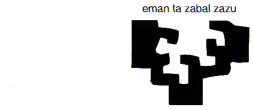
Universidad del País Vasco
Euskal Herriko Unibertsitatea

PhD dissertation

Certificates for Decision Problems in Temporal Logic using Context-Based Tableaux and Sequent Calculi

Alex Abuin Yepes

2022



Computer Languages and Systems
Lenguajes y Sistemas Informáticos
Lengoaia eta Sistema Informatikoak

Dependable Embedded Systems
Sistemas Embebidos Confiables
Sistema Txertatu Fidagarriak

Certificates for Decision Problems in Temporal Logic using Context-Based Tableaux and Sequent Calculi

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

Alex Abuin Yepes

Advisors:
Dr. Montserrat Hermo Huguet (UPV/EHU)
Dr. Jorge Parra Molina(Ikerlan)

2022

Happiness can be found, even in the darkest of times,
if one only remembers to turn on the light.
— Albus Percival Wulfric Brian Dumbledore

ACKNOWLEDGEMENTS

I would like to start by thanking all the technical and emotional support given to me by the University of the Basque Country and Ikerlan. Thanks to all the hours of work in front of a blackboard and all the calls from Montse, Unai and Paqui, this work has come to a good end. Thank you for guiding me and accompanying me in this learning process, taking me by the hand and not letting me go at any time. I would also like to mention Franco Raimondi and Alexander Bolotov for hosting me in London and giving me their time. In addition, I would like to thank Alexander for his continuous follow-up and feedback and for insisting on the word formulae as the plural of formula (it is more elegant).

During this process, I can't forget my colleagues from Ikerlan and from the university. People like Unai, Maider, Mikel, Jorge, Lorea, Aitor, Itziar, Ana, Goiuri, Javi, Iñigo, Alex II (Bea's brother), Laura, Xabi and many more - I'm going to leave out a lot of people - have made my day-to-day life more bearable. Thank you for contributing to the good working atmosphere that in many cases has ended in friendship, cool trips and wonderful coffees. Thank you for helping me when I have asked for help and above all when you have reached out to me without my asking for it.

And how can I talk about good atmosphere and friendship without mentioning Ariane, Aritz, Ana and Eneko. We have spent many hours together and you have made me happy on the sad days, entertained me during the most boring periods, listened when I needed to talk and helped me when I needed it. You are the workers that every company should want and more importantly the friends that everyone would want to have. I am lucky to have you.

As much as the thesis has been academic and work-related, it has crossed the border (sometimes too much) and has affected my personal life. They have been years of a lot of learning, of very positive moments but also very hard ones. There have been people who have always had a place for me, a space in which to express myself, a time to have fun and a few days to travel. Mikel 'Goiko"s advice has helped me to put my feet on the ground and to relativise in moments of saturation. Mikel's company has given me the strength to go on, inspired me and saved me from several holes (not to mention his direct contribution to the thesis). Iñaki's laughs and anecdotes have helped me to escape from negative thoughts and enjoy the moment. You three and all the people who have come into my life because of you form one of the strongest pillars I have. The chosen family. And continuing with the chosen family, I would like to expressly thank Borja. You arrived at a delicate moment 'my last year of my thesis' and even though you knew that the pool was not completely full, you jumped in. I thank you for that, for not separating from me and for always accompanying me.

Anyone who knows me knows that my brothers, Ander and Beñat, are among the most important things I have. They are part of who I am, of my personality and therefore part of achievements like this. In any case, this new generation of the family - my brothers, Amaia and Borja and myself - would not be where it is without the strongest base it has: Aly and Mary. From them I have learned the most important things: generosity, dedication, kindness, sacrifice, work, caring for others, the team, the family. One thing is very clear to me: this is the result of your support, the result of your sacrifice, the result of your work and the result of absolutely everything you have given us: the title will be mine, but the merit is yours.

AGRADECIMIENTOS

Quisiera empezar agradeciendo todo el soporte técnico y emocional que me han dado por parte de la Universidad del País Vasco y por parte de Ikerlan. Gracias a todas las horas de trabajo frente a una pizarra y todas las llamadas de Montse, Unai y Paqui, este trabajo llega a buen puerto. Gracias por guiarme y acompañarme en este proceso de aprendizaje, cogerme de la mano y no soltarme en ningún momento. También me gustaría mencionar a Franco Raimondi y a Alexander Bolotov por haberme acogido en Londres y haberme dedicado su tiempo. Además, me gustaría agradecer a Alexander su seguimiento y su feedback continuo y por insistir en la palabra formulae como plural de fórmula (queda más elegante).

Durante este proceso, no me puedo olvidar de mis compañeros de Ikerlan y de la universidad. Gente como Unai, Maider, Mikel, Jorge, Lorea, Aitor, Itziar, Ana, Goiuri, Javi, Iñigo, Alex II (hermano de Bea), Laura, Xabi y muchos más -me dejaré a muchísima gente-, han hecho que mi día a día haya sido más llevadero. Gracias por contribuir al buen ambiente de trabajo que en muchos casos ha terminado en amistad, en viajes guays y en cafés maravillosos. Gracias por ayudarme cuando he pedido ayuda y sobre todo cuando me habéis tendido la mano sin yo pedirla.

Y cómo hablar de buen ambiente y de amistad sin mencionar a Ariane, Aritz, Ana y Eneko. Hemos pasado muchas horas juntos y me habéis alegrado en los días tristes, entretenido en los periodos más aburridos, escuchado cuando he necesitado hablar y ayudado en los momentos que lo he necesitado. Sois los trabajadores que toda empresa debería querer y más importante los amigos que todo el mundo querría tener. Soy afortunado de teneros.

Por mucho que la tesis haya sido algo académico y laboral, ha cruzado la frontera (a veces demasiado) y ha afectado a mi vida personal. Han sido años de mucho aprendizaje, de momentos muy positivos pero también muy duros. Ha habido gente que siempre ha tenido un hueco para mí, un espacio en el que expresarme, un rato para divertirse y unos días para viajar. Los consejos de Mikel 'Goiko' me han ayudado a poner los pies en la tierra y a relativizar en momentos de saturación. La compañía de Mikel me ha dado fuerzas a seguir, me ha inspirado y me ha salvado de varios agujeros (por no hablar de su aportación directa en la tesis). Las risas y anécdotas de Iñaki me han ayudado a evadirme de los pensamientos negativos a disfrutar del momento. Vosotros tres y toda la gente cercana que ha llegado a mi vida a raiz de vosotros formáis uno de los pilares más fuertes que tengo. La familia elegida. Y siguiendo con la familia elegida, quisiera dar las gracias expresamente a Borja. Llegaste en un momento delicado 'mi último año de tesis' y aún sabiendo que la piscina no estaba llena del todo te lanzaste a ella. Te doy las gracias por ello, por no separarte de mi lado y por acompañarme siempre.

Cualquiera que me conozca sabe que mis hermanos, Ander y Beñat, son de las cosas más importantes que tengo. Forman parte de quién soy, de mi personalidad y por lo tanto partícipes de logros como éste. De todas formas, esta nueva generación de la familia - mis hermanos, Amaia y Borja y yo- no estaría donde está sin la base más fuerte que tiene: Aly y Mary. De ellos he aprendido las cosas más importantes: la generosidad, la entrega, la bondad, el sacrificio, el trabajo, el cuidar a los demás, el equipo, la familia. Hay algo que tengo muy claro: este es el resultado de vuestro apoyo, el resultado de vuestro sacrificio, el resultado de vuestro trabajo y el resultado de absolutamente todo lo que nos habéis dado: el título será mío, pero el mérito es vuestro.

ABSTRACT

We introduce a context-based tableau method to solve the problem of satisfiability and model checking for temporal logic. In particular, we apply the method to the propositional linear-time temporal logic (PLTL), the computation tree logic (CTL) and to one of its extensions (ECTL). The main feature of our method is that it does not require auxiliary constructions or extra-logic rules, which allow us to maintain the classical duality between tableaux and sequent calculi. Context-based tableaux work as follows: for any input formula, φ, a closed tableau represents a formal sequent proof that certifies the unsatisfiability of φ, whereas an open tableau provides at least one model that certifies the satisfiability of φ. Therefore, in this framework satisfiability and model-checking tests can be performed and complemented by two types of certificates: proofs and models.

In this thesis, we specialise the general method when apply to PLTL (symbolic) model checking. Concretely, we propose the use of a SAT solver to deal with those formulae that specify transitions systems. In this scenario, we also explore the use of the interactive theorem prover, called Isabelle, to perform two certification-related tasks. The first is to certify the method itself. As an example of this, an automated proof that our method is sound (performed in Isabelle) is presented. The second is the possibility offered by Isabelle to re-verify the certificates produced by the tableaux. We provide an example where the output of a closed tableau is a refutational proof with a syntax that Isabelle understands and then, invoking Isabelle the proof can be replicated.

The proposed specialization in addressing PLTL model checking applies immediately to PLTL satisfiability, since the latter problem can be reduced to the problem of PLTL model checking. As a result, the same mechanism is suitable for certifying both PLTL model checking and PLTL satisfiability. However, in the context of branching-time temporal logics, the satisfiability problem cannot be reduced to the model-checking problem. Hence, model-checking algorithms for these logics cannot be adapted for testing satisfiability.

In this dissertation, we study in depth the satisfiability for (CTL) and (ECTL) and propose two context-based tableau methods to solve both problems. The proof of the correctness (soundness, completeness and termination) of the methods are provided. We define algorithms for obtaining tableaux which produce models and formal proofs depending on whether the input formula is satisfiable or not. Our proposal is again suitable for certifying the model checking for branching-time temporal logics. This is due to the fact that model checking always reduces to satisfiability.

