\} & & \\
\Pi_1^7 = & \{q \lor a \leftarrow \top, & \Gamma_1^7 = & \{\Box(\bot \leftarrow q \land r), & (Unx) & \\
& q \lor \circ(b\,\mathcal{U}\,q) \leftarrow \top\} & & \Box(\bot \leftarrow q \land \circ\Diamond r), & & \\
& & & \Box(\bot \leftarrow a \land r), & & \\
& & & \Box(\bot \leftarrow a \land \circ\Diamond r), & & \\
& & & \Box(\bot \leftarrow b), & & \\
& & & \bot \leftarrow r, \bot \leftarrow \circ\Diamond r\} & & \\
\Pi_2^0 = & \emptyset & \Gamma_2^0 = & \{\Box(\bot \leftarrow q \land r), & (\Diamond B_+) & \mathsf{sel\_ev\_set}_2 = \emptyset \\
& & & \Box(\bot \leftarrow q \land \circ\Diamond r), & & \\
& & & \Box(\bot \leftarrow a \land r), & & \\
& & & \Box(\bot \leftarrow a \land \circ\Diamond r), & & \\
& & & \Box(\bot \leftarrow b), & & \\
& & & \underline{\bot \leftarrow \Diamond r}\} & & \\
\Pi_2^1 = & \emptyset & \Gamma_2^1 = & \{\Box(\bot \leftarrow q \land r), & (Unx) & \mathsf{sel\_ev\_set}_2^* = \emptyset \\
& & & \Box(\bot \leftarrow q \land \circ\Diamond r), & & \\
& & & \Box(\bot \leftarrow a \land r), & & \\
& & & \Box(\bot \leftarrow a \land \circ\Diamond r), & & \\
& & & \Box(\bot \leftarrow b), & & \\
& & & \bot \leftarrow r, \bot \leftarrow \circ\Diamond r\} & & \\
& \vdots & & \vdots & &
\end{array}$$

$\Pi_2^0 = \Pi_j^0, \Gamma_2^0 = \Gamma_j^0, \Pi_2^1 = \Pi_j^1, \Gamma_2^1 = \Gamma_j^1$ and
$\mathsf{sel\_ev\_set}_j = \mathsf{sel\_ev\_set}_j^* = \emptyset$ for every $j \geq 3$
$\{\neg q, r, \neg a\} \mapsto \{q, \neg r, \neg b\} \mapsto \{\neg r, \neg b\} \mapsto \{\neg r, \neg b\} \cdots$

*Figure 5.18:* IFT-derivation for $\Pi = \{\Diamond q \leftarrow \top, \Diamond r \leftarrow \top\}$ and $\Gamma = \{\Box(\bot \leftarrow q \land \Diamond r)\}$ (Part 3 of 3)

*selected first by the operation* fair_select*, the eventuality $\diamond r$ is fulfilled before $\diamond q$. Note that backtracking is not used.*

*After the first selection,* $\mathsf{sel\_ev\_set}_0 = \{\diamond q\}$*. Then the application of the rule $(\diamond C_+)$ with context $\{\diamond r \leftarrow \top\}$ generates the program clause $q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top$ and the goal clause $\Box(\bot \leftarrow a \wedge \diamond r)$ where $a$ is a fresh propositional variable. Additionaly the value of* $\mathsf{sel\_ev\_set}_0^*$ *is set to $\{a\,\mathcal{U}\,q\}$. Then, the rule applications that correspond to the* supported_free_close *operation (see Figure 5.13, line 8 and Definition 5.4.3) are carried out and the* IFT*-closed pair $(\Pi_0^{12}, \Gamma_0^{12})$ is obtained. Next, by rule $(Unx)$, the pair $(\Pi_1^0, \Gamma_1^0)$ is generated. Since the atom $a\,\mathcal{U}\,q$ belongs to* $\mathsf{EventLits}(\Pi_1^0 \cup \Gamma_1^0)$*, it remains as the selected literal and, consequently, the rule $(\mathcal{U}\,C_+)$ is applied to $(\Pi_1^0 \cup \Gamma_1^0)$ with $a\,\mathcal{U}\,q$ as selected literal (i.e.,* $\mathsf{sel\_ev\_set}_1 = \{a\,\mathcal{U}\,q\}$*) and with empty context, obtaining the pair $(\Pi_1^1 \cup \Gamma_1^1)$ and setting* $\mathsf{sel\_ev\_set}_1^*$ *to $\{b\,\mathcal{U}\,q\}$, where $b$ is a fresh propositional variable. The operation* supported_free_close *that yields the* IFT*-closed pair $(\Pi_1^7 \cup \Gamma_1^7)$ from $(\Pi_1^1 \cup \Gamma_1^1)$, encapsulates several applications of the rule $(Res)$ and the rule $(Sbm)$. The pair $(\Pi_2^0, \Gamma_2^0)$ is obtained from $(\Pi_1^7 \cup \Gamma_1^7)$ by using the rule $(Unx)$. Since the set* $\mathsf{EventLits}(\Pi_2^0 \cup \Gamma_2^0)$ *is empty, the value of* $\mathsf{sel\_ev\_set}_2$ *as well as the value of* $\mathsf{sel\_ev\_set}_2^*$ *is the empty set. Therefore no context-dependent rule is applied to $(\Pi_2^0, \Gamma_2^0)$ and we get the* IFT*-closed pair $(\Pi_2^1, \Gamma_2^1)$ by applying the context-free rule $(\diamond B_+)$. From that point onwards the derivation is a repetition where $\Pi_j^0 = \Pi_2^0$, $\Gamma_j^0 = \Gamma_2^0$, $\Pi_2^1 = \Pi_j^1$, $\Gamma_2^1 = \Gamma_j^1$ and* $\mathsf{sel\_ev\_set}_j = \mathsf{sel\_ev\_set}_j^* = \emptyset$ *for every $j \geq 3$.*

*The pairs $(\Pi_0^{12}, \Gamma_0^{12}), (\Pi_1^7, \Gamma_1^7), (\Pi_2^1, \Gamma_2^1), (\Pi_3^1, \Gamma_3^1), \ldots$ characterize a collection of models for the initial pair $(\Pi, \Gamma)$. All the models of such collection make true the literals $\{\neg q, r, \neg a\}$ in $s_0$, the literals $\{\neg r, \neg b\}$ in $s_1$ and also the literals $\{\neg r, \neg b\}$ in all the states $s_j$ such that $j \geq 2$. Moreover, the atom $q$ must be true in $s_k$ for some $k \geq 1$. For instance, the* PLTL*-structure $\mathcal{M}$ with states $s_0, s_1, s_2, \ldots$ such that $V_{\mathcal{M}}(s_0) = \{r\}$, $V_{\mathcal{M}}(s_1) = \{q\}$ and $V_{\mathcal{M}}(s_j) = \emptyset$ for every $j \geq 2$ is a model of $\Pi \cup \Gamma$.*

By means of Example 5.4.7 we would like to stress that the strategy for selecting eventualities does not compromise the completeness of our resolution mechanism, although it can affect efficiency. As already pointed out after Example 4.6.8 , if we had selected the eventuality $\diamond r$ instead of the eventuality $\diamond q$, the derivation would have been considerably longer.

### 5.4.3 Logical Semantics

In this subsection we define the logical characterization of the declarative meaning of TeDiLog programs.

The logical characterization of the declarative semantics of a TeDiLog program $\Pi$ is given, as usual in Logic Programming, by the set of all the formulas that represent (in a particular simplified way) negations of goals and that are logical consequences of the program $\Pi$.

In classical LP (see e.g. [88]), and also in some extensions like Templog ([12]) and Chronolog ([127, 103, 99]), where a goal is of the form $\bot \leftarrow B$, the declarative meaning of a program is formalized in three equivalent ways:

1. Logically, as the set of bodies that are logical consequences of the program.

2. Model-theoretically, by means of the minimal model of the program.

3. By fixpoint characterization, based on the immediate consequence operator.

These three formalizations are equivalent in the sense that, on one hand, the bodies that are logical consequences of the program are just the bodies that are satisfied by the minimal model of the program and, on the other hand, the minimal model of the program is the fixpoint of the immediate consequence operator.

In DLP ([89]), and existing temporal extensions of DLP ([68]), where a goal is of the form $\{\bot \leftarrow B_1, \dots, \bot \leftarrow B_n\}$, the logical characterization of the declarative meaning of a program is provided by the set of formulas of the form $B_1 \vee \dots \vee B_n$ (i.e. disjunctions of bodies) that are logical consequences of the program. The model-theoretic characterization is provided by means of all the minimal models (in general there is no only one minimal model). The fixpoint characterization can also be extended to the disjunctive paradigm as shown in [89, 68].

In TeDiLog a goal $\Gamma = \{\Box^{b_1}(\bot \leftarrow B_1), \dots, \Box^{b_n}(\bot \leftarrow B_n)\}$ is understood as the conjunction of the goal clauses in $\Gamma$. Since a goal clause $\Box^{b}(\bot \leftarrow B)$ represents the formula $\neg\Diamond^{b}B$, the set $\Gamma$ is logically equivalent to the formula $\neg\Diamond^{b_1}B_1 \wedge \dots \wedge \neg\Diamond^{b_n}B_n$ or equivalently to $\neg(\Diamond^{b_1}B_1 \vee \dots \vee \Diamond^{b_n}B_n)$.

**Definition 5.4.8.** *The declarative semantics of a* TeDiLog *program* $\Pi$ *is logically characterized as the set of all the formulas of the form* $\Diamond^{b_1}B_1 \vee \dots \vee \Diamond^{b_n}B_n$ *that are logical consequences of* $\Pi$.

We do not provide model-theoretical and fixpoint characterizations for TeDiLog due to technical difficulties that we explain in the next subsection.

### *5.4.4 Equivalence between operational and logical semantics*

In this subsection we address the soundness and completeness of IFT-resolution with respect to the logical semantics of TeDiLog.

Soundness and completeness results guarantee the equivalence between operational and logical semantics.

Soundness is a consequence of the fact that each rule preserves satisfiability (indeed, some of them preserve logical equivalence).

**Theorem 5.4.9. (Soundness)** *If there exists an* IFT*-refutation from* $\Pi$ *with top-goal* $\Gamma$*, then* $\Pi \cup \Gamma$ *is unsatisfiable.*

*Proof.* If $\Box^{b}(\bot \leftarrow \top) \in \Gamma'$ for some $(\Pi', \Gamma')$ in an IFT-derivation from $(\Pi, \Gamma)$, then $\Pi' \cup \Gamma'$ is unsatisfiable. Therefore, since the rule $(Unx)$ preserves satisfiability and the initial set and the target set of every application of the remaining rules are equisatisfiable, $\Pi \cup \Gamma$ is also unsatisfiable. ∎

For more details about the proof of the above theorem see Section 4.5.