Finally, we describe in detail the algorithm for CTL, present a real implementation, and provide experimental results. The prototype has been done using the Dafny language, which is a verification-ready programming language that constantly flags any

errors and gives the go-ahead when the code matches its specification. Dafny can compile the final code to C#, Java, JavaScript or Go. We have generated Java code, but also and more importantly, Dafny has allowed us to prove critical properties that give confidence in the validity and accuracy of the prototype.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In this thesis, we apply the approach to deduction methods for Propositional Linear-time Temporal Logic (PLTL), introduced in [40, 41], to the decision problems of model checking and satisfiability in various temporal logics. The proposal presented in [40, 41] was based on an inductive definition of eventualities that was different from the usual one. There, dual systems of tableaux and sequents for PLTL were presented. Here, we take up these systems and adapt them to both PLTL and branching-time temporal logics with two goals in mind. The first is to make these systems able to provide certificates. The second is to implement them so that they can compete with current automated solvers.

Temporal logics are formal systems for reasoning about time. Due to the fact that time has an important role in both hardware and software behaviour, temporal logics have found extensive application in computer science. A major distinction between temporal logics is whether they see time as linear or branching. This is reflected in the classes of structures that interpret formulae in these logics: linear orderings or trees. In PLTL, all instants are linearly ordered from past to future and there is only one possible future. In branching-time logics, the future is not determined and any given instant may have several distinct immediate successors.

Model checking and satisfiability are two of the most important decision problems in logic. Many questions in several fields of computer science, can be solved by encoding them as instances of these two problems. In general, given a logic L, a class of structures C over which L is interpreted, and a formula $\varphi \in L$, the model-checking problem asks whether a given structure $M \in C$ makes the formula φ true, while the satisfiability problem decides whether such a structure exists. The first problem is, then, a verification problem, where a proposed solution, namely the structure M, is checked for correctness w.r.t. the formula φ. On the other hand, the satisfiability corresponds to a solution problem, where a correct solution, some structure M satisfying φ, has to be founded.

One measure to increase confidence in automated solvers, particularly those dedicated to solving model-checking and satisfiability problems is to make them output a certificate. This certificate is then used to verify the answer of the solver using a different mechanism. In the case of satisfiability, most solvers, given a formula as input that is satisfiable, output a structure that is a model of the input formula. Such structure is an easily checkable certificate. Certificates for unsatisfiability are usually more complicated: the solver output must be a deductive proof showing that any structure is not a model of the input formula. The case of model-checking is similar. Automated model checkers determine whether a given structure is a model of a given formula. They provide a counterexample as a certificate when the structure does not satisfy the formula. However, no such certificate is usually produced if the structure is a model of the

formula. Again, a deductive proof is needed to show this fact.

After outlining the most significant concepts of this thesis, we present some important elements that will be used and explained more deeply throughout this dissertation.

1.1 SAT Solvers and SMT Solvers

Propositional satisfiability (SAT) is the problem of determining, for a formula in the propositional calculus, whether there exists a satisfying assignment for its propositional variables.

SAT solvers are nowadays very efficient tools for deciding SAT (see [76] for a survey). In the positive case, they provide the models of the formula. The numerous applications of SAT solvers have fueled computing research with more efficient and scalable methods. **Satisfiability Module Theories (SMT)** generalizes pure propositional satisfiability (SAT) adding other first-order theories. The SMT framework is extensible, meaning that new theories can be defined and added to deal with different sort of problems. Further information about SMT can be found in [57, 80]. An SMT-solver is an automatic software deciding tool that checks the satisfiability of a formula expressed in a combination of these first-order theories. More specifically, given a (quantifier-free) formula φ and a theory T (a set of sentences), they check whether there exists a model of T that satisfies φ. Most SMT-solvers also give the model (if there exists) that satisfies φ. The power of SMT-solvers comes from their ability to automatically reason about arithmetic, boolean operators, arrays, matrixes, digital circuits, character strings, software data structures, such as lists, trees, etc. There are many available SMT-solvers for the main operative systems, such as, ABSolver [11], Yices [31], Z3 [30], OpenSMT [17], etc.

Over the last years, the increase in power of modern SAT/SMT solvers has been so remarkable that they have become the key enabling technology for symbolic model checking tools. Indeed, the reimplementation of the model checker SMV, called NuSMV [22], integrates SAT solving technology (in combination with Ordered Binary Decision Diagrams - OBDDs) since the second version (NuSMV2 [21]). The model checker NuSMV is open source and is released by IRST in Trento, Italy.

Throughout this thesis, we use a SAT solver instead of an SMT solver, because the considered languages do not include more than propositional variables, i.e. we have not yet considered other elements such as integers, arrays, etc. Nevertheless, adding them would be straightforward by using an SMT solver instead of a SAT solver.

1.2 Tableau Techniques for Temporal Logics

Temporal logic is nowadays essential for the specification and verification of concurrent and reactive systems. There are two types of temporal logics in relation to the treatment of the future at a given instant of time. First type, when each instant of time has a unique possible future we have timelines as linear sequences of this instant (or state). Thus, linear temporal logics extend classical propositional logic by future time temporal operators such as $\circ$ - 'at the next instant of time', $\Diamond$ - 'eventually in the future', $\Box$ - 'always in the future' $\mathcal{U}$ - 'from now until', and $\mathcal{R}$ - 'releases'. Second type, when each instant of time is allowed to have several possible futures (or paths) we have branching-time

temporal logics. The language of branching-time temporal logics extends the language of the linear ones with paths quantifiers A - 'for all paths' quantifier, and E - 'there exists a path' quantifier. Thus, in linear temporal logics, the underlying models of time are discrete, linear sequences of states, finite in the past and infinite in the future. For branching-time temporal logics the underlying models of time are trees (or computation trees) where each branch is a discrete, linear sequence of states, finite in the past and infinite in the future. Note that the formulation of temporal logics with only future-time operators is based on the remarkable Gabbay's *separation result* [38, 39].

The method of semantic tableaux, invented in the 1950's by Beth and Hintikka and later perfected by Smullyan [82] and Fitting [37], is nowadays well established in the field of automated deduction. It brings together the proof-theoretical and the semantic approaches to the presentation of a logical system as a set of inference rules. A tableau is a tree-like structure designed to make semantic reasoning fully systematic to decide whether a set of formulae is satisfiable or not. The construction of a tableau is guided by a set of rules that decompose formulae according to the semantics of its logical operators.

Temporal tableaux, first introduced in [83], tackling the problem that formulae must be analysed in a infinite sequences of states, introduce a mechanism which controls repeated appearances of formulae and identify periodic situations in finite time. Later, we explain the two options to cope with this problem. For the survey of the tableau method for temporal logic we refer an interested reader to [45].

There exists many tableau techniques for a rich variety of temporal logics: PLTL; *Computation Tree Logic* (CTL); (CTL*) which generalizes PLTL and CTL, etc. (an excellent survey can be found in [45]). The most difficult task in tableaux methods for temporal logics is to check out the fulfilment of eventualities, i.e., formulae of the form 'eventually φ' or 'φ until ψ'. Traditional tableaux methods require two phases to perform this test. In the first phase, they construct a graph of states. This graph represents all possible pre-models. In the second phase, for each state s that contains some eventuality, φ, a graph-theoretic algorithm should look for a state, reachable from s, that satisfies φ. Moreover, nodes with negative test should be pruned. The two-pass tableau methods fail to carry out the classical correspondence between tableaux and sequents that associates a sequent-proof to any closed tableau.

To avoid the second phase and, hence, to keep generating (sequent) proofs from tableaux, in [40, 41], **dual systems of tableaux and sequents** for PLTL, were presented. Every system defined in [40, 41] was proved to be sound and complete. In particular, [41] contains a detailed proof showing that the tableau system is a decision method for PLTL, i.e., it is sound, refutationally complete, and terminating. The termination property is achieved on the basis of any fair selection strategy. The tableau method in [41] makes use of the so-called *context* of an eventuality to force its fulfilment. The context of an eventuality is simply the set of formulae that 'accompanies' the eventuality in the label of a node in the tableau. From now on, we call this tableau method **one-pass context-based tableaux** or simply context-based tableaux.

An extensive search on tableau techniques for CTL has not shown a great variety of systems. For example, [12] presents a two-pass tableau, where in the first pass the tableau rules are applied creating a cyclic graph. In the second pass, 'bad loops' are pruned (where a 'bad loop' is a loop containing some eventuality that is not fulfilled along it). In [1, 46] the authors introduce a single-pass tableaux decision procedure for

CTL. It is based on Schwendimann's one-pass procedure for PLTL [79]. This tableau method uses an additional mechanism for collecting information on the set of formulae in the nodes, and passing it, to subsequent nodes along branches. The information on previously generated nodes helps detecting 'bad loops' without constructing the whole graph.

1.3 Model Checking

Although **Model Checking** is a more general concept, it is commonly understood [24, 77, 28, 10, 27] as an algorithmic method for determining whether a complex hardware or software system satisfies a given property. Many important properties to be verified reflect the system's dynamics and are expressed in some temporal logic. Thus, a model checker receives as input a system (usually called the transition system) and a temporal formula (the property). If the property does not hold, the checker returns a counterexample: a trace/model of the transition system that does not satisfy the property. This counterexample acts as a 'certificate' of the failure and its role is to help the user to identify the source of the problem which could be in the transition system design, in the property, and even in the model checker.

Model checking using formulae to represent the transition system is called **Symbolic Model Checking** [19, 71]. The term emphasizes that the specification of the transition system is represented symbolically, namely, by a formula. The earliest symbolic model checker SMV [71] applies Ordered Binary Decision Diagrams (OBDDs) [18, 19] as a canonical form for boolean formulae that is very compact because of variable-sharing. Very efficient algorithms have been developed for manipulating OBDDs. This made possible to verify transition systems with an extremely large number of states –some orders of magnitude larger than could be handled by the explicit-state model checkers (e.g. SPIN [50]). In symbolic model checking, two-pass tableaux are extensively used to construct different kinds of state machines (graphs) for representing transition systems and properties (cf.e.g. [28]). These graphs are indeed the result of the first pass of a two-pass tableau method and they are represented using OBDDs. However, for larger transition systems, the OBDDs generated during model checking become too large, and the generation of a variable ordering that results in small OBDDs is often time and space consuming or needs manual intervention. For many examples no efficient variable ordering exists. In other words, the bottleneck of these methods is the amount of resources that are required for storing and manipulating OBDDs.