In logic programming, completeness proofs are usually addressed through the minimal model and the immediate consequence operator. In the case of TeDiLog there are many difficulties for using classical notions of minimal model and immediate consequence operator $T_\Pi$ in a customary completeness proof. The main reason is related to the contexts that are essential for IFT-resolution. Concretely, context handling prevents to deduce the refutability of $T_\Pi^n(\emptyset)$ from the refutability of $T_\Pi^{n+1}(\emptyset)$ (see e.g. Lemma 4.6 in [12]). As pointed out in Section 5.1, these difficulties are closely related to the problem of syntactical cut elimination in PLTL. We have explored non-conventional notions of minimal model and immediate consequence operator, which are not only based on programs, but also need to consider all the possible goals. Unfortunately, these intricate notions of minimal model and immediate consequence operator do not facilitate the understanding of the declarative meaning of TeDiLog programs. Hence, we decided not to include them in this thesis.

TeDiLog's completeness means that whenever a set of clauses $\Pi \cup \Gamma$ is unsatisfiable, the IFT-resolution algorithm gives a refutation for $(\Pi, \Gamma)$. Since the algorithm for IFT-resolution is a straightforward adaptation of the systematic algorithm $\mathcal{SR}$ in Subsection 4.6.1, the idea behind the completeness proof and the involved technical details are very similar with respect to the ones presented to proof the completeness of the TRS resolution method. The main differences arise from the fact that, for satisfiable sets of clauses, the algorithm $\mathcal{SR}$ produces cyclic derivations whereas the algorithm for IFT-resolution produces infinite derivations. In this subsection, we adapt notions and results introduced in Section 4.4, Subsection 4.6.3 and Section 4.7 to TeDiLog. To prove completeness, we build a model $\mathcal{M}$ of any satisfiable set of clauses $\Pi \cup \Gamma$ on the basis of the infinite IFT-derivation $\mathcal{D}(\Pi, \Gamma)$ obtained by the IFT-resolution algorithm. The main difficulty in the construction of the model $\mathcal{M}$ are the eventuality literals. In particular, we must ensure that the fulfillment of eventualities is not infinitely delayed in the PLTL-structures obtained from $\mathcal{D}(\Pi, \Gamma)$ and that are intended to give rise to models of $\Pi \cup \Gamma$. [4] With such a purpose, we first show that the sequence of the so-called descendants of a selected eventuality is finite.

**Definition 5.4.10.** *We say that an eventuality literal $N'$ is a* direct descendant *of other eventuality literal $N$ with respect to an* IFT*-derivation $\mathcal{D}$, if* $\mathsf{sel\_ev\_set}_i = \{N\}$ *and* $\mathsf{sel\_ev\_set}_i^* = \{N'\}$ *in $\mathcal{D}$. The* sequence of descendants *of $N$ with respect to $\mathcal{D}$, is the longest sequence $N_0, N_1, \ldots$ such that $N_0 = N$ and for all $j \geq 0$: $N_{j+1}$ is a direct descendant of $N_j$ with respect to $\mathcal{D}$. Any literal $N_j$ in that sequence is called a descendant of $N$ if $j \geq 1$.*

An infinite sequence of descendants for the selected eventuality requires the existence of an infinite number of different contexts, since the repetition of a context yields a refutation. An infinite number of different contexts is only possible if the IFT-resolution procedure introduces fresh propositional variables in the context. A priori, there could be two ways for generating new propositional variables in the IFT-derivation. The first is the translation to clausal form applied in the output to the context-dependent rules (function def). However, no new variables are introduced in this way because classical distribution laws are enough to obtain the clausal form (more details in Subsection 4.6.3 and Subsection 4.2.2). The second potential source of new propositional variables is the explicit occurrence of a fresh variable in the consequent of each context-dependent rule. However, we can ensure that the new variables explicitly introduced by the context-dependent rules are never part of the context. Indeed, it is a consequence of the following three facts:

1. The clauses defining a new variable are always-clauses, which are excluded from the negated context.
2. The context-dependent rules are always applied just after the application of the rule $(Unx)$ to sets where the propositional variables introduced (as fresh) by previous applications of

[4] Under the assumption that the strategy for selecting eventualities is fair in the sense that every eventuality that from some moment onwards is available for being selected whenever an eventuality must be selected, is selected at some time.

context-dependent rules are also out of the context.

3. The marking strategy prioritizes the selection of the descendants of an eventuality literal that has previously been selected.

Consequently, the number of possible different contexts is finite and the construction of $\mathcal{M}$ is based on the following auxiliary lemma that ensures that clauses containing eventuality literals can be (finitely) satisfied in $\mathcal{M}$.

**Lemma 5.4.11.** *Let $\mathcal{D}(\Pi,\Gamma)$ be a derivation and let $N$ be an eventuality literal such that $N \in$ $\mathsf{sel\_ev\_set}_i$ for some $i \geq 0$. The sequence of descendants of $N$ with respect to $\mathcal{D}(\Pi,\Gamma)$ is finite.*

*Proof.* This is a particular case of Lemma 4.6.13 in Chapter 4. ∎

Next, we construct a model $\mathcal{M}$ of $(\Pi,\Gamma)$ from the infinite IFT-derivation $\mathcal{D}(\Pi,\Gamma)$. First, we introduce some auxiliary notions and results. In particular we need to extend (with literals) the pairs in the derivation $\mathcal{D}(\Pi,\Gamma)$ in a coherent way that allows us to get models.

**Definition 5.4.12.** *A local derivation is called a* local refutation *if it is a refutation. Given a program $\Pi$ and a goal $\Gamma$, the pair $(\Pi,\Gamma)$ is* locally inconsistent *iff there exists a local refutation for $(\Pi,\Gamma)$. Otherwise it is* locally consistent.

**Definition 5.4.13.** *Let $\Pi$ be a program and $\Gamma$ a goal. A* literal-based extension of $(\Pi,\Gamma)$ is any pair $(\widehat{\Pi},\widehat{\Gamma})$ of sets that satisfies the following conditions:

*(a)* $\Pi \subseteq \widehat{\Pi} \subseteq (\Pi \cup \mathsf{Lits}(\Pi))$ and $\Gamma \subseteq \widehat{\Gamma} \subseteq (\Gamma \cup \mathsf{Lits}(\Pi \cup \Gamma))$

*(b)* For every literal $N \in \widehat{\Pi} \cup \widehat{\Gamma}$, if $N$ is of the form $A$ then $A \notin \widehat{\Gamma}$ and if $N$ is of the form $\neg A$ then $\neg A \notin \widehat{\Pi}$.

Given a literal-based extension $(\widehat{\Pi},\widehat{\Gamma})$ of $(\Pi,\Gamma)$, we denote as $\mathsf{PG}(\widehat{\Pi},\widehat{\Gamma})$ the pair formed by the program $\Pi \cup \{A \leftarrow \top \mid A \in \widehat{\Pi} \cap \mathsf{Lits}(\Pi)\}$ and the goal $\Gamma \cup \{\bot \leftarrow A \mid \neg A \in \widehat{\Gamma} \cap \mathsf{Lits}(\Pi \cup \Gamma)\}$.

**Definition 5.4.14.** *Let $\Pi$ be a program, $\Gamma$ a goal and $(\widehat{\Pi},\widehat{\Gamma})$ a literal-based extension of $(\Pi,\Gamma)$. The pair $(\widehat{\Pi},\widehat{\Gamma})$ is* literal-closed *iff $(\widehat{\Pi} \cup \widehat{\Gamma}) \cap \mathsf{Lits}(C) \neq \emptyset$ for every $C \in \Pi \cup \Gamma$. Besides, $\mathsf{lclc}(\Pi,\Gamma)$ denotes the collection of all the literal-based extensions $(\widehat{\Pi},\widehat{\Gamma})$ of $(\Pi,\Gamma)$ such that $(\widehat{\Pi},\widehat{\Gamma})$ is literal-closed and $\mathsf{PG}(\widehat{\Pi},\widehat{\Gamma})$ is locally consistent. We say that each $(\widehat{\Pi},\widehat{\Gamma}) \in \mathsf{lclc}(\Pi,\Gamma)$ is an* lclc-extension *of $(\Pi,\Gamma)$.*

**Proposition 5.4.15.** *If $(\Pi,\Gamma)$ is a locally consistent pair such that the set of atoms of the clauses in $\Pi \cup \Gamma$ is exclusively formed by atoms in $\mathsf{Prop}$ and atoms of the form $\circ A$ then, $\mathsf{lclc}(\Pi,\Gamma) \neq \emptyset$.*

*Proof.* Straightforward adaptation of Proposition 4.7.2. ∎

Next we introduce the notion of standard lclc-extensions of a pair formed by a program and a goal.

**Definition 5.4.16.** *Let $(\Pi,\Gamma)$ be a locally consistent IFT-closed pair where $\Pi$ is a program and $\Gamma$ a goal. We say that $(\widehat{\Pi},\widehat{\Gamma}) \in \mathsf{lclc}(\Pi,\Gamma)$ is* standard *iff it satisfies the following conditions:*

*(a) If $\circ A \in \widehat{\Pi}$, then there exists a clause $\Box^b(\circ A \vee \circ H \leftarrow \circ B) \in \Pi$*

*(b) If $\neg\circ A \in \widehat{\Gamma}$, then there exists a clause $\Box^b(\circ H \leftarrow \circ A \wedge \circ B) \in (\Pi \cup \Gamma)$*

*(c) If $A \in \widehat{\Pi}$, then $(\widehat{\Pi} \setminus \{A\}, \widehat{\Gamma}) \notin \mathsf{lclc}(\Pi, \Gamma)$.*

*(d) If $\neg A \in \widehat{\Gamma}$, then $(\widehat{\Pi}, \widehat{\Gamma} \setminus \{\neg A\}) \notin \mathsf{lclc}(\Pi, \Gamma)$.*

The following lemma ensures the existence of at least one standard lclc-extension of any locally consistent IFT-closed pair $(\Pi, \Gamma)$.

**Lemma 5.4.17.** *Let $(\Pi, \Gamma)$ be a locally consistent* IFT*-closed pair. There exists at least one standard pair in $\mathsf{lclc}(\Pi, \Gamma)$.*

*Proof.* Straightforward adaptation of Lemma 4.7.6. ■

We build standard lclc-extensions of each IFT-closed pair $(\Pi^*_i, \Gamma^*_i)$ in $\mathcal{D}(\Pi, \Gamma)$. Note that each $(\Pi^*_i, \Gamma^*_i)$ is the last pair of the sequence $S_i$ (see Section 5.4.1) for every $i \geq 0$. We denote by $(\widehat{\Pi^*_i}, \widehat{\Gamma^*_i})$ any lclc-extension of $(\Pi^*_i, \Gamma^*_i)$. The infinite sequences of $(\widehat{\Pi^*_i}, \widehat{\Gamma^*_i})$ will represent models of $\Pi \cup \Gamma$. Such infinite sequences must be coherent with respect to the meaning of temporal connectives. To this end, a successor relation is defined for the lclc-extensions of the IFT-closed pairs $(\Pi^*_i, \Gamma^*_i)$. This successor relation on

$$\{\mathsf{lclc}(\Pi^*_{i-1}, \Gamma^*_{i-1}) \times \mathsf{lclc}(\Pi^*_i, \Gamma^*_i) \mid i \geq 1\}$$

is presented in Definition 5.4.18.