1.4 Certified Model Checking

For years, model checkers do not produce a **certificate** when the transition system meets the property. Hence, for positive answers, the model checkers do not provide any hint of the truth of the property nor do they help the user find a problem when the property is not expected to be true. In this context, the sequential calculus associated with context-based tableaux allow us to generate formal proofs certifying that a particular transition system satisfies a property.

On the other hand, given the high complexity of the implementation of model checkers, a natural question that arises is: who checks the checker? An alternative consists

on proving, only once, the correction of the underlying algorithm of the model checker. For this task, interactive proof assistants such as Coq or Isabelle [75] are good tools. They allow us to certify a model checker and even to obtain an executable program by the refinement of some extraction mechanism. For instance Amjad [7] described how to code BDD-based symbolic model-checking algorithms into an automatic theorem prover. More recently, Esparza et al. [36] have verified an automata-based model checker with Isabelle theorem prover.

As just mentioned, another option is to return a proof evidence for every positive answer [74, 72] that could help the user –maybe using an auxiliary theorem prover– to trust the model checker or not. In this scenario, symbolic model checking is especially suitable, because the transition systems are specified by formulae. In this way, the model-checking problem can be seen as a particular case of satisfiability, since deciding whether a transition system M meets a property φ is logically equivalent to deciding whether $(M \wedge \neg\varphi)$ is unsatisfiable. And vice versa, the fact that M does not meet a property φ is logically equivalent to the fact that $(M \wedge \neg\varphi)$ is satisfiable.

1.5 Contribution

This dissertation contributes in several aspects related to certified model checking and certified satisfiability of temporal formulae. The logics worked on the thesis are PLTL, the Computation Tree Logic (CTL), and the Extended Computation Tree Logic (ECTL), which enriches CTL with simple fairness formulae. CTL and ECTL were introduced in [33] and [35] respectively.

1.5.1 Certified Model Checking for PLTL

In this thesis a Certified Model Checker (CMC) for PLTL is proposed based on dual systems of context-based tableaux and sequent calculus, originally introduced in [40, 41]. It produces certified proofs - formal proofs in the sequent calculus, and counterexamples - open branches in the tableau. One of the main advantages of this approach is that the same reasoning mechanism applies for both certificates - formal proofs and counterexamples.

The most remarkable difference of our proposal (regarding [66, 67]) is that CMC generates deductive proofs thanks to the use of the context-based tableau along with its dual sequent calculus. It should be noted that the context-based tableau is particularly well suited for dealing with the specifications of transition systems ('always'-formulae). The context plays the role 'of forcing eventualities to be fulfilled as soon as possible' and acts as a semantic constraint that prevents the generation of many states that are produced in the two-pass approach.

In building the proof, Isabelle [75] is invoked to verify the construction of the tableaux. This external validation is carried out by a sequent calculus TTC (Tait-style Temporal Calculus) which is formalized in Isabelle (see the file `TTC_Calculus.thy` in `http://github.com/alexlesaka/OnePassTableau`). This calculus is dual to a variant of the context-based tableau method that works with formulae in negation normal form. The formalization has been codified as an Isabelle theory and we also provide (in Isabelle) the soundness proof of such theory (see the file `TTC_Soundness.thy` in

http://github.com/alexlesaka/OnePassTableau).

The author's experience implementing a previous prototype of the context-based tableau method [1] revealed that a large proportion of the computational effort is spent doing classical propositional reasoning (e.g. analysing boolean combinations of literals.), for which SAT/SMT-solvers are very efficient tools.

The idea of encoding transition systems into propositional formulae were first proposed in [54] for AI planning problems. The authors show that SAT algorithms scale much better on the SAT-encodings than planing algorithms on the original graph formulation. Following this success, SAT solvers have been also used in *Bounded Model Checking* (BMC) [13]. In both frameworks, a propositional formula is used to encode the input problem. Planning problems deal with finite traces along a finite graph, hence the encoding is complete w.r.t. the original problem. However, in model checking, traces within the transition systems are, in general, infinite. SAT-based BMC works as follows: the propositional encoding expresses that there exists a length-k trace (along the transition system) that does not satisfy a given property. The BMC increases k until either the SAT's answer is a counterexample or k reaches a certain bound. The original SAT-based BMC algorithm [13], although complete for finite traces, is limited in practice to falsification. Many additional strategies have been introduced to make BCM complete, see [14] for a good survey.

There is also a large amount of work, starting with [78], on using SAT solvers for improving the satisfiability test of the full PLTL. Recent papers [66, 67] use SAT solvers to seek for a model of an input formula. This model is essentially a graph/automaton produced by the first pass of a two-pass tableau method, which should be followed for testing the fulfilment of eventualities. SAT solvers are called for the generation of all (different) successors of every state in the graph. The authors use well-known temporal equivalences (like $p\mathcal{U}q \equiv (q \vee (p \wedge \circ(p\mathcal{U}q)))$ to compute the successors of the given state. They propose a method that utilises the renaming of subformulae containing temporal operators (such as $p\mathcal{U}q$) by fresh propositional variables and uses the SAT solving to calculate different successor states of each state in the transition system. This prevents the repeated generation of 'propositionally equivalent' states. The relevant heuristics for pruning the search-space and on-the-fly mechanism for testing eventuality fulfilment are introduced in the implementation.

In our setting, we delegate to a SAT solver the non-temporal reasoning part of the tableau construction. The main reason is that context-based tableaux are much simpler to construct when their inputs are formulae that specify a transition system (along with a temporal property to test). Only a very restricted subset of temporal formulae can be used to design such transition systems. Coding them into classical propositional formulae allows us to leave the temporal reasoning only in the scope of the properties to be checked and not in the systems to be tested. In contrast to proposals that translate the whole model checking problem on one SAT-solver query (e.g. [72]), our method combines SAT solving with tableau-based temporal reasoning. As a consequence, the certified proof that is generated shall contain the trace of the temporal reasoning, which provides hints that are close to the tested transition system. Moreover, context-based tableau method (helped, for efficiency, by a SAT solver) is complete for deciding (unbounded)

[1] The prototype is available in http://www.sc.ehu.es/jiwlucap/TTM.html

model-checking problems and works on infinite traces.

To summarize, first we adapt the original dual method for formulae in negation normal form. Second, we present an optimized version of this adaptation by using a SAT solver, which performs the CMC for PLTL. Finally, we use Isabelle to prove the soundness of our method.

1.5.2 *Certified Satisfiability Checking for* CTL *and* ECTL

The context-based tableau method [41] was extended to a concurrent constraint logic in [29], and also to ECTL$^{\#}$ - a branching-time sublogic of CTL* in [15]. This thesis continues the development of such technique for other temporal logics, this time for the branching-time temporal logics CTL and ECTL. These two logics are simpler than ECTL$^{\#}$, but the application of the general method for ECTL$^{\#}$ to the simpler cases of CTL and ECTL would become too 'non-intuitive' due to the complexity of the rules that are necessary for this richer logic. It is worth noting that the distinguished (and unavoidable) feature of the context-based technique for ECTL$^{\#}$ is the utilisation of two types of contexts, unlike in the case of PLTL. These types of contexts are the so-called 'outer' context, which is a collection of state formulae, and the so called 'inner' context, a collection of path formulae. Therefore, the development of a simpler context-based tableau method for CTL and ECTL is an important task. The method we present for these simpler logics, similarly to PLTL, only needs the 'outer' context.

Obviously, the classical duality between tableaux and sequent calculi is maintained for CTL and ECTL. This allows us to return deductive proofs in a sequent calculus when the CTL or ECTL formula to be checked is unsatisfiable. We include the soundness and completeness proofs of both methods: the context-based tableaux and its dual sequent calculus for CTL and ECTL.

A remarkable observation about PLTL is that the satisfiability checking can be reduced to model checking. Consider a formula φ over a set, $Prop$, of propositional variables. If a model M is universal, that is, it contains all possible traces over $Prop$, then φ is satisfiable precisely when the model M does not satisfy $\neg\varphi$. Thus, it is easy to add a satisfiability-checking feature to PLTL model-checking tools. Both the satisfiability and the model-checking problems are PSPACE-complete [81]. However, the CTL satisfiability problem cannot be reduced to the CTL model-checking problem. In particular, a model checking algorithm for CTL (for example the one implemented in NuSMV [19]) cannot be adapted for testing CTL satisfiability. This contrasts with what we have pointed out above: the model checking problem always reduces to the satisfiability problem, but not the other way around. In fact, the model checking problem for CTL is known to be P-complete [26], while the satisfiability problem for CTL is EXPTIME-complete [34].

We have applied all the knowledge in the development of the PLTL certified model checker in the creation of the dual satisfiability method for CTL and ECTL that is able to return certificates. Our method is already optimized for 'always'-formulae (used for the transition systems in the context of model checking). Thus, the presented technique is also able to provide certificates for model checking in CTL and ECTL.

We think this is the main contribution of the thesis. We develop an intuitive tableau method that serves as a decision procedure for CTL and ECTL satisfiability. The method

provides certificates and can also be used in model checking for CTL and ECTL. We prove the effectiveness of the approach that now covers both linear-time and a range of branching-time temporal logics. Moreover, the results give us formalisms, which are well suited for automation and are amenable to implementation.

1.5.3 Implementation and Experimental Results

This dissertation presents a Java prototype called MomoCTL that performs certified satisfiability checking for CTL. The prototype is available in `http://github.com/alexlesaka/MomoCTL`. Most of the code is automatically generated from the language Dafny [60], which is a program verifier of functional correctness. Thanks to Dafny we are able to verify crucial properties of MomoCTL.