**Definition 5.4.18.** *Let $i \geq 1$. We say that a pair $(\widehat{\Pi^*_i}, \widehat{\Gamma^*_i})$ is a* successor *of $(\widehat{\Pi^*_{i-1}}, \widehat{\Gamma^*_{i-1}})$ if for every $\circ\lambda \in (\widehat{\Pi^*_{i-1}} \cup \widehat{\Gamma^*_{i-1}}) \cap \mathsf{Lits}(\Pi^*_{i-1} \cup \Gamma^*_{i-1})$ there is some $S \in \mathsf{nxclo}_i(\circ\lambda)$ such that $S \subseteq \widehat{\Pi^*_i} \cup \widehat{\Gamma^*_i}$, where $\mathsf{nxclo}_i$ is defined as follows*

1. $\mathsf{nxclo}_i(\circ p) = \{\{p\}\}$ *and* $\mathsf{nxclo}_i(\neg\circ p) = \{\{\neg p\}\}$
2. $\mathsf{nxclo}_i(\circ\circ A) = \{\{\circ A\}\}$ *and* $\mathsf{nxclo}_i(\neg\circ\circ A) = \{\{\neg\circ A\}\}$
3. $\mathsf{nxclo}_i(\circ(p_1 \mathcal{U} p_2)) = \{\{p_2\}, \{p_1, \circ(p_1 \mathcal{U} p_2)\}\}$ *if* $p_1 \mathcal{U} p_2 \notin \mathsf{sel\_ev\_set}_i$
4. $\mathsf{nxclo}_i(\circ(p_1 \mathcal{U} p_2)) = \{\{p_2\}, \{p_1, \circ(a \mathcal{U} p_2)\}\}$ *if* $p_1 \mathcal{U} p_2 \in \mathsf{sel\_ev\_set}_i$ *and* $a \mathcal{U} P_2 \in \mathsf{sel\_ev\_set}^*_i$
5. $\mathsf{nxclo}_i(\neg\circ(p_1 \mathcal{U} p_2)) = \{\{\neg p_2, \neg p_1\}, \{\neg p_2, \neg\circ(p_1 \mathcal{U} p_2)\}\}$.

*The definition of $\mathsf{nxclo}_i$ for each of the remaining cases –i.e. $\circ(\neg p_1 \mathcal{U} p_2)$, $\neg\circ(\neg p_1 \mathcal{U} p_2)$, $\circ(L \mathcal{R} p)$, $\neg\circ(L \mathcal{R} p)$, $\circ\Diamond p$, $\neg\circ\Diamond p$, $\circ\Box p$ and $\neg\circ\Box p$– follows straightforwardly from the corresponding equivalence.*
*If $(\widehat{\Pi^*_i}, \widehat{\Gamma^*_i})$ is a* successor *of $(\widehat{\Pi^*_{i-1}}, \widehat{\Gamma^*_{i-1}})$, we also say that $(\widehat{\Pi^*_{i-1}}, \widehat{\Gamma^*_{i-1}})$ is a* predecessor *of $(\widehat{\Pi^*_i}, \widehat{\Gamma^*_i})$.*
*The set of successors of a given pair $(\widehat{\Pi^*_j}, \widehat{\Gamma^*_j})$ is denoted by $\mathsf{succ}(\widehat{\Pi^*_j}, \widehat{\Gamma^*_j})$.*

The existence of infinite paths of standard lclc-extensions is based on the existence of a predecessor for each standard lclc-extension of an IFT-closed pair in the derivation which is a standard lclc-extension of the previous IFT-closed set in the derivation.

**Proposition 5.4.19.** *For every $i \geq 1$ and every standard pair $(\widehat{\Pi^*_i}, \widehat{\Gamma^*_i}) \in \mathsf{lclc}(\Pi^*_i, \Gamma^*_i)$, there exists a standard pair $(\widehat{\Pi^*_{i-1}}, \widehat{\Gamma^*_{i-1}}) \in \mathsf{lclc}(\Pi^*_{i-1}, \Gamma^*_{i-1})$ such that $(\widehat{\Pi^*_i}, \widehat{\Gamma^*_i}) \in \mathsf{succ}(\widehat{\Pi^*_{i-1}}, \widehat{\Gamma^*_{i-1}})$.*

*Proof.* Straightforward adaptation of Proposition 4.7.9. ■

**Proposition 5.4.20.** *For every $h \geq 0$ and every standard pair $(\widehat{\Pi_h^*}, \widehat{\Gamma_h^*}) \in \mathsf{lclc}(\Pi_h^*, \Gamma_h^*)$, there exists a sequence $(\widehat{\Pi_0^*}, \widehat{\Gamma_0^*}), (\widehat{\Pi_1^*}, \widehat{\Gamma_1^*}), \dots, (\widehat{\Pi_h^*}, \widehat{\Gamma_h^*})$ such that*

*(i) $(\widehat{\Pi_j^*}, \widehat{\Gamma_j^*}) \in \mathsf{lclc}(\Pi_j^*, \Gamma_j^*)$ and $(\widehat{\Pi_j^*}, \widehat{\Gamma_j^*})$ is standard for all $j \in \{0, \dots, h\}$ and*

*(ii) $(\widehat{\Pi_k^*}, \widehat{\Gamma_k^*}) \in \mathsf{succ}(\widehat{\Pi_{k-1}^*}, \widehat{\Gamma_{k-1}^*})$ for all $k \in \{1, \dots, h\}$.*

*Proof.* By induction on $h$. For $h = 0$ it holds trivially. For $h \geq 1$, by Proposition 5.4.19, there exists a standard $(\widehat{\Pi_{h-1}^*}, \widehat{\Gamma_{h-1}^*}) \in \mathsf{lclc}(\Pi_{h-1}^*, \Gamma_{h-1}^*)$ such that $(\widehat{\Pi_h^*}, \widehat{\Gamma_h^*}) \in \mathsf{succ}(\widehat{\Pi_{h-1}^*}, \widehat{\Gamma_{h-1}^*})$, therefore, by induction hypothesis on $(\widehat{\Pi_{h-1}^*}, \widehat{\Gamma_{h-1}^*})$, we can ensure the existence of the sequence $(\widehat{\Pi_0^*}, \widehat{\Gamma_0^*}), (\widehat{\Pi_1^*}, \widehat{\Gamma_1^*}), \dots, (\widehat{\Pi_h^*}, \widehat{\Gamma_h^*})$. ■

**Definition 5.4.21.** *We associate to $\mathcal{D}(\Pi, \Gamma)$ the set $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ that is formed by all the infinite sequences of the form $(\widehat{\Pi_0^*}, \widehat{\Gamma_0^*}), (\widehat{\Pi_1^*}, \widehat{\Gamma_1^*}), \dots$ such that $(\widehat{\Pi_j^*}, \widehat{\Gamma_j^*}) \in \mathsf{lclc}(\Pi_j^*, \Gamma_j^*)$ is standard for all $j \geq 0$ and $(\widehat{\Pi_k^*}, \widehat{\Gamma_k^*}) \in \mathsf{succ}(\widehat{\Pi_{k-1}^*}, \widehat{\Gamma_{k-1}^*})$ for every $k \geq 1$.*

**Proposition 5.4.22.** *If $\mathcal{D}(\Pi, \Gamma)$ is an infinite* IFT*-derivation, then the set $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ is non-empty.*

*Proof.* A direct consequence of Proposition 5.4.20. ■

**Definition 5.4.23.** *A sequence $\sigma \in \mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ is fulfilling for some $(\widehat{\Pi_j^*}, \widehat{\Gamma_j^*})$ in $\sigma$ and some literal $\circ(p_1 \mathcal{U} p_2) \in (\widehat{\Pi_j^*}, \widehat{\Gamma_j^*})$ iff there exists $k > j$ such that $p_2 \in (\widehat{\Pi_k^*}, \widehat{\Gamma_k^*})$ and $p_1 \in (\widehat{\Pi_h^*}, \widehat{\Gamma_h^*})$ for all $h \in \{j+1, \dots, k-1\}$.*
*The fulfilling notion is extended to literals $\neg p_1 \mathcal{U} p_2$, $\neg(L \mathcal{R} p)$, $\diamond p$ and $\neg\Box p$ in the obvious manner. A sequence $\sigma$ is fulfilling iff it is fulfilling for every eventuality literal that occurs in any of its pairs.*

The next three propositions are auxiliary results about the fulfillment of eventualities, which are useful for proving Lemma 5.4.27. In the three propositions only the case of eventuality literals of the form $p \mathcal{U} q$ has been considered. The proofs for the remaining cases –i.e. $(\neg p) \mathcal{U} q$, $\neg(p \mathcal{R} q)$, $\neg((\neg p) \mathcal{R} q)$, $\diamond q$ and $\neg\Box q$– are very similar.

**Proposition 5.4.24.** *Let $\sigma$ be a sequence in $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ and $(\widehat{\Pi_j^*}, \widehat{\Gamma_j^*})$ a pair in $\sigma$ such that $j \geq 0$ and $\circ(p_j \mathcal{U} p) \in \widehat{\Pi_j^*}$ and $p_j \mathcal{U} p \in \mathsf{sel\_ev\_set}_{j+1}$, then $p \in \widehat{\Pi_k^*}$ for some $k > j$.*

*Proof.* Let us suppose that $p \notin \widehat{\Pi_i^*}$ for every $i > j$. Since $\sigma$ is infinite and $(\widehat{\Pi_h^*}, \widehat{\Gamma_h^*})$ is a successor of $(\widehat{\Pi_{h-1}^*}, \widehat{\Gamma_{h-1}^*})$ for every $h \geq 1$, there exists, by Definition 5.4.18, an infinite sequence of descendants for $p_j \mathcal{U} p$ contradicting the result obtained in Lemma 5.4.11. ■

**Proposition 5.4.25.** *Let $\sigma$ be a sequence in $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ and $(\widehat{\Pi_j^*}, \widehat{\Gamma_j^*})$ a pair in $\sigma$ such that $j \geq 0$, $\circ(p_j \mathcal{U} p) \in \widehat{\Pi_j^*}$ and $p_j \mathcal{U} p \in \mathsf{sel\_ev\_set}_{j+1}$. If $h \geq j+1$ and $p \notin \widehat{\Pi_k^*}$ for every $k \in \{j+1, \dots, h\}$, then $p_j \in \widehat{\Pi_k^*}$ for every $k \in \{j+1, \dots, h\}$.*

*Proof.* A straightforward adaptation of Proposition 4.7.18. ■

**Proposition 5.4.26.** *Let $\sigma$ be a sequence in $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$. If $\{p_0 \,\mathcal{U}\, p, \circ(p_0 \,\mathcal{U}\, p)\} \cap \mathsf{Lits}(\Pi \cup \Gamma) \neq \emptyset$ and $\circ(p_0 \,\mathcal{U}\, p) \in (\widehat{\Pi^*_j}, \widehat{\Gamma^*_j})$ for some $j \geq 0$ and $\sigma$ is not fulfilling for $(\widehat{\Pi^*_j}, \widehat{\Gamma^*_j})$ and $\circ(p_0 \,\mathcal{U}\, p)$, then $p_0 \,\mathcal{U}\, p \notin \mathsf{sel\_ev\_set}_k$ and $\{p_0, \circ(p_0 \,\mathcal{U}\, p)\} \subseteq \widehat{\Pi^*_k}$ for every $k \geq j+1$.*

*Proof.* Since $\sigma$ belongs to $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$, by Definitions 5.4.18, 5.4.21 and 5.4.23, we can ensure that $p_0 \in \widehat{\Pi^*_k}$ and $p \notin \widehat{\Pi^*_k}$ for every $k \geq j+1$. Therefore, by using Propositions 5.4.24 and 5.4.25, we can ensure that $p_0 \,\mathcal{U}\, p \notin \mathsf{sel\_ev\_set}_k$ for every $k \geq j+1$, since otherwise $\sigma$ would be fulfilling for $(\widehat{\Pi^*_j}, \widehat{\Gamma^*_j})$ and $\circ(p_0 \,\mathcal{U}\, p)$. Consequently, by Definitions 5.4.18 and 5.4.21, we can also ensure that $\{p_0, \circ(p_0 \,\mathcal{U}\, p)\} \subseteq \widehat{\Pi^*_k}$ for every $k \geq j+1$. ■

Next we prove that every $\sigma \in \mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ is fulfilling. As a consequence, we know that there exists at least one fulfilling sequence in $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$.

**Lemma 5.4.27.** *For any infinite derivation $\mathcal{D}(\Pi,\Gamma)$, the set $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ contains at least one fulfilling sequence $\sigma$.*

*Proof.* By Proposition 5.4.22 the set $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ is non-empty. We show, by contradiction, that every sequence in $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ is fulfilling. For that, let us suppose that there is a sequence $\sigma$ in $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$ that is non-fulfilling, i. e., $\sigma$ does not fulfill a literal $\circ(p_0 \,\mathcal{U}\, p) \in \widehat{\Pi^*_j}$ for some $j \geq 0$. Then, by Proposition 5.4.26, $p_0 \,\mathcal{U}\, p \notin \mathsf{sel\_ev\_set}_k$ for every $k \geq j+1$ and $\{p_0, \circ(p_0 \,\mathcal{U}\, p)\} \subseteq \widehat{\Pi^*_k}$ for every $k \geq j+1$. This contradicts the fairness of the selection operation. ■

**Theorem 5.4.28. (Completeness)** *For any program $\Pi$ and any goal $\Gamma$, if $\Pi \cup \Gamma$ is unsatisfiable then there exists an* IFT*-refutation for $(\Pi, \Gamma)$.*

*Proof.* If there is no IFT-refutation for $(\Pi,\Gamma)$, the algorithm in Figure 5.13 produces an infinite derivation $\mathcal{D}(\Pi,\Gamma)$. By Lemma 5.4.27 there exists an infinite fulfilling sequence $\sigma$ in $\mathcal{G}_{\mathcal{D}(\Pi,\Gamma)}$. Now we define the PLTL-structure $\mathcal{M}_\sigma = (\sigma, V_{\mathcal{M}_\sigma})$ where the states are the pairs $(\widehat{\Pi^*_k}, \widehat{\Gamma^*_k})$ that form $\sigma$, and $V_{\mathcal{M}_\sigma}((\widehat{\Pi^*_k}, \widehat{\Gamma^*_k})) = \{p \in \mathsf{Prop} \mid p \in \widehat{\Pi^*_k}\}$ for every $k \geq 0$. It is routine to see that $\langle \mathcal{M}_\sigma, (\widehat{\Pi^*_k}, \widehat{\Gamma^*_k})\rangle \models C$ for all $C \in (\Pi^*_k, \Gamma^*_k)$ and all $k \geq 0$. Since any lclc-extension contains at least one element $\lambda$ that belongs to $\mathsf{Lits}(C)$, this is made by structural induction on the form of $\lambda$ and using Definition 5.4.18 and the fact that $\sigma$ is fulfilling (by Lemma 5.4.27). In particular $\mathcal{M}_\sigma$ is a model of $\Pi^*_0 \cup \Gamma^*_0$ and we can ensure that $\Pi_0 \cup \Gamma_0$ is satisfiable because all the rules other than $(Unx)$ preserve equisatisfiability. Hence, since $\Pi_0 \cup \Gamma_0 = \Pi \cup \Gamma$, the set of clauses $\Pi \cup \Gamma$ is satisfiable. ■

## 5.5 Related work

In Section 5.1, we have already surveyed the main features of the works that are more close to our proposal. In this section we add more details.

### 5.5.1 Templog: Abadi & Manna [2] and Baudinet [12]

The only temporal connectives allowed in the TLP language Templog introduced in [2, 12] are $\Box$, $\Diamond$ and $\circ$. An atom is of the form $\circ^i A$ where $A$ is a classical atom. A body $B$ is recursively defined as a conjunction $B_1 \wedge \ldots \wedge B_n$ with $n \geq 0$ and where each $B_i$ is a classical atom $A$, a formula of the form $\circ B'$, i.e., a body preceded by the connective $\circ$, or a formula of the form $\Diamond B'$, i.e., a body preceded bay the connective $\Diamond$. Program clauses are of the form $\Box^b((\Box^{b'} \circ^i A) \leftarrow B)$, with $b, b' \in \{0, 1\}$, and goal clauses are of the form $\bot \leftarrow B$. Templog does not deal with eventualities because the connective $\Diamond$ appears only in clause bodies. As can be appreciated in the recursive definition of bodies, the nesting of connectives in Templog clauses is not as restricted as in TeDiLog. Therefore, the structure of clauses is considerably more complex in Templog than in TeDiLog. For example, we do not allow the connective $\Diamond$ to prefix a conjunction of atoms. Since this normal form of Templog clauses is not well suited for resolution, the notion of canonical body is additionally considered in Templog. A canonical body is a body in which occurrences of the connectives $\wedge$ and $\Diamond$ cannot appear in the scope of the connective $\circ$ and every atom of the form $\circ^i A$ is in the scope of the least possible numbers of $\Diamond$. The equivalences $\circ(\varphi \wedge \psi) \equiv \circ\varphi \wedge \circ\psi$, $\circ\Diamond\psi \equiv \Diamond\circ\psi$ and $\Diamond(\Diamond\circ\varphi \wedge \Diamond\psi) \equiv \Diamond(\Diamond\varphi \wedge \psi)$ are used to obtain the canonical form of bodies. However, although the bodies of the premises are in canonical form, the resolvent obtained by a resolution application may yield a clause whose body is not in canonical form, hence a transformation to obtain the canonical form may be required after each resolution application. The resolution procedure TSLD ([12]) consists of eight rules obtained by considering all the possible cases in which temporal atoms of a program clause and a goal clause can be resolved. For instance, we depict here one of the rules

$$\frac{\Box(\circ^j A \leftarrow B_0) \qquad\qquad \bot \leftarrow B_1 \wedge \Diamond(B_2 \wedge \circ^i A \wedge B_3) \wedge B_4}{\bot \leftarrow B_1 \wedge \Diamond(\circ^{j-i} B_2 \wedge B_0 \wedge \circ^{j-i} B_3) \wedge B_4} \text{ where } j \geq i$$

This resolution rule states that a program clause of the form $\Box(\circ^j A \leftarrow B_0)$ is resolved with a goal clause of the form $\bot \leftarrow B_1 \wedge \Diamond(B_2 \wedge \circ^i A \wedge B_3) \wedge B_4$ and the resolvent $\bot \leftarrow B_1 \wedge \Diamond(\circ^{j-i} B_2 \wedge B_0 \wedge \circ^{j-i} B_3) \wedge B_4$ is obtained, whenever $j \geq i$. Note that $A$ is a classical atom. The Templog resolution procedure does not follow the state by state forward reasoning approach and, consequently, it does not use any rule similar to our rule $(Unx)$. As already mentioned in Section 5.1, the satisfiability of a Templog program can be reduced to the satisfiability of a (possibly infinite) classical logic program. This is easily made by considering, for instance, that a clause of the form $\circ^i A \leftarrow \Diamond B$ can be expressed by means of the infinite set of clauses $\{\circ^i A \leftarrow \circ^j B \mid j \geq 0\}$ and, in the same way, a clause of the form $(\Box \circ^i A) \leftarrow B$ can be expressed by means of the infinite set of clauses $\{(\circ^{j+i} A) \leftarrow B \mid j \geq 0\}$. This approach is possible neither when the connectives $\Diamond$ and $\mathcal{U}$ appear in the head of a clause nor when the connectives $\Box$ and $\mathcal{R}$ appear in the body. For instance, note that a clause of the form $\Diamond A \leftarrow B$ should be replaced with a unique clause $\circ^k A \leftarrow B$ but the value of such $k$ is unknown. As a consequence, the minimal model characterization of Templog (see [12]) is a straightforward adaptation of the classical case. Unlike Templog, TeDiLog does not have the classical Minimal Model Property (MMP in short). The presence of the connectives $\Diamond$ and $\mathcal{U}$ in clause heads and $\Box$ and $\mathcal{R}$ in clause bodies (see [101]) as well as the use of disjunction in clause heads (see e.g. [89]) prevent from having such property. The compensation for the loss of the MMP is that TeDiLog is much more expressive than the propositional fragment of Templog.