We test our prototype on the collection of benchmarks borrowed from `http://users.cecs.anu.edu.au/~rpg/CTLComparisonBenchmarks/`, which was created for the comparison of various CTL provers in [48]. Also, we report on our performance results and compare them with the single-pass tableau for CTL ([1]) known as the Gore's tableaux. MomoCTL returns either a tree model when the input is a satisfiable CTL formula or a deductive proof in the dual sequent calculus. This is the main feature of our prototype compared to previous CTL provers. Moreover, since model checking reduces to satisfiability checking, MomoCTL is a model checker for CTL capable of providing both types of certificates.

1.6 Outline of the Thesis

This thesis is organized in five chapters as follows:

- In Chapter 2, we introduce the PLTL logic and point out that our tableaux and sequent proofs work only with temporal formulae in Negation Normal Form for efficiency reasons. This differentiates us from the initial proposal of [40, 41]. We explain the adaptation of the dual method for formulae in Negation Normal Form. We provide some examples of tableaux and also proofs in the associated sequent calculus. The chapter ends with the formalization of the sequent calculus in Isabelle and the interactive proof (again done in Isabelle) that the calculus is sound. This last part of the chapter aims to show the potential of proof assistants like Isabelle or Coq. The contents of this chapter is based on [6, 2, 3].

- In Chapter 3, we present a new PLTL Certified Model Checking procedure. We explain the optimization with SAT solvers of the original dual method of context-based tableaux and sequents. Finally, we present a prototype that is able to generate deductive proofs in Isabelle in case the property is satisfied. The contents of this chapter is strongly based on [6, 3]

- In Chapter 4, we focus on CTL and ECTL logics and introduce a new context-based tableau method with its associated sequent calculus. The method decides the satisfiability of CTL and ECTL formulae and provides certificates. We include illustrative examples and prove the soundness and completeness of the method. We also justify that our proposal can solve the model-checking problem for CTL and ECTL with the additional benefit of producing certificates. The contents of Chapter 4 is strongly based is [4, 5].

- In Chapter 5, we introduce the Dafny language. Next, we present a Dafny implementation of the method explained in Chapter 4 for CTL. Finally, we compare the performance of our implementation with Gore's CTL-prover [1, 47]. The contents of Chapter 5 is strongly based is [4, 5].
- In Chapter 6, we summarise the contributions of the thesis and define future work.

2. PROPOSITIONAL LINEAR TEMPORAL LOGIC

This chapter is divided into three main sections: Section 2.1 introduces the *Propositional Linear Temporal Logic* PLTL. In Section 2.2, we propose the adaptation of the dual systems of context-based tableaux and sequents for PLTL (introduced in [39]), to Negated Normal Form formulae. Finally, in Section 2.3 the formalization of the sequent calculus for PLTL and the proof that the calculus is sound, both done in Isabelle, are presented.

2.1 The Logic PLTL

The main goal of this section is to introduce how PLTL extends the syntax of classical propositional logic by allowing the use of temporal operators.

Definition 1 (Syntax of PLTL)**.** *Let* Prop *be a fixed set of propositional variables and let* $p \in$ Prop*. We define a* PLTL *formula* φ *over* Prop *as follows:*

$$\varphi :: \mathbf{T} \mid \mathbf{F} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \circ\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid \Diamond\varphi \mid \Box\varphi \mid \varphi_1 \mathcal{R} \varphi_2.$$

Note that the set $\{\neg, \wedge, \circ$ and $\mathcal{U}\}$ is known to be sufficient to represent all other connectives: $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\Diamond\varphi \equiv \mathbf{T}\mathcal{U}\varphi$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$, $\varphi\mathcal{R}\psi \equiv \neg(\neg\varphi\mathcal{U}\neg\psi)$. PLTL formulae will be named in lower-case Greek letters like φ or ψ. Formulae of the form $\varphi\mathcal{U}\psi$ and $\Diamond\varphi$ are called *eventualities*. Those of the form $\circ\varphi$ are *next formulae*. Formulae of the type $\Box\varphi$ are called *always formulae*.

The upper-case Greek letters like Δ or Σ denote finite set of formulae. Given a set of formulae $\Sigma = \{\sigma_1, ..., \sigma_n\}$ we use $\neg\Sigma$ to denote the formula $\neg(\sigma_1 \wedge ... \wedge \sigma_n)$. When Σ is empty, $\neg\Sigma$ is the constant $\mathbf{F}$.

Formulae are interpreted in the states of PLTL-structures.

Definition 2 (PLTL-structure)**.** *A* PLTL*-structure* M *is a pair* (S_M, V_M) *where* S_M *is a enumerable sequence of states* $s_0, s_1, s_2, \ldots$ *and* $V_M : S_M \rightarrow 2^{\text{Prop}}$ *maps each state* $s_i \in S_M$ *into a subset of* Prop*.*

Intuitively, V_M specifies which propositional variables are necessarily true in each state. Definition 3 introduces how a PLTL formula is evaluated over a PLTL-structure.

Definition 3. *The truth of a formula* φ *in the state* $s_j \in S_M$ *of a* PLTL*-structure* M*, which is denoted by* $\langle M, s_j\rangle \models \varphi$*, is inductively defined as follows:*

$\langle M, s_j\rangle \not\models \mathbf{F}$

$\langle M, s_j\rangle \models p$ *if and only if* $p \in V_M(s_j)$ *for any* $p \in$ *Prop*

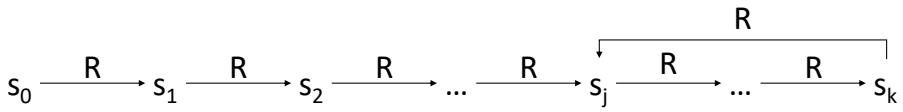


Figure 2.1: Cyclic sequence of states.

$\langle M, s_j \rangle \models \neg\varphi$ *if and only if* $\langle M, s_j \rangle \not\models \varphi$

$\langle M, s_j \rangle \models \varphi \wedge \psi$ *if and only if* $\langle M, s_j \rangle \models \varphi$ *and* $\langle M, s_j \rangle \models \psi$

$\langle M, s_j \rangle \models \circ\varphi$ *if and only if* $\langle M, s_{j+1} \rangle \models \varphi$

$\langle M, s_j \rangle \models \varphi \mathcal{U} \psi$ *if and only if* $\langle M, s_k \rangle \models \psi$ *for some* $k \geq j$ *and* $\langle M, i \rangle \models \varphi$ *for every* $j \leq i < k$

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

$\langle M, s_j \rangle \models \mathbf{T}$

$\langle M, s_j \rangle \models \varphi \vee \psi$ *if and only if* $\langle M, s_j \rangle \models \varphi$ *or* $\langle M, s_j \rangle \models \psi$

$\langle M, s_j \rangle \models \Diamond\varphi$ *if and only if* $\langle M, s_k \rangle \models \varphi$ *for some* $k \geq j$

$\langle M, s_j \rangle \models \Box\varphi$ *if and only if* $\langle M, s_k \rangle \models \varphi$ *for every* $k \geq j$

$\langle M, s_j \rangle \models \varphi \mathcal{R} \psi$ *if and only if for every* $k \geq j$*, either* $\langle M, s_k \rangle \models \psi$ *or there exists* i *such that* $j \leq i < k$ *and* $\langle M, s_i \rangle \models \varphi$

In addition, for any set Σ *of formulae,* $\langle M, s_j \rangle \models \Sigma$ *if and only if* $\langle M, s_j \rangle \models \sigma$*, for all* $\sigma \in \Sigma$*.*

Definition 4 (Satisfiability)**.** *For a set of* PLTL *formulae* Σ*, the set of its models,* $\mathsf{Mod}(\Sigma)$*, is formed by all pairs* $\langle M, s \rangle$ *such that* $\langle M, s \rangle \models \Sigma$*.* Σ *is satisfiable* ($\mathsf{Sat}(\Sigma)$) *if* $\mathsf{Mod}(\Sigma) \neq \emptyset$*, otherwise* Σ *is unsatisfiable* ($\mathsf{UnSat}(\Sigma)$)*.*

Definition 5 (Logical equivalence)**.** *Two* PLTL *formulae* φ *and* ψ *are logically equivalent, denoted as* $\varphi \equiv \psi$*, when* $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\psi)$*. Two sets of* PLTL *formulae* Σ *and* Δ *are logically equivalent when* $\mathsf{Mod}(\Sigma) = \mathsf{Mod}(\Delta)$*.*

Any infinite sequence $s_0, s_1, ..., s_k, ...$ involves an implicit successor relation, namely R, such that $(s_i, s_{i+1}) \in R$ for all $i \in \mathbb{N}$. A finite sequence gives also a corresponding implicit successor relation with a pair for each element except for the last one.

In order to construct models for satisfiable sets of formulae, we use cyclic PLTL-structures that we define in terms of cycling sequences.

Definition 6 (the authors presented a dual method to analyse Path)**.** *Let* π *be a finite sequence of states* $\pi = s_0, s_1, \ldots, s_j$*. Then*

- π *is cyclic if and only if there exists* s_i*,* $0 \leq i \leq j$ *such that* $(s_j, s_i) \in R$*.*

- *The sequence $s_i, \ldots, s_j$ is called the loop of π.*
- *A cyclic path over a cyclic sequence π is the infinite sequence*

$$path(\pi) = s_0, s_1, \ldots, s_{i-1}, \langle s_i, s_{i+1}, \ldots, s_j \rangle^\omega$$

where $\langle s_i, s_{i+1}, \ldots, s_j \rangle^\omega$ denotes the infinite sequence that results by repeating the loop of π infinitely many times.