In order to study Templog's expressiveness, Baudinet considers in [12, 14] the propositional fragment TL1 where the connective $\diamond$ is not allowed at all and $\square$ is not allowed in clause heads. Consequently, TL1 program clauses are of the form $\square^b(\circ^i A_0 \leftarrow \circ^{j_1} A_1 \wedge \ldots \wedge \circ^{j_n} A_n)$ where $b \in \{0, 1\}$ and $n \geq 0$ and goal clauses are of the form $\perp \leftarrow \circ^{j_1} A_1 \wedge \ldots \wedge \circ^{j_n} A_n$ where $n \geq 0$. Baudinet shows that the expressiveness of TL1 and propositional Templog is the same. On one hand, Templog clauses of the form $\square \circ^i p \leftarrow B$ can be expressed without using the connective $\square$ by introducing a fresh propositional variable. So that, the above program clause can be expressed by means of the program clauses $\{q \leftarrow B, \square(\circ^i p \leftarrow q), \square(\circ q \leftarrow q)\}$ where $q$ is fresh. On the other hand, each element of the form $\diamond \circ^i p$ in a body of a clause, can be substituted by a fresh propositional symbol $q$ and then the clauses that define the meaning of $q$ would be added: $\{\square(q \leftarrow \circ^i p), \square(q \leftarrow \circ q)\}$. Moreover, Baudinet shows that, for instance, it is possible to define, in TL1, a predicate that holds exactly when $p \mathcal{U} q$ holds, whereas the connective $\mathcal{U}$ is not expressible in temporal logic with only $\circ$, $\square$ and $\diamond$ (see [80]). So that, there are predicates that can be defined by using TL1 but are inexpressible in temporal logic. Baudinet also shows that, for instance, the connective $\square$ is not expressible in Templog, in the sense that is not possible to prove $\square p$ or to write a Templog program defining a predicate that would hold exactly when $\square p$ holds. This last result proves that TeDiLog is more expressive than (propositional) Templog, because in TeDiLog $\square p$ can be proved, as has been shown in Example 5.4.5 (Figure 5.15).

### *5.5.2 Chronolog: Wadge [127] and Orgun [97, 99]*

In Chronolog ([127, 97, 99]) the only temporal operators are the unary connectives first and next. The connective first serves to refer to the state $s_0$. Therefore the connective $\square$ is not needed to differentiate between always- and now-clauses. The TeDiLog now-clauses $p \leftarrow \circ q$, $\square p \leftarrow \circ q$ and $p \leftarrow \diamond \circ q \wedge r$ can be expressed in Chronolog as $\mathsf{first}\, p \leftarrow \mathsf{first}\, \mathsf{next}\, q$, $p \leftarrow \mathsf{first}\, \mathsf{next}\, q$ and $\mathsf{first}\, p \leftarrow \mathsf{next}\, q \wedge \mathsf{first}\, r$, respectively. The TeDiLog always-clause $\square(p \leftarrow \circ q)$ can be expressed in Chronolog as $p \leftarrow \mathsf{next}\, q$. Note that in the Chronolog clauses above, there is a hidden temporal information not made explicit by means of temporal connectives. Regarding always-clauses of the form $\square(\square p \leftarrow \circ q)$ and $\square(s \leftarrow \diamond r)$, the translations pointed out to obtain TL1 clauses in the previous subsection must be considered for $\square p$ and $\diamond r$. Consequently, intricate sets of Chronolog clauses are needed for expressing interesting properties. In TeDiLog, the explicit use of temporal connectives, together with the fact that such connectives are more expressive, facilitates readability and understanding of program and goal clauses. In [14], Baudinet shows –by means of TL1– that Templog and Chronolog have the same expressive power. Hence Chronolog can be considered as a syntactical variant of Templog. In fact, Templog and Chronolog also coincide in the metalogical properties of minimal model existence and fixpoint characterization. The resolution procedure TiSLD that defines the operational semantics of Chronolog, applies the resolution rule to *rigid instances* of program clauses and goal clauses, which are formed by atoms of the form $\mathsf{first}\, \mathsf{next}^n\, p$ with $n \geq 0$. In [99], the inclusion of the temporal connectives $\diamond$ and $\square$ is discussed. However, by taking into account the results presented in [101], and in order to keep the metalogical properties of Chronolog, only the use of $\diamond$ in clause bodies and $\square$ in clause heads is proposed. This extension would yield a language that would be (syntactically) very similar to Templog. However, the expressive power would remain unchanged. The disjunctive extension presented in [68] combines Chronolog with the Disjunctive LP paradigm. Therefore, only the temporal connectives first and next are used and the results obtained in the

Disjunctive Logic Programming paradigm are extended to the language presented in [68] in the same way that the results obtained in classical Logic Programming are extended to Chronolog.

### 5.5.3 Temporal Prolog: Gabbay [55]

Gabbay's Temporal Prolog allows eventuality literals in clause heads but not in clause bodies. In particular, $\diamond$ is allowed in clause heads but $\square$ is not allowed in clause bodies. A program clause is either a now-clause $H \leftarrow B$ or an always clause $\square(H \leftarrow B)$. The head $H$ is either a classical atom $A$ or a formula of the form $\circ\diamond C$ where $C$ is a conjunction of now-clauses. The body $B$ is a classical atom $A$, a conjunction of bodies or a formula of the form $\circ\diamond B'$ where $B'$ is a body. A goal clause is of the form $\bot \leftarrow B$ where $B$ is a body. Additionally, a connective to express "sometime in the past" is also used. So that, the clausal form of Gabbay's Temporal Prolog is more complex than ours. In particular, the nesting of connectives is not so restricted as in TeDiLog. Although eventuality literals are allowed in clause heads, the way of dealing with them is very different from our method. For instance, given a goal of the form $\bot \leftarrow \circ\diamond p$ the resolution procedure tries to find a program clause whose head is either $p$ or $\circ\diamond p$. If such clause is found, a forward jump is produced. The resolution procedure of TeDiLog is based on a state by state forward reasoning and eventualities are dealt with by means of the context-dependent rules which do not allow to indefinitely postpone the fulfillment of such eventualities. As mentioned above, unlike in TeDiLog, the connective $\square$ is not allowed in clause bodies, hence TeDiLog is more expressive. For Gabbay's Temporal Prolog the MMP does not hold because of the use of eventualities in clauses heads. Additionally, the completeness proof of the resolution procedure is not provided. The IFT-resolution procedure for TeDiLog is complete.

### 5.5.4 MetateM: Barringer et al. [9]

MetateM programs are sets of clauses in the Separated Normal Form (SNF), where clauses are of the form $\varphi \rightarrow \psi$ such that $\varphi$ is a conjunction of propositional literals and $\psi$ is either of the form $\diamond\chi$ –where $\chi$ is a propositional literal– or a disjunction of propositional literals prefixed by the connective $\circ$ (see also Subsection 4.8.5). MetateM is as expressive as TeDiLog and complete for full PLTL. However, MetateM is based on the imperative future approach and is not based on resolution. Regarding execution, at each step the MetateM execution procedure must build the next state by choosing to make true one proposition from the $\psi$ part of each clause for which the $\varphi$ part is true in the current state. In this way, a sequence of states is produced with the aim of building a model for the program. Choices that lead to inconsistency must be repaired by means of backtracking, which serves to choose another disjunct from the corresponding $\psi$ part. Additionally, the finite-model property is used to calculate an upper bound of forward chaining steps and, in this way, to detect model construction processes where the fulfillment of an eventuality is being indefinitely delayed. Such upper bound, in the worst case, is $2^{5|\Pi|}$ where $|\Pi|$ is the size of the initial program $\Pi$ (see [9] and Subsection 6.2.4 in [44]). The IFT-resolution procedure underlying TeDiLog does need neither backtracking nor the calculation of upper bounds. As in TeDiLog, the execution mechanism of MetateM must make sure that the satisfaction of an eventuality is not continually postponed. For a clause $\varphi \rightarrow \circ\diamond p$, it is possible to make true $p$ or to make true $\diamond p$ in the next state. If there are two clauses of the form $\varphi \rightarrow \circ\diamond p$ and $\varphi' \rightarrow \circ\diamond\neg p$ such that $\varphi$ and $\varphi'$ are satisfied in every state, it is necessary to satisfy $p$ and $\neg p$ in an interleaved way. Therefore, fairness is required when deciding which eventuality to satisfy. This is handled by keeping an ordered list of eventualities (see Subsection 6.2.7 in [44]).

### *5.5.5 Clausal Temporal Resolution for* PLTL*: Fisher [40]*

The clausal temporal resolution method introduced in [40] (see also [45]) is complete for full PLTL. Our clausal normal form is different from the Separated Normal Form used in that method but the crucial difference of our method with respect to that method is that TeDiLog's resolution mechanism is powerful enough to deal with eventualities without requiring invariant generation. See also Subsection 4.8.5 for more details.

# 6. CONCLUSIONS

This chapter reviews our central results and primary contributions, lists our publications and relevant research activity related to the results that appear in this thesis and proposes areas for future research.

## *6.1 Results and Contributions*

In this section, we review the results and contributions that have been presented in previous chapters.

We have introduced tableau, sequent and resolution methods that differ from previously existing systems in the way eventualities are dealt with. Traditional two-pass temporal tableaux and the previously existing one-pass tableau method presented by Schwendimann in [117] need to check the fulfillment of eventualities in cyclic sequences of states. By contrast, our one-pass tableau method TTM includes a rule that prevents from indefinitely delaying the fulfillment of eventualities. As a consequence, TTM generates classical-like tableaux. In the case of unsatisfiable sets of formulas, closed branches whose last nodes contain a formula and its negation are obtained in TTM. Regarding satisfiable sets of formulas, when an open cyclic branch is marked as expanded (i.e, sufficiently enlarged) in TTM, that branch yields a model. It is worth remarking that given an unsatisfiable set of formulas, the tableau method in [117] may yield –unlike in classical tableaux– cyclic and non-fulfilling (closed) branches whose last nodes do not contain a formula and its negation. In order to detect that a cyclic branch is non-fulfilling (i.e. closed) and that, consequently, it cannot yield a model, an additional handling of information is required in [117] because accessible branches must be checked to rule out the existence of a fulfilling cycle that may involve more than one branch. In the case of satisfiable sets of formulas, a cyclic (open) branch –that cannot be enlarged– may not yield a model by itself in the Schwendimann's tableau system because the fulfillment of eventualities may depend on more than one cyclic branch. The systematic tableau algorithm that we provide gives rise to a decision procedure for PLTL. On the basis of this new temporal deductive approach, we have defined two cut-free and, in particular, invariant-free finitary sequent calculi TTC and GTC that are also weakening- and contraction-free. These tableau and sequent systems allow us to prove that the classical duality between tableaux and sequents holds also for temporal logic.

By adapting the idea behind the dual tableau and sequent systems to the resolution framework, we have presented a new method for temporal resolution that is sound and complete for PLTL and does not require invariant generation. This feature is a crucial difference of our method with respect to the clausal resolution method introduced in [40] (see also [45]) which needs to generate invariant formulas for solving eventualities. We have provided the conversion of any formula to clausal form, a resolution system called TRS that extends classical resolution, and an easily implementable algorithm that decides the satisfiability of any set of clauses. Moreover, together with its yes/no answer, the algorithm provides an (un/)satisfiability proof. That is, either a systematic refutation or a canonical model of the set of clauses that has been given as input. As in the classical case, models are more easily generated from cyclic tableau branches than from cyclic resolution derivations.

On the basis of the invariant-free resolution method TRS, we have defined the propositional temporal logic programming language TeDiLog with the aim of providing a single framework in which dynamic systems can be specified, developed, validated and verified by means of executable specifications. The language TeDiLog has a purely declarative nature and mathematically defined semantics. This language is strictly more expressive than the propositional fragments of the main declarative TLP languages in the literature ([2, 12, 127, 99, 55, 68]). TeDiLog's resolution mechanism is powerful enough to deal with eventualities and dispenses with invariant generation. The most significant imperative TLP language MetateM ([9]) is as expressive as TeDiLog. However, MetateM is a very different approach that is not based on resolution and uses an upper bound to detect unsuccessful model constructions and backtracking. TeDiLog requires neither upper bounds nor backtracking. We see TeDiLog as the propositional kernel of a new generation of TLP languages based on the invariant-free temporal resolution method TRS. In this sense we hope that TeDiLog could influence the design of future TLP languages in order to incorporate more expressive temporal features and new resolution procedures for temporal reasoning.

To sum up, we have contributed new ideas to the proof-theory of PLTL. In particular, we believe that automated reasoning in temporal logic can take benefit from the systems presented in this dissertation.

## 6.2 Related Publications, Presentations and Research Activity

Below we list the publications, presentations and relevant research activity we carried out in relation to the results provided in this dissertation.

**Journal Publications**

- **Dual Systems of Tableaux and Sequents for** PLTL
  J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro and F. Orejas
  Journal of Logic and Algebraic Programming, 78(8):701–722, 2009.
  DOI 10.1016/j.jlap.2009.05.001

- **Invariant-Free Clausal Temporal Resolution**
  J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro and F. Orejas
  Journal of Automated Reasoning. To appear.
  DOI 10.1007/s10817-011-9241-2
  Published online: 2 December 2011

**Conference Proceedings**

- **A Cut-Free and Invariant-Free Sequent Calculus for** PLTL
  J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro and F. Orejas
  J. Duparc, T. A. Henzinger (eds.) Proceedings of Computer Science Logic, 21st

International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, 11-15 September 2007, volume 4646 of Lecture Notes in Computer Science, pages 481–495. Springer, 2007.
DOI: 10.1007/978-3-540-74915-8

- **Systematic Semantic Tableaux for** PLTL
  J. Gaintzarain, M. Hermo, P. Lucio and M. Navarro
  E. Pimentel (ed.) Proceedings of the 7th Spanish Conference on Programming and Languages (PROLE 2007), Zaragoza, Spain, 11-14 September 2007, Selected Papers, volume 206 of Electronic Notes in Theoretical Computer Science, pages 59-73, 2008
  DOI 10.1016/j.entcs.2008.03.075

- **A New Approach to Temporal Logic Programming**
  J. Gaintzarain and P. Lucio
  P. Lucio, G. Moreno, R. Peña (eds.) Proceedings of the 9th Spanish Conference on Programming and Languages (PROLE 2009), San Sebastián, Spain, 8-11 September 2009, pages 341–350, 2009.
  http://www.sistedes.es/ficheros/actas-conferencias/PROLE/2009.pdf
  ISBN: 978-84-692-4600-9

- **An Implementation of the Context-Based Tableau**
  J. Gaintzarain, J. A. Hernandez and P. Lucio
  P. Arenas, V. M. Gulías, P. Nogueira (eds.) Proceedings of the 11th Spanish Conference on Programming and Languages (PROLE 2011), A Coruña, Spain, 5-7 September 2011, pages 169–184, 2011.
  http://www.sistedes.es/ficheros/actas-conferencias/PROLE/2011.pdf
  ISBN: 978-84-9749-487-8

**Contributed Talk**

- **Invariant-Free Clausal Temporal Resolution**
  J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro and F. Orejas
  Workshop on Modal Fixpoint Logics 2008 (WMFL 2008)
  http://staff.science.uva.nl/ yde/mfl/
  http://staff.science.uva.nl/ yde/mfl/contributed/gaintzarain.pdf
  Institute for Logic, Language and Computation, University of Amsterdam.
  Amsterdam, The Netherlands, 25–27 March 2008

**Research Seminars**

- **Invariant-Free Clausal Temporal Resolution**
  J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro and F. Orejas
  http://www2.wmin.ac.uk/bolotoa/HSCS_SEMINARS/seminars.html
  Department of Computer Science and Software Engineering, School of Electronics and Computer Science, University of Westminster.

London, United Kingdom, 27 November 2009

- **Invariant-Free Deduction Methods for** PLTL
  P. Lucio and J. Gaintzarain
  Department of Computer Science, University of Liverpool.
  Liverpool, United Kingdom, 1 February 2011

**Research Visit**

- **Research Area: Invariant-Free Deduction Systems for Temporal Logic**
  Research Visitor: Jose Gaintzarain
  Supervisor: Alexander Bolotov
  Distributed and Intelligent Systems Research Group
  School of Electronics and Computer Science, University of Westminster.
  London, United Kingdom, from 1 October 2009 to 31 January 2010

**Journal Paper Under Review**

- **Logical Foundations for More Expressive Declarative Temporal Logic Programming Languages**
  J. Gaintzarain and P. Lucio
  Submitted (Under review)

## 6.3 Future Work

We believe that the work presented in this dissertation opens many interesting topics for future research.
The extension of our invariant-free deductive approach to more expressive logics is a wide area of work. In particular, we hope that the presented resolution method gives an opportunity to develop the first resolution method for Full Computation Tree Logic CTL$^\star$. Although the first complete tableau system for CTL$^\star$ has been recently published in [109], a resolution procedure for CTL$^\star$ is not known yet. Additionally, a tableau method based on the invariant-free deductive approach would still be valuable. The extension of TRS-resolution to the incomplete First-order Linear-time Temporal Logic (FLTL), besides its own relevance, could produce a new class of decidable fragments of FLTL along with their associated decision procedures based on TRS-resolution. For instance, one may consider the clausal FLTL-language that is obtained from our clausal language by allowing, as atoms, predicate symbols applied to first-order terms, instead of propositional variables. A syntactical restriction of this clausal FLTL-language would be decidable provided that the set of all possible different contexts –in any application of the rule $(\mathcal{U}\,Set)$– were ensured to be finite in the restricted language. Moreover, particular syntactical restrictions could allow to specialize the general TRS-procedure in order to gain efficiency (as it is done in [35, 36]). The TRS-resolution method could also be applied to other extensions of PLTL like spatial, dynamic, etc.

The development of practical automated reasoning tools based on the TTM tableau method and the TRS resolution system constitutes a broad area of present and future work. At the moment, preliminary prototypes for the TTM tableau method and the TRS resolution method are available online, respectively, in `http://www.sc.ehu.es/jiwlucap/TTM.html` and `http://www.sc.ehu.es/jiwlucap/TRS.html`. A report about the implementation of the prototype for the TTM tableau method is provided in [63]. On one hand, this prototype for TTM is a direct implementation of the systematic tableau algorithm. On the other hand, the prototype for the TRS resolution method is a direct implementation of the transformation CNF and the algorithm $\mathcal{SR}$. There is only a small amount of nondeterminism in these algorithms. Moreover, the form of nondeterminism in these algorithms is sometimes called *angelic* nondeterminism, in the sense that backtracking is not required to ensure termination. The crucial actions upon which the implementation of the systematic tableau algorithm and the algorithm $\mathcal{SR}$ depends are the fair selection of eventualities, the application of each rule, and the test for termination. We plan to gradually improve these prototypes and to compare them with other available automated reasoning tools for PLTL. In particular with the temporal resolution prover TRP++ [76] that implements the method introduced in [40]. We are also interested in comparison with the implementations of the tableau-based methods presented in [79, 117] that are available in the Logics Workbench Version 1.1 (`http://www.lwb.unibe.ch`).

The decision problem for PLTL is known to be PSPACE-complete (see e.g. [119]). The two-pass tableau method presented in [128] works in EXPTIME, hence it is optimal. The worst case complexity of our tableau and resolution methods (as well as for the tableau method and the resolution method presented, respectively, in [117] and [40]) is 2EXPTIME, and consequently suboptimal. However it has been shown by experimental analysis (see e.g. [69, 78]) that for many randomly generated formulas of some classes, the average performance of a doubly exponential algorithm can be better than the average performance of an exponential one. The reason is that, in the former the cases with high complexity rarely occur, while in the latter the cases with exponential complexity occur very often. The above mentioned classes of formulas include conjunctions of eventualities, nested eventualities, especial conjunctions of clauses in Separated Normal Form, etc. The results obtained in the empirical analysis carried out in [77] give hints about improvements to be considered for a practical implementation. Also the above mentioned possibility of searching for tractable fragments (see [35, 36]) is open. The accurate study of the complexity of the TTM tableau method and the TRS resolution method seems to be also interesting.

We are also considering the possibility of combining TRS-resolution with the one-pass tableau method TTM to produce a kind of *hyper tableaux* that would be interesting for practical implementation purposes.

The implementation of TeDiLog remains as future work. The adaptation of the prototype for the TRS resolution method (`http://www.sc.ehu.es/jiwlucap/TRS.html`) to TeDiLog is straightforward, but much experimentation is needed for optimization and improvement. The worst case complexity for TeDiLog (regarding the generation of a refutation proof) is doubly exponential.

It is well known (see [11, 12, 13, 14]) that, although logic programs are formulas of a given logic, a logic programming language may be in some respects more expressive than its underlying logic. Intuitively, a logic formula characterizes just the collection of its models whereas a logic program characterizes the collection of facts that can be inferred from it. The notion of deduction intervenes and adds the ability to express properties that are not expressible in the underlying logic. In this sense it would be interesting to compare the expressiveness of TeDiLog to other formalisms such as PLTL, automata-theoretic formalisms, quantified PLTL (i.e. QPTL),

$\mu$TL, etc. We have already tackled the issue of relating different formalisms. Concretely, in [59] we studied the translation of the propositional fragment of the logic programming language $Horn^{\supset}$ into Boolean circuits, Boolean formulas and conjunctions of propositional Horn clauses. $Horn^{\supset}$ is a logic programming language that extends usual Horn clauses by adding intuitionistic implication in goals and clause bodies.

# BIBLIOGRAPHY

[1] M. Abadi and Z. Manna. Nonclausal temporal deduction. In *Proceedings of the International Conference on Logics of Programs*, volume 193 of *LNCS*, pages 1–15. Springer, 1985.