A PLTL*-structure M is cyclic if its sequence of states is a cyclic path over a cyclic sequence.*

Figure 2.2: PLTL-structure

Example 1 ($M \models \{\Box(\Diamond p \wedge \Diamond \neg p)\}$)**.** *Let φ be the formula $\Box(\Diamond p \wedge \Diamond \neg p)$. Let M be the* PLTL*-structure defined as $V_M(s_0) = \{\ \}, V_M(s_1) = \{\ \}, V_M(s_2) = \{p\}, V_M(s_3) = \{\ \}, V_M(s_{2k}) = V_M(s_2)$ and $V_M(s_{2k+1}) = V_M(s_3)$ for every $k \geq 2$ (graphically in Figure 2.2). It is easy to see that M is a cyclic* PLTL*-structure and $M \models \{\Box(\Diamond p \wedge \Diamond \neg p)\}$. We can represent M as the cyclic path $\{\ \}, \{\ \}, \langle \{p\}, \{\ \} \rangle^w$. From now on, we will use this representation for cyclic* PLTL*-structures.*

Below we present the syntax of PLTL in *negation normal form* (shortly, NNF).

Definition 7 (Syntax of PLTL in NNF)**.** *Let* Prop *be a fixed set of propositional variables, let $p \in$* Prop *and let $Lit ::= \mathbf{F} \mid \mathbf{T} \mid p \mid \neg p$ be the set of literals. The set of* PLTL *formulae in* NNF *over* Prop *is given by the grammar*

$$\varphi :: Lit \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \circ\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid \Diamond\varphi \mid \Box\varphi \mid \varphi_1 \mathcal{R} \varphi_2.$$

Therefore, a PLTL formula φ is in NNF whenever every occurrence of the negation connective $\neg$ is in front of a propositional variable. The following result [68] can be easily established.

Proposition 1. *For any* PLTL *formula φ there exists a* PLTL *formula,* NNF(φ)*, which is in* NNF *such that* Mod(φ) = Mod(NNF(φ))*.*

Proof. By structural induction on the formulae, using the following well known equivalences (e.g. [32]):

$$\begin{array}{cccc} \neg\mathbf{T} \equiv \mathbf{F} & \neg\mathbf{F} \equiv \mathbf{T} & \neg\neg\varphi \equiv \varphi & \neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi \\ \neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi & \neg\circ\varphi \equiv \circ\neg\varphi & \neg\Box\varphi \equiv \Diamond\neg\varphi & \neg\Diamond\varphi \equiv \Box\neg\varphi \\ \neg(\varphi \mathcal{U} \psi) \equiv \neg\varphi \mathcal{R} \neg\psi & \neg(\varphi \mathcal{R} \psi) \equiv \neg\varphi \mathcal{U} \neg\psi & & \end{array}$$

□

Definition 8 (Notation)**.** *Let φ be a* PLTL *formula. The formula* NNF$(\neg\varphi)$ *is denoted as $\sim\varphi$.*

Note that the set of PLTL formulae in NNF is closed under the operation $\sim$, i.e., if φ is in NNF, then $\sim\varphi$ is also in NNF.

Example 2 (Transformation to NNF)**.** *The following are examples of the transformation of* PLTL *formulae to* NNF*:*

- $\sim(\Diamond a \wedge \Box b) = \Box\neg a \vee \Diamond\neg b$
- $\sim\circ(\Box a \wedge (\Box b)\mathcal{U}c) = \circ(\Diamond\neg a \vee (\Diamond\neg b)\mathcal{R}\neg c)$
- $\sim(a\mathcal{U}((\Box\neg b)\mathcal{R}c)) = (\neg a)\mathcal{R}((\Diamond b)\mathcal{U}\neg c)$
- $\sim\circ(((a \vee \Diamond b) \wedge \Box c)\mathcal{U}(\Box b)) = \circ(((\neg a \wedge \Box\neg b) \vee \Diamond\neg c)\mathcal{R}(\Diamond\neg b))$

Tableaux are refutational methods based on symbolic handling of set of formulae for detecting syntactic inconsistencies on them, in the sense of the following definition.

Definition 9 (Syntactically consistent set of formulae)**.** *A set Σ of* PLTL *formulae is syntactically consistent if and only if $\mathbf{F} \notin \Sigma$ and $\{\varphi, \sim\varphi\} \not\subseteq \Sigma$ for any* PLTL *formula φ. Otherwise, Σ is inconsistent.*

2.2 Dual Methods for PLTL: Context-Based Tableaux and Sequent Calculus

The exposition of the Dual System of Tableaux and Sequents for PLTL is structured in two parts. In the first part, Section 2.2.1, the tableau method is presented. In the second part, section 2.2.2, the sequent calculus is introduced.

2.2.1 Systematic Tableaux Construction

In [41], the authors presented a dual method to analyse the satisfiability of a set of PLTL formulae: the TTM Tableaux method (Temporal Tableaux Method) and its dual TTC sequent calculus (Tait-style Temporal Calculus). The completeness, soundness and termination of both methods were given in [41].

In this thesis we present an adaptation of TTM and TTC to work with PLTL formulae in NNF. It is worth noting that the temporal tableaux in TTM are one-pass [41], in the sense that they do not require to check an auxiliary graph of states in order to determine if every eventuality is satisfied. As a consequence, temporal states are represented inside the branches of the tableaux instead of in an auxiliary graph. This fact keeps the formulation of a sequent calculus as dual to the tableau method and allows us to provide certifications for both possibilities of an input formula - its satisfiability and unsatisfiability.

Another feature of TTM is that the tableau construction is based on the concept of a context of an eventuality, which is a set of formulae that accompanies the eventuality in the label of a node of a tableau tree. Our specific tableau rules, which involve context, force the earliest fulfilment of eventualities. For that reason, we call our tableau-method context-based tableaux.

In this section we overview the construction of the context-based tableaux.

Definition 10 (Tableau, Consistent Node, Closed branch)**.** *A tableau for a set of* PLTL *formulae* Σ *is a tree,* T*, where nodes are labelled with sets of formulae, such that the following two conditions hold:*

(a) The root is labelled by the tableau input, Σ*.*

(b) Any other node, m*, is labelled with sets of formulae as the result of the application of one of the expansion rules to the parent node of* m [1]*.*

A node n *of* T *is consistent, if its label is a consistent set of formulae, else* n *is inconsistent. Any sequence of consecutive nodes, whose first element is the root of the tableau, is called a tableau branch. If a branch,* b*, of* T*, contains a syntactically inconsistent node, then* b *is closed else* b *is open. Closed branches are not further expanded, i.e. expansion rules only apply to open branches.*

The generation of new nodes in the tableau is made according to a set of expansion rules: α rules that result in a single node, β rules that cause branching, next-state rule which applies to the whole set of formulae in the label of the node and finally β^+ rules. In the presentation of the rules, φ and ψ represent the subformulae of the formula to which the rule is applied. The set Σ represents the context, set of formulae that accompanies the formula to which the rule applies.

Before presenting the expansion rules, we introduce the PLTL basic modalities in Definition 11. It reflects the restrictions on forming admissible combinations of temporal operators.

Definition 11 (PLTL Basic Modalities)**.**
$\mathsf{M}_{\mathsf{PLTL}} ::= c \mid \circ\mathsf{M} \mid \Box\mathsf{M} \mid \mathsf{M}\mathcal{U}\mathsf{M} \mid \Diamond\mathsf{M} \mid \mathsf{M}\mathcal{R}\mathsf{M}$. *where* c *stands for a purely classical formula (we can consider a purely classical formula as a zero-degree basic modality) and* M *stands for any basic modality of* PLTL*.*

Our α and β rules are based on fixpoint characterisation of its basic modalities (in the equations below ν and μ stand for 'minimal fixpoint' and 'maximal fixpoint' operators, respectively):

$$\Box\varphi = \nu\rho(\varphi \wedge \circ\rho) \qquad \varphi\mathcal{U}\psi = \mu\rho(\psi \vee (\varphi \wedge \circ\rho))$$
$$\Diamond\varphi = \mu\rho(\varphi \vee \circ\rho) \qquad \varphi\mathcal{R}\psi = \mu\rho((\psi \wedge \varphi) \vee (\psi \wedge \circ\rho))$$

This fixpoint characterisation of basic PLTL modalities as maximal or minimal fixpoints give rise to their analytical classification as α- or β-formulae which are associated, in the tableau with α- and β rules: $\Box$ is maximal fixpoint and is classified as α-formulae while $\Diamond$, $\mathcal{U}$ and $\mathcal{R}$ as minimal fixpoints are β-formulae. This is also reflected in the known logical equivalences:

$$\Box\varphi \equiv \varphi \wedge \circ\Box\varphi \qquad \varphi\mathcal{U}\psi \equiv \psi \vee (\varphi \wedge \circ(\varphi\mathcal{U}\psi))$$
$$\Diamond\varphi \equiv \varphi \vee \circ\Diamond\varphi \qquad \varphi\mathcal{R}\psi \equiv (\psi \wedge \varphi) \vee (\psi \wedge \circ(\varphi\mathcal{R}\psi))$$

[1] We explain the expansion rules in the next subsection.

α and β rules

$$(\wedge)\ \frac{\Sigma, \varphi \wedge \psi}{\Sigma, \varphi, \psi} \qquad (\Box)\ \frac{\Sigma, \Box\varphi}{\Sigma, \varphi, \circ\Box\varphi}$$

Figure 2.3: α rules

There are two α rules (Figure 2.3): *And* ($\wedge$) and *Always* ($\Box$). The ($\wedge$) rule applies to $\varphi \wedge \psi$ and separates φ and ψ creating a new node $\{\Sigma,\varphi,\psi\}$. The ($\Box$) rule applies to $\Box\varphi$ formula and unfolds the formula: in the current state φ is true (φ) and from the next state always φ is true ($\circ\Box\varphi$).

$$\begin{array}{cc} \underline{a \wedge b}, \Box c & a, b, \underline{\Box c} \\ \quad\big|\ (\wedge) & \quad\big|\ (\Box) \\ a, b, \Box c & a, b, c, \circ\Box c \end{array}$$

Figure 2.4: Examples of the application of α rules.

In Figure 2.4 there are two examples of the application of each of the rules ($\wedge$) and ($\Box$).

$$(\vee)\ \frac{\Sigma, \varphi \vee \psi}{\Sigma, \varphi \ \mid\ \Sigma, \psi} \qquad (\mathcal{U})\ \frac{\Sigma, \varphi\mathcal{U}\psi}{\Sigma, \psi \ \mid\ \Sigma, \varphi, \circ(\varphi\mathcal{U}\psi)}$$

$$(\Diamond)\ \frac{\Sigma, \Diamond\varphi}{\Sigma, \varphi \ \mid\ \Sigma, \circ\Diamond\varphi} \qquad (\mathcal{R})\ \frac{\Sigma, \varphi\mathcal{R}\psi}{\Sigma, \varphi, \psi \ \mid\ \Sigma, \psi, \circ(\varphi\mathcal{R}\psi)}$$

Figure 2.5: β rules

There are four β rules (Figure 2.5): *Or* ($\vee$), *Until* ($\mathcal{U}$), *Release* ($\mathcal{R}$) and *Eventually* ($\Diamond$). The ($\vee$) rule applies to $\varphi \vee \psi$ and creates two branches where in the first branch φ is true and in the second branch ψ is true. The ($\mathcal{U}$) rule applies to $\varphi\mathcal{U}\psi$ and creates two branches with the two possible cases: in the first one the eventuality is fulfilled, ψ is true; in the second one, the fulfilment of the eventuality is postponed. In that case φ is true and the eventuality is checked again in the next state ($\circ(\varphi\mathcal{U}\psi)$). The ($\mathcal{R}$) rule applies to $\varphi\mathcal{R}\psi$ and creates two branches where in the first one φ and ψ are true in the current state (φ, ψ) and in the second one ψ is true and the release is checked again in the next state ($\circ(\varphi\mathcal{R}\psi)$). The ($\Diamond$) rule applies to $\Diamond\varphi$ and works the same as $\mathtt{T}\mathcal{U}\varphi$: a branch is created when the eventuality is satisfied (φ) and when it is not satisfied, the eventuality is checked in the next state ($\circ\Diamond\varphi$).

In Figure 2.6 there are four examples illustrating the application of the rules ($\vee$), ($\mathcal{U}$), ($\mathcal{R}$) and ($\Diamond$).

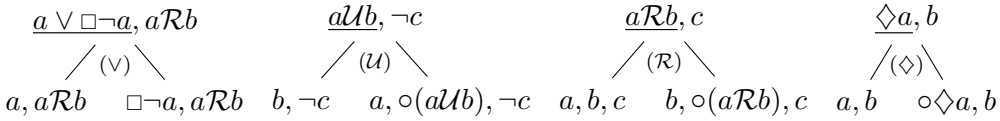


Figure 2.6: Examples of application of β rules.

Next-state rule

The next state rule is the only rule that applies to the complete set of formulae. It jumps to the next state. The formulae that do not have a next operator are not passed to the next state.

Definition 12 (Elementary formulae)**.** *Literals and formulae of the form $\circ\varphi$ are called elementary, the remaining formulae are called non-elementary. In addition, sets of elementary formulae are also called elementary.*

$$(\circ)\ \frac{\Sigma_1, \circ(\Sigma_2)}{\Sigma_2}$$

where Σ_1 is formed by literals and Σ_2 is formed by next formulae.

Figure 2.7: Next-state rule

The $(\circ)$ rule (Figure 2.7) applies when the set of formulae is elementary (Definition 12). The next formulae become the current state formulae.

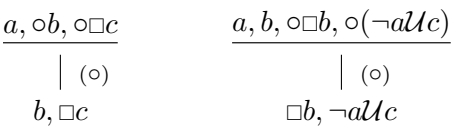


Figure 2.8: Examples of the application of next-state rule.

In Figure 2.8 there are two examples illustrating the application of the rule $(\circ)$.

β^+ rules

In the set of expansion rules, apart from the standard α and β rules, we also have β^+ rules that are characteristic (and crucial!) for our construction. They were introduced in [40, 41].

$$(\mathcal{U}^+)\quad \frac{\Sigma, \varphi\mathcal{U}\psi}{\Sigma, \psi \;\mid\; \Sigma, \varphi, \circ((\varphi \wedge \sim\Sigma')\mathcal{U}\psi)} \qquad (\Diamond^+)\quad \frac{\Sigma, \Diamond\varphi}{\Sigma, \varphi \;\mid\; \Sigma, \circ((\sim\Sigma')\mathcal{U}\psi)}$$

where $\sim\Sigma' = \bigvee_{\sigma\in\Sigma\setminus\mathcal{P}} \sim\sigma$. If Σ is empty, $\sim\Sigma' = \mathbf{F}$.

Set $\mathcal{P}$ includes the persistent formulae, which are those of the form $\circ^i\Box\delta$ with $i \geq 0$.

Figure 2.9: β^+ rules.

There are two β^+ rules (Figure 2.9): $(\mathcal{U}^+)$ and $(\Diamond^+)$. Thank to these rules the auxiliary graph created in two-pass tableaux method is unnecessary for context-based tableaux. $(\mathcal{U}^+)$ and $(\Diamond^+)$ use the *context*, Σ. By the context of an eventuality, we understand the collection of all other formulae within the label of the node except those of the form $\circ^i\Box\delta$ with $i \geq 0$ and δ being any PLTL formula. We say that these formulae are persistent

When the rule is applied, the first branch represents the fulfilment of the eventuality and is identical to the $(\mathcal{U})$ or $(\Diamond)$ rules. In the case that the eventuality is not satisfied (second branch), indeed when the eventuality fulfilment is delayed, the negated context prevents the same situation from being repeated: the eventuality is not satisfied and the context had previously occurred.

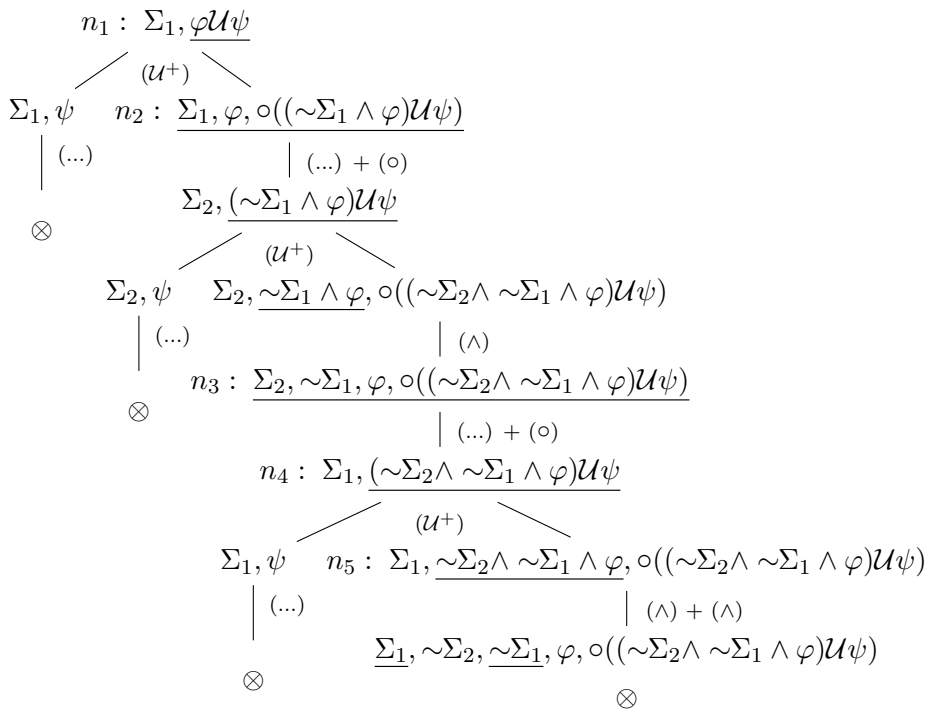


Figure 2.10: Use of context.

Figure 2.10 illustrates the use of the negated context. Note that the underlined formula at each node is the one to which the rule applies. In the node n_1 the context is Σ_1. $(\mathcal{U}^+)$ is applied and the node n_2 represents the case where the eventuality is

not satisfied. In that case, the negation of the context Σ_1 is included in the left-part of the *until-formula* (inside the *next formula*) not to allow repetition of that context. After applying the rest of the rules to the formulae that conform the node, next-state rule is applied. Let's consider that the context of this new node is different to Σ_1 (Σ_2). Node n_3 shows how when the eventuality is not satisfied, the previous negated context and the left-part of the eventuality come out and they are separated using the ($\wedge$) rule. Let's assume there is no contradiction between $\sim\Sigma_1$ and Σ_2. Once again, the negated context is introduced at the left part of the *until-formula* inside the *next formula*. Node n_4 represents a state where the context of the eventuality is exactly the same as in node n_1, Σ_1. If the eventuality is not satisfied (node n_5), the previous negated context comes out and is separated using the ($\wedge$) rule. As Σ_1 and $\sim\Sigma_1$ occur, the branch is closed.