[2] M. Abadi and Z. Manna. Temporal logic programming. In *Proceedings of the International Symposium on Logic Programming*, pages 4–16. IEEE Computer Society Press, 1987.

[3] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

[4] M. Abadi and Z. Manna. Nonclausal deduction in first-order temporal logic. *Journal of the ACM*, 37:279–317, 1990.

[5] P. Abate, R. Goré, and F. Widmann. One-pass tableaux for computation tree logic. In *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 4790 of *LNCS*, pages 32–46. Springer, 2007.

[6] F. Aguado, P. Cabalar, G. Pérez, and C. Vidal. Strongly equivalent temporal logic programs. In *Proceedings of the 11th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 5293 of *LNAI*, pages 8–20. Springer, 2008.

[7] F. Aguado, P. Cabalar, G. Pérez, and C. Vidal. Loop formulas for splitable temporal logic programs. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 6645 of *LNCS*, pages 80–92. Springer, 2011.

[8] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In *Proceedings of the Temporal Logic in Specification*, volume 398 of *LNCS*, pages 62–74. Springer, 1987.

[9] H. Barringer, M. Fisher, D. M. Gabbay, G. Gough, and R. Owens. MetateM: A framework for programming in temporal logic. In *Proceedings of the REX (Research and Education in Concurrent Systems) Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *LNCS*, pages 94–129. Springer, 1989.

[10] M. Baudinet. On the semantics of temporal logic programming. Technical Report CS-TR-88-1203, Department of Computer Science, Stanford University, California, USA, 1988.
ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/88/1203/CS-TR-88-1203.pdf.

[11] M. Baudinet. *Logic Programming Semantics: Techniques and Applications*. PhD thesis, Department of Computer Science, Stanford University, California, USA, 1989.

[12] M. Baudinet. Temporal logic programming is complete and expressive. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 267–280. ACM Press, 1989.

[13] M. Baudinet. A simple proof of the completeness of temporal logic programming. In *Intensional Logics for Programming*, pages 51–83. Oxford University Press, 1992.

[14] M. Baudinet. On the expressiveness of temporal logic programming. *Information and Computation*, 117(2):157–180, 1995.

[15] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Highly Dependable Software*, volume 58 of *Advances in Computers*. Academic Press, 2003.

[16] A. Bolotov and A. Basukoski. A clausal resolution method for branching-time logic $\mathsf{ECTL}^{+}$. *Annals of Mathematics and Artificial Intelligence*, 46(3):235–263, 2006.

[17] A. Bolotov and A. Basukoski. A clausal resolution method for extended computation tree logic $\mathsf{ECTL}$. *Journal of Applied Logic*, 4(2):141–167, 2006.

[18] A. Bolotov and M. Fisher. A clausal resolution method for $\mathsf{CTL}$ branching-time temporal logic. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):77–93, 1999.

[19] K. Brünnler. Deep sequent systems for modal logic. *Archive for Mathematical Logic*, 48(6):551–577, 2009.

[20] K. Brünnler and M. Lange. Cut-free sequent systems for temporal logic. *The Journal of Logic and Algebraic Programming*, 76(2):216–225, 2008.

[21] C. Brzoska. Temporal logic programming and its relation to constraint logic programming. In *Proceedings of the International Symposium on Logic Programming (ISLP)*, pages 661–677. MIT Press, 1991.

[22] C. Brzoska. Temporal logic programming with bounded universal modality goals. In *Proceedings of the 10th International Conference on Logic Programming (ICLP)*, pages 239–256. MIT Press, 1993.

[23] C. Brzoska. Temporal logic programming in dense time. In *Proceedings of the International Logic Programming Symposium (ILPS)*, pages 303–317. MIT Press, 1995.

[24] C. Brzoska. Temporal logic programming with metric and past operators. In *Proceedings of the IJCAI'93 Workshop on Executable Modal and Temporal Logics*, volume 897 of *LNAI*, pages 21–39. Springer, 1995.

[25] C. Brzoska. Programming in metric temporal logic. *Theoretical Computer Science*, 202(1-2):55–125, 1998.

[26] C. Brzoska and K. Schäfer. Temporal logic programming applied to image sequence evaluation. In *Logic Programming: Formal Methods and Practical Applications*, Studies in Computer Science and Artificial Intelligence, pages 381–395. Elsevier Science B.V./North-Holland, 1995.

[27] A. Cau, H. Zedan, N. Coleman, and B. C. Moszkowski. Using ITL and Tempura for large-scale specification and simulation. In *Proceedings of the 4th Euromicro Workshop on Parallel and Distributed Processing (PDP)*, pages 493–500. IEEE Computer Society Press, 1996.

[28] A. R. Cavalli. A method of automatic proof for the specification and verification of protocols. *Computer Communication Review*, 14(2):100–106, 1984.

[29] A. R. Cavalli and L. Fariñas del Cerro. A decision method for linear temporal logic. In *Proceedings of the 7th International Conference on Automated Deduction (CADE)*, volume 170 of *LNCS*, pages 113–127. Springer, 1984.

[30] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[31] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 2001.

[32] A. Degtyarev, M. Fisher, and B. Konev. A simplified clausal resolution procedure for propositional linear-time temporal logic. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 2381 of *LNCS*, pages 85–99. Springer, 2002.

[33] C. Dixon. Search strategies for resolution in temporal logics. In *Proceedings of the 13th International Conference on Automated Deduction (CADE)*, volume 1104 of *LNCS*, pages 673–687. Springer, 1996.

[34] C. Dixon and M. Fisher. The set of support strategy in temporal resolution. In *Proceedings of the 5th International Workshop on Temporal Representation and Reasoning (TIME)*, pages 113–120. IEEE Computer Society Press, 1998.

[35] C. Dixon, M. Fisher, and B. Konev. Is there a future for deductive temporal verification? In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 11–18. IEEE Computer Society Press, 2006.

[36] C. Dixon, M. Fisher, and M. Reynolds. Execution and proof in a Horn-clause temporal logic. In *Advances in Temporal Logic*, pages 413–433. Kluwer Academic Publishers, 2000.

[37] Z. Duan, X. Yang, and M. Koutny. Semantics of framed temporal logic programs. In *Proceedings of the 21st International Conference on Logic Programming (ICLP)*, volume 3668 of *LNCS*, pages 356–370. Springer, 2005.

[38] Z. Duan, X. Yang, and M. Koutny. Framed temporal logic programming. *Science of Computer Programming*, 70(1):31–61, 2008.

[39] E. Eder. *Relative complexities of first order calculi*. Artificial intelligence. Vieweg, 1992.

[40] M. Fisher. A resolution method for temporal logic. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 99–104. Morgan Kaufmann, 1991.

[41] M. Fisher. A normal form for first-order temporal formulae. In *Proceedings of the 11th International Conference on Automated Deduction (CADE)*, volume 607 of *LNCS*, pages 370–384. Springer, 1992.

[42] M. Fisher. Concurrent MetateM – A language for modeling reactive systems. In *Proceedings of the Conference on Parallel Architectures and Languages, Europe (PARLE)*, volume 694 of *LNCS*, pages 185–196. Springer, 1993.

[43] M. Fisher. Implementing BDI-like systems by direct execution. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 316–321. Morgan Kaufmann, 1997.

[44] M. Fisher. *An Introduction to Practical Formal Methods Using Temporal Logic*. John Wiley & Sons, Ltd, 2011.

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

[46] M. Fisher, D. Gabbay, and L. Vila, editors. *Handbook of Temporal Reasoning in Artificial Intelligence, volume 1 of Foundations of Artificial Intelligence*. Elsevier Press, 2005.

[47] M. Fisher and C. Ghidini. Executable specifications of resource-bounded agents. *Autonomous Agents and Multi-Agent Systems*, 21(3):368–396, 2010.

[48] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishing Company, 1983.

[49] T. W. Frühwirth. Annotated constraint logic programming applied to temporal reasoning. In *Proceedings of the 6th International Symposium on Programming Language Implementation and Logic Programming (PLILP)*, volume 844 of *LNCS*, pages 230–243, 1994.

[50] T. W. Frühwirth. Temporal logic and annotated constraint logic programming. In *Proceedings of the IJCAI'93 Workshop on Executable Modal and Temporal Logics*, volume 897 of *LNAI*, pages 58–68. Springer, 1995.

[51] T. W. Frühwirth. Temporal annotated constraint logic programming. *Journal of Symbolic Computation*, 22(5/6):555–583, 1996.

[52] M. Fujita, S. Kono, H. Tanaka, and T. Moto-Oka. Tokio: Logic programming language based on temporal logic and its compilation to Prolog. In *Proceedings of the 3rd International Conference on Logic Programming (ICLP)*, volume 225 of *LNCS*, pages 695–709, 1986.

[53] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 163–173. ACM Press, 1980.

[54] D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Proceedings of the Colloquium on Temporal Logic in Specification*, volume 398 of *LNCS*, pages 409–448, 1987.

[55] D. M. Gabbay. Modal and temporal logic programming. In *Temporal Logics And Their Application*, pages 197–237. Academic Press, 1987.

[56] D. M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal logic (vol. 1): mathematical foundations and computational aspects*. Oxford University Press, 1994.

[57] D. M. Gabbay, M. A. Reynolds, and M. Finger. *Temporal logic (vol. 2): mathematical foundations and computational aspects*. Oxford University Press, 2000.

[58] J. Gaintzarain, M. Hermo, P. Lucio, and M. Navarro. Systematic semantic tableaux for PLTL. In *Proceedings of the 7th Spanish Conference on Programming and Languages (PROLE), Selected Papers*, volume 206 of *Electronic Notes in Theoretical Computer Science*, pages 59–73, 2008.

[59] J. Gaintzarain, M. Hermo, P. Lucio, and M. Navarro. Translating propositional extended conjunctions of Horn clauses into Boolean circuits. *Theoretical Computer Science*, 411(16-18):1723–1733, 2010.

[60] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. A cut-free and invariant-free sequent calculus for PLTL. In *Proceedings of the Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL*, volume 4646 of *LNCS*, pages 481–495. Springer, 2007.

[61] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. Dual systems of tableaux and sequents for PLTL. *The Journal of Logic and Algebraic Programming*, 78(8):701–722, 2009.

[62] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. Invariant-free clausal temporal resolution. *Journal of Automated Reasoning*, 2011. To appear. DOI 10.1007/s10817-011-9241-2. Available online.

[63] J. Gaintzarain, J. A. Hernandez, and P. Lucio. An implementation of the context-based tableau. In *Proceedings of the 11th Spanish Conference on Programming and Languages (PROLE)*, pages 169–184, 2011.
http://www.sistedes.es/Actas/2011_coruna/PROLE/PROLE/S6/17_article.pdf.