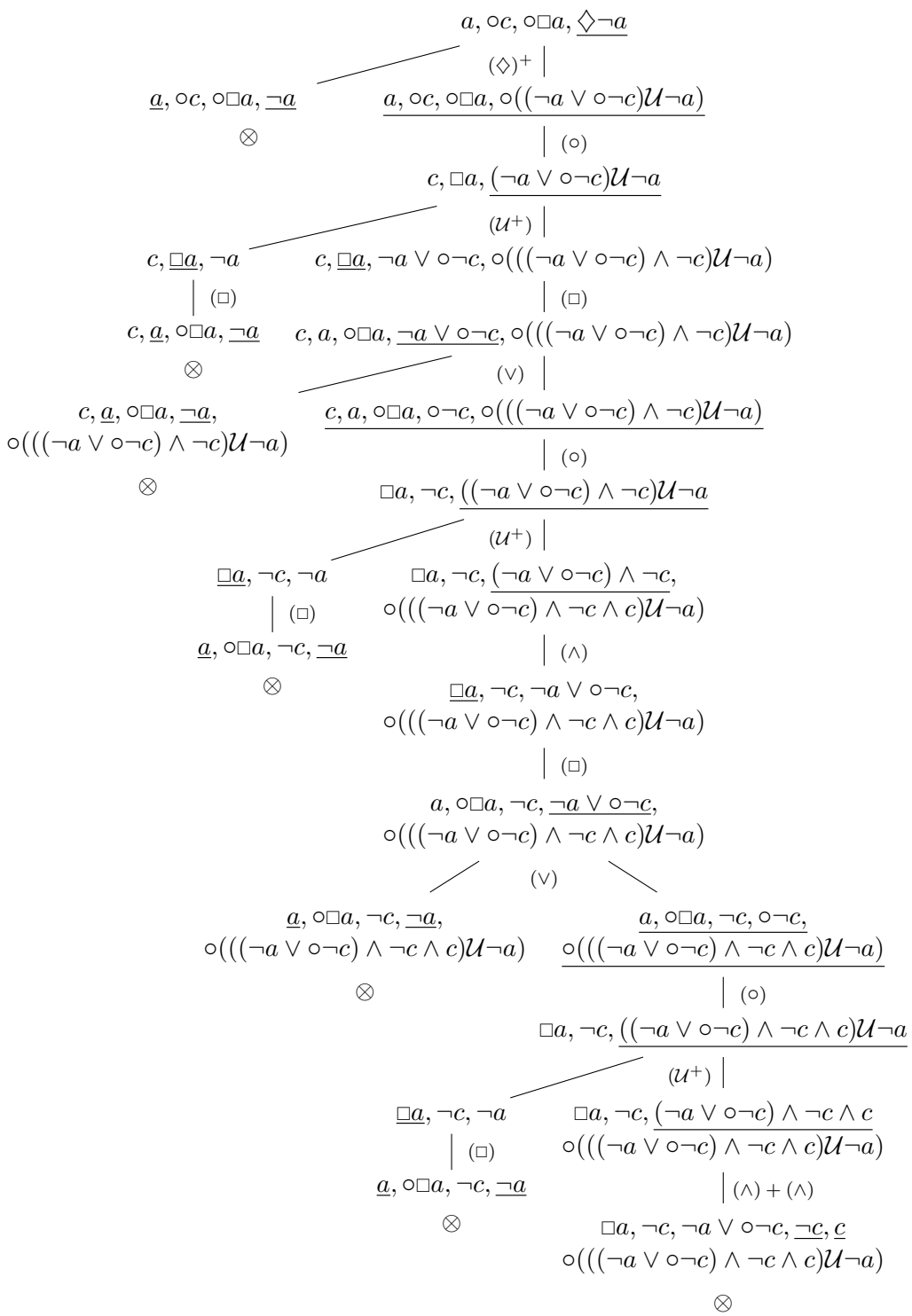


Figure 2.11: Closed tableau for $\{a, \circ c, \circ\Box a, \Diamond\neg a\}$

Figure 2.11 is an example of the use of the negated context and how the pattern explained in the previous example is met. Note that the formulae $\circ\Box a$ and $\Box a$ are excluded from the negation of contexts because they are persistent formulae. The non-

inclusion of persistent formulae in the negated contexts is based on Proposition 2.

Proposition 2. *The formula* $(\circ^{i-1}\Box\delta) \wedge (\circ^{i}\Diamond\neg\delta)$ *for any* $i > 0$ *is unsatisfiable.*

Proof.

$$\begin{aligned}
(\circ^{i-1}\Box\delta) \wedge (\circ^{i}\Diamond\neg\delta) &\equiv \circ^{i-1}(\Box\delta \wedge \circ\Diamond\neg\delta) \\
&\equiv \circ^{i-1}(\delta \wedge \circ\Box\delta \wedge \circ\Diamond\neg\delta) \\
&\equiv \circ^{i-1}(\delta \wedge \circ(\Box\delta \wedge \Diamond\neg\delta)) \\
&\equiv \circ^{i-1}(\delta \wedge \circ\mathbf{F}) \\
&\equiv \mathbf{F}
\end{aligned}$$

□

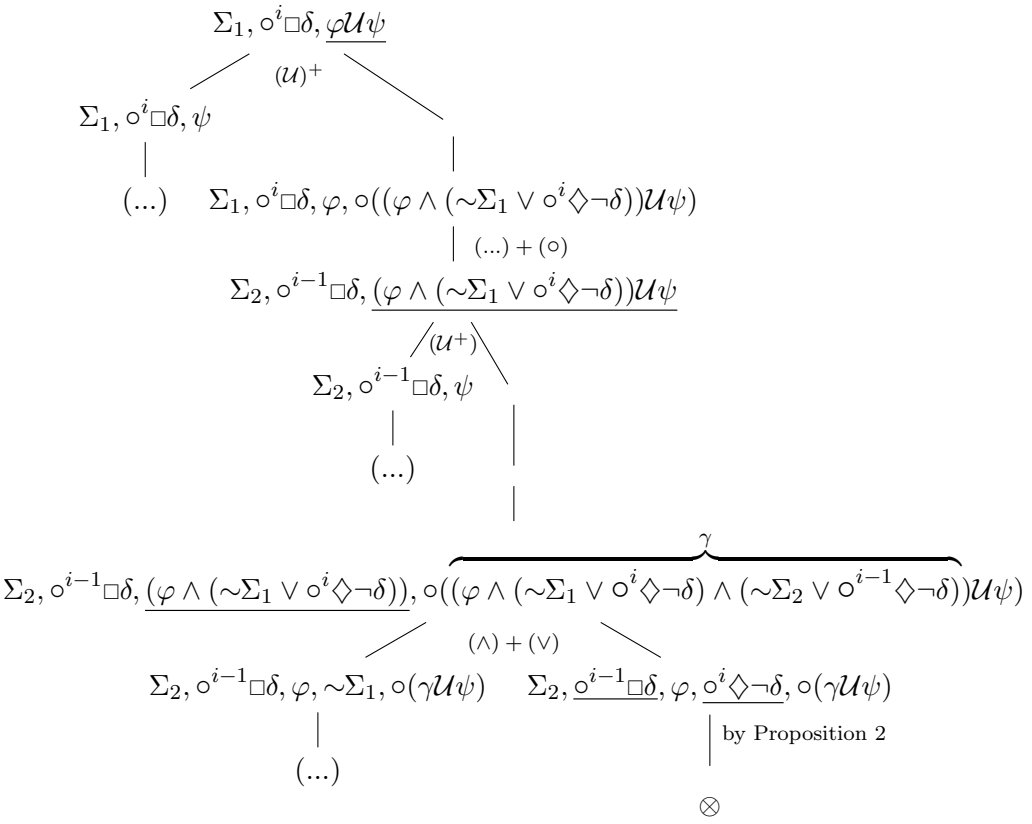


Figure 2.12: Exclusion of persistent formulae in the negated context

Proposition 2 ensures that tableaux for sets of formulae of form $\{\Sigma_1, \circ^i\Box\delta, \varphi\mathcal{U}\psi\}$ (like the one in Figure 2.12) can prune all branches corresponding to the persistent formulae that are included in the negated context, since these branches will always close. In other words, for all $i \geq 0$, it always happens that the sets of formulae

$$\{\circ^i\Box\delta, \varphi, \circ((\varphi \wedge (\sim\Sigma_1 \vee \circ^i\Diamond\neg\delta))\mathcal{U}\psi)\} \quad \text{and} \quad \{\circ^i\Box\delta, \varphi, \circ((\varphi \wedge \sim\Sigma_1)\mathcal{U}\psi)\}$$

are logically equivalent.

At this end of this subsection we would like to point out that the application of the β^+ rules is so important that it determines when a tableau is open. A tableau is closed if all its branches are closed. A tableau is open when one of its branches has a consistent node n and the following holds:

- the label of n is included in the label of a previous node in the branch.
- Along the branch, from the root until node n, the β^+ rule has been applied to all eventualities occurring in the branch.

Any open branch represents a cyclic PLTL-structure that corresponds with a model of the set of formulae labelling its root. The nature of our tableau rules guarantees that the tableaux always end up being either closed or open. A closed tableau indicates that the input does not have a model, hence it is unsatisfiable. An open tableau provides a cyclic PLTL-structure that satisfies the tableau input.

Tableaux Construction

We now present the algorithm that constructs a context-based tableau from an initial set of formulae. The management of eventualities is the core of the method, where the β^+ rules play the main role. Henceforth, we emphasise the details of the algorithm related to eventualities and the β^+ rules application.

Algorithm 1: `Tableau`
Input = Δ: set of formulae, `selectedFormula`: formula
Output = M: Model, b: boolean

```
1  if Incons(Δ) then
2  |  M, b := ∅, false;
3  else if there exists Δ', an ancestor of Δ, such that Δ ⊆ Δ' and all eventualities
       in the branch has been already selected then
4  |       ▷ where model is the model constructed from the branch where Δ' and Δ are.
5  |  M, b := model, true;
6  else if is_elementary(Δ) then
7  |  Δ' = apply-next-state-rule(Δ);
8  |  M, b := Tableau(Δ',selectedFormula);
9  else
10 |  selectedFormula = select_formula(Δ);
11 |  if selectedFormula ≠ F then
12 |  |  M, b := apply-β+-rule(Δ,selectedFormula);
13 |  Let ψ be any formula in Δ which is not elementary;
14 |  if α_is_applicable(ψ) then
15 |  |  M, b := apply-α-rule(ψ,Δ,selectedFormula);
16 |  else // β_is_applicable(ψ)
17 |  |  M, b := apply-β-rule(ψ,Δ,selectedFormula);
18 |  end
19 end
```

Algorithm 1 is a recursive procedure called `Tableau` whose input parameters are the set of formulae, Δ, which is the label of the current node in the tableau, and a selected eventuality (if there are any) to which apply the β^+ rules. Initially, Δ is the set of formulae for satisfiability checking and `selectedFormula` is `F` (no eventuality selected yet).