[64] J. Gaintzarain and P. Lucio. A new approach to temporal logic programming. In *Proceedings of the 9th Spanish Conference on Programming and Languages (PROLE)*, pages 341–350, 2009. http://www.sistedes.es/PROLE/actas_Prole2009.pdf.

[65] G. Gentzen. Untersuchungen über das Logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1934. English translation in [66], pages 68–131.

[66] G. Gentzen. *The Collected Papers of Gerhard Gentzen*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1969. Edited by M. E. Szabo.

[67] M. Gergatsoulis. Temporal and modal logic programming languages. In *Encyclopedia of Microcomputers*, volume 27, pages 393–408. CRC Press, 2001.

[68] M. Gergatsoulis, P. Rondogiannis, and T. Panayiotopoulos. Temporal disjunctive logic programming. *New Generation Computing*, 19(1):87–102, 2000.

[69] V. Goranko, A. Kyrilov, and D. Shkatov. Tableau tool for testing satisfiability in LTL: Implementation and experimental analysis. In *Proceedings of the 6th Workshop on Methods for Modalities*, volume 262 of *Electronic Notes in Theoretical Computer Science*, pages 113–125, 2010.

[70] V. Goranko and D. Shkatov. Tableau-based decision procedure for full coalitional multiagent temporal-epistemic logic of linear time. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) - Volume 2*, pages 969–976. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[71] R. Goré. Tableau methods for modal and temporal logics. In *Handbook of Tableau Methods*, pages 297–396. Kluwer Academic Publishers, 1999.

[72] R. Goré and F. Widmann. An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In *Proceedings of the 22nd International Conference on Automated Deduction (CADE)*, volume 5663 of *LNCS*, pages 437–452. Springer, 2009.

[73] G. D. Gough. Decision procedures for temporal logic. Master's thesis, Technical report UMCS-89-10-1, Department of Computer Science, University of Manchester, UK, 1984.

[74] T. Hrycej. Temporal prolog. In *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI)*, pages 296–301. Pitmann Publishing, 1988.

[75] T. Hrycej. A temporal extension of Prolog. *Journal of Logic Programming*, 15(1 & 2):113–145, 1993.

[76] U. Hustadt and B. Konev. Trp++2.0: A temporal resolution prover. In *Proceedings of the 19th International Conference on Automated Deduction (CADE)*, volume 2741 of *LNCS*, pages 274–278. Springer, 2003.

[77] U. Hustadt and R. A. Schmidt. An empirical analysis of modal theorem provers. *Journal of Applied Non-Classical Logics*, 9(4):479–522, 1999.

[78] U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR)*, pages 533–544. Morgan Kaufmann, 2002.

[79] G. Janssen. *Logics for Digital Circuit Verification: Theory, Algorithms, and Applications*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1999.

[80] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Department of Computer Science, University of California at Los Angeles, California, USA, 1968.

[81] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In *Proceedings of the 5th International Conference on Computer Aided Verification (CAV)*, volume 697 of *LNCS*, pages 97–109, 1993.

[82] S. Kono. A combination of clausal and non clausal temporal logic programs. In *Proceedings of the IJCAI'93 Workshop on Executable Modal and Temporal Logics*, volume 897 of *LNAI*, pages 40–57. Springer, 1995.

[83] S. Kono, T. Aoyagi, M. Fujita, and H. Tanaka. Implementation of temporal logic programming language Tokio. In *Proceedings of the 4th Conference on Logic Programming (LP)*, volume 221 of *LNCS*, pages 138–147, 1985.

[84] R. Kontchakov, C. Lutz, F. Wolter, and M. Zakharyaschev. Temporalising tableaux. *Studia Logica*, 76(1):91–134, 2004.

[85] S. Konur. A survey on temporal logics. *CoRR*, http://arxiv.org/abs/1005.3199, 2010.

[86] F. Kröger and S. Merz. *Temporal Logic and State Systems*. Springer, 2008.

[87] O. Lichtenstein and A. Pnueli. Propositional temporal logics: Decidability and completeness. *Logic Journal of the IGPL (Interest Group in Pure and Applied Logic)*, 8(1):55–85, 2000.

[88] J. W. Lloyd. *Foundations of Logic Programming, 1st Edition*. Springer, 1984.

[89] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of disjunctive logic programming*. MIT Press, 1992.

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

[91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems – Safety*. Springer, 1995.

[92] S. Merz. Decidability and incompleteness results for first-order temporal logics of linear time. *Journal of Applied Non-Classical Logics*, 2(2):139–156, 1992.

[93] S. Merz. Efficiently executable temporal logic programs. In *Proceedings of the IJCAI'93 Workshop on Executable Modal and Temporal Logics*, volume 897 of *LNAI*, pages 69–85. Springer, 1995.

[94] B. C. Moszkowski. *Executing temporal logic programs*. Cambridge University Press, 1986.

[95] B. C. Moszkowski. Compositional reasoning using interval temporal logic and Tempura. In *Compositionality: The Significant Difference. International Symposium, COMPOS'97. Revised Lectures*, volume 1536 of *LNCS*, pages 439–464. Springer, 1998.

[96] H. Nakamura, M. Nakai, S. Kono, M. Fujita, and H. Tanaka. Logic design assistence using temporal logic based language Tokio. In *Proceedings of the 8th Conference on Logic Programming (LP)*, volume 485 of *LNAI*, pages 174–183. Springer, 1989.

[97] M. A. Orgun. *Intensional Logic Programming*. PhD thesis, Department of Computer Science, University of Victoria, British Columbia, Canada, 1991.

[98] M. A. Orgun. Temporal and modal logic programming: An annotated bibliography. *SIGART Bulletin*, 5(3):52–59, 1994.

[99] M. A. Orgun. Foundations of linear-time logic programming. *International Journal of Computer Mathematics*, 58(3-4):199–219, 1995.

[100] M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In *Proceedings of the 1st International Conference on Temporal Logic (ICTL)*, volume 827 of *LNCS*, pages 445–479. Springer, 1994.

[101] M. A. Orgun and W. W. Wadge. Towards a unified theory of intensional logic programming. *Journal of Logic Programming*, 13(4):413–440, 1992.

[102] M. A. Orgun and W. W. Wadge. Extending temporal logic programming with choice predicates non-determinism. *Journal of Logic and Computation*, 4(6):877–903, 1994.

[103] M. A. Orgun, W. W. Wadge, and W. Du. Chronolog (Z): Linear-time logic programming. In *Proceedings of the 5th International Conference on Computing and Information (ICCI)*, pages 545–549. IEEE Computer Society Press, 1993.

[104] B. Paech. Gentzen-systems for propositional temporal logics. In *Proceedings of the 2nd Workshop on Computer Science Logic (CSL)*, volume 385 of *LNCS*, pages 240–253, 1988.

[105] R. Pliuskevicius. Investigation of finitary calculus for a discrete linear time logic by means of infinitary calculus. In *Baltic Computer Science, selected papers*, volume 502 of *LNCS*, pages 504–528, 1991.

[106] R. Pliuskevicius. Logical foundation for logic programming based on first order linear temporal logic. In *Proceedings of the First (1990) and Second (1991) Russian Conference on Logic Programming (RCLP)*, volume 592 of *LNCS*, pages 391–406, 1992.

[107] A. Raffaetà and T. W. Frühwirth. Two semantics for temporal annotated constraint logic programming. In *Proceedings of the 12th International Symposium on Languages for Intensional Programming (ISLIP)*, pages 126–140. World Scientific Press, 1999.

[108] H. Reichgelt. Semantics for reified temporal logic. In *Advances in Artificial Intelligence*, pages 49–61. John Wiley & Sons, Ltd., 1987.

[109] M. Reynolds. A tableau for $\mathsf{CTL}^\star$. In *Proceedings of the 2nd World Congress on Formal Methods (FM)*, volume 5850 of *LNCS*, pages 403–418. Springer, 2009.

[110] M. Reynolds and C. Dixon. Theorem-proving for discrete temporal logic. In *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*, pages 279–314. Elsevier Press, 2005.

[111] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[112] P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. Cactus: A branching-time logic programming language. In *Proceedings of the 1st International Joint Conference on Qualitative and Quantitative Practical Reasoning (ECSQARU-FAPR)*, volume 1244 of *LNCS*, pages 511–524. Springer, 1997.

[113] P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. Branching-time logic programming: The language Cactus and its applications. *Computer Languages*, 24(3):155–178, 1998.

[114] T. Sakuragawa. Temporal Prolog. Technical report, Kyoto University, 1986. http://repository.kulib.kyoto-u.ac.jp/dspace/bitstream/2433/99379/1/0586-16.pdf.

[115] U. Schöning. *Logic for Computer Scientists*. Birkhäuser, 1989.

[116] K. Schutte. Schluweisen-kalkule der pradikatenlogik. *Mathematische Annalen*, 122:47–65, 1950.

[117] S. Schwendimann. A new one-pass tableau calculus for PLTL. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 1397 of *LNCS*, pages 277–292, 1998.

[118] Y. Shoham. Reified temporal logics: Semantical and ontological considerations. In *Proceedings of the 7th European Conference on Artificial Intelligence (ECAI)*, pages 183–190. North-Holland, 1986.

[119] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[120] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[121] A. Szalas. Temporal logic of programs: A standard approach. In *Time and Logic. A Computational Approach*, pages 1–50. UCL Press Ltd., 1995.

[122] A. Szalas and L. Holenderski. Incompleteness of first-order temporal logic with until. *Theoretical Computer Science*, 57:317–325, 1988.

[123] W. Tait. Normal derivability in classical logic. In *The Syntax and Semantics of Infinitary Languages*, volume 72 of *Lecture Notes in Mathematics*, pages 204–236. Springer, 1968.

[124] C.-S. Tang. Toward a unified logical basis for programming languages. Technical Report STAN-CS-81-865, Department of Computer Science, Stanford University, California, USA, 1981.
ftp://db.stanford.edu/pub/cstr/reports/cs/tr/81/865/CS-TR-81-865.pdf.

[125] C.-S. Tang. Toward a unified logical basis for programming languages. In *Proceedings of the 9th World Computer Congress on Information Processing (IFIP–International Federation for Information Processing)*, pages 425–429. North-Holland/IFIP, 1983.

[126] G. Venkatesh. A decision method for temporal logic based on resolution. In *Proceedings of the 5th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 206 of *LNCS*, pages 272–289. Springer, 1985.

[127] W. W. Wadge. Tense logic programming: a respectable alternative. In *Proceedings of the International Symposium on Lucid and Intensional Programming*, pages 26–32, 1988.

[128] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.

[129] X. Yang, Z. Duan, and Q. Ma. Axiomatic semantics of projection temporal logic programs. *Mathematical Structures in Computer Science*, 20(5):865–914, 2010.