If Δ is not syntactically consistent the branch is closed (line 2). On the contrary if there is an ancestor of the current node, labelled by Δ', where $\Delta \subseteq \Delta'$ and all the eventualities have been selected, a cycle from Δ to Δ' is created and the branch is open (line 5). Otherwise, when Δ is elementary (see Definition 12), `Tableau` applies the next-state rule to jump to a new state and goes on from there with a recursive call (line 8). When Δ is a consistent non-elementary set of formulae, the algorithm must apply one tableau rule to some non-elementary formula φ in Δ. The choice of φ (and hence the applicable tableau rule) prioritizes eventualities and uses the `selectedFormula` to first apply a $(\beta)^+$ rule to exactly the selected eventuality (see Algorithm 2).

Algorithm 2: `apply-`β^+`-rule`
Input = Δ: set of formulae, `selectedFormula`: formula
Output = M: Model, b: boolean

```
1  Let nc the negated context of Δ in NNF;
2  if selectedFormula == φUφ' then
3  |  Δ'_1 := (Δ \ {φUφ'}) ∪ {φ'};
4  |  Δ'_2 := (Δ \ {φUφ'}) ∪ {φ, ○((nc ∧ φ)Uφ')};
5  |  selectedFormula' := (nc ∧ φ)Uφ';
6  else // selectedFormula == ♢φ
7  |  Δ'_1 := (Δ \ {♢φ}) ∪ {φ};
8  |  Δ'_2 := (Δ \ {♢φ}) ∪ {○(ncUφ)};
9  |  selectedFormula' := ncUφ;
10 end
11 pair = Tableau(Δ'_1,F);
12 if pair ≠ (∅, false) then
13 |  M, b := pair;
14 else
15 |  M, b := Tableau(Δ'_2,selectedFormula');
16 end
```

Algorithm 3: `apply-`α`-rule`
Input = ψ: formula, Δ: set of formulae, `selectedFormula`: formula
Output = M: Model, b: boolean

```
1  if ψ == φ ∧ φ' then
2  |  Δ' := (Δ \ {ψ}) ∪ {φ, φ'};
3  else // ψ == □φ
4  |  Δ' := (Δ \ {ψ}) ∪ {φ, ○□φ};
5  end
6  M, b := Tableau(Δ',selectedFormula);
```

Algorithm 4: `apply-`β`-rule`
Input = ψ: formula, Δ: set of formulae, `selectedFormula`: formula
Output = M: Model, b: boolean

```
1  if ψ == φ ∨ φ' then
2  |  Δ'_1 := (Δ \ {ψ}) ∪ {φ};  Δ'_2 := (Δ \ {ψ}) ∪ {φ'};
3  else if ψ == φ R φ' then
4  |  Δ'_1 := (Δ \ {ψ}) ∪ {φ, φ'};  Δ'_2 := (Δ \ {ψ}) ∪ {φ', ○(φ R φ')};
5  else if ψ == ◇φ then
6  |  Δ'_1 := (Δ \ {ψ}) ∪ {φ};  Δ'_2 := (Δ \ {ψ}) ∪ {○◇φ};
7  else // ψ == φ U φ'
8  |  Δ'_1 := (Δ \ {ψ}) ∪ {φ'};  Δ'_2 := (Δ \ {ψ}) ∪ {φ, ○(φ U φ')};
9  end
10 pair = Tableau(Δ'_1,selectedFormula);
11 if pair ≠ (∅, false) then
12 |  M, b := pair;
13 else
14 |  M, b := Tableau(Δ'_2,selectedFormula);
15 end
```

Before running Algorithm 2, `Tableau` makes sure that some eventuality (if any) is selected (line 10). This task is performed by `select_formula`, which works as follows. If `selectedFormula` is $\mathbf{F}$ and Δ contains eventualities, one of them is assigned to `selectedFormula`. Otherwise, `selectedFormula` does not change. In this way, two purposes are achieved: (1) if Δ contains one selected eventuality, then this selection is preserved and (2) if Δ contains at least one eventuality, some eventuality is selected. It should be noted that the selection mechanism is random, but `Tableau` must be implemented with a fair strategy, that is, an eventuality cannot remain unselected indefinitely.

Regarding Algorithm 2, it should be remarked that the application of the β^+ rule creates two new nodes with labels Δ'_1 and Δ'_2. The case of Δ'_1 corresponds to the fact that the selected eventuality is satisfied and consequently, another eventuality must be selected. On the other side, if the eventuality is not satisfied (case of Δ_2) the *until-formula* inside the *next formula* must remain selected to be treated later.

From line 13 to the end, `Tableau` applies either an α-rule or a β-rule to a chosen formula ψ in Δ (see Algorithms 3 and 4). Note that all eventualities except the selected one are treated with the ($\mathcal{U}$) and ($\Diamond$) rules.

Several examples of the algorithm are shown below. Examples 3 and 4 illustrate the algorithm running with unsatisfiable and satisfiable sets respectively.

Example 3 (Examples of closed tableaux)**.** *Figure 2.13 and 2.14 illustrate the tableau built from* $\{p\mathcal{U}\mathbf{F}\}$ *and* $\{\Box(a \wedge \neg b), \Diamond\neg a\}$ *respectively. Both sets of formulae are unsatisfiable and all the branches are closed. For readability, we mark the selected eventualities with a gray box.*

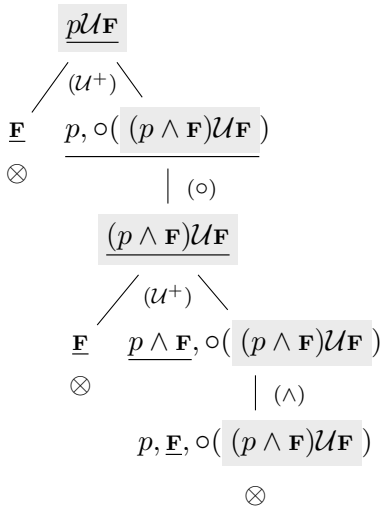


Figure 2.13: Example $p\mathcal{U}\mathbf{F}$

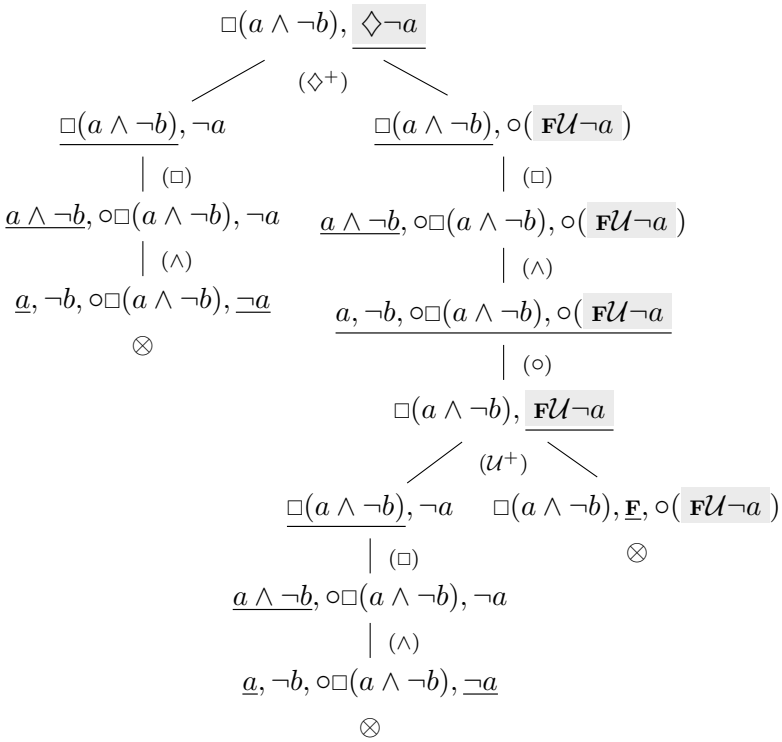


Figure 2.14: Example $\Box(a \wedge \neg b), \Diamond\neg a$

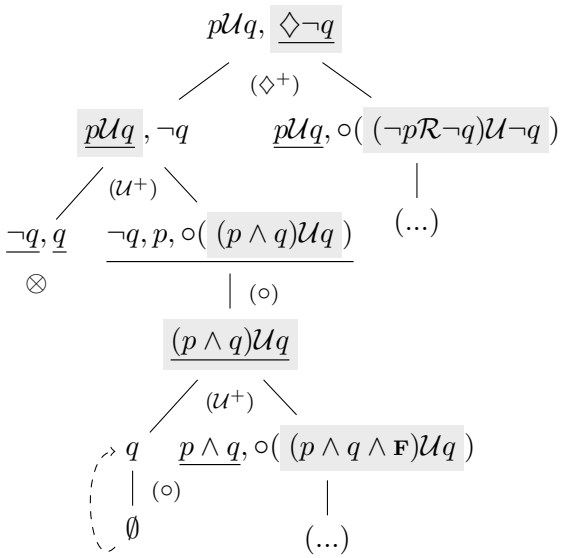


Figure 2.15: Example of satisfiable set of formulae: $\{p\mathcal{U}q, \Diamond\neg q\}$

Example 4 (Example of an open tableau)**.** *Figure 2.15 shows the systematic tableau for* $\{p\mathcal{U}q, \Diamond\neg q\}$*. The example is borrowed from [41]. In contrast to [41], the procedure presented in Figure 2.15 is the adapted one for* NNF *formulae. From the leftmost open branch of the tableau, a* PLTL*-structure can be obtained, meaning that the set is satisfiable. In this case we obtain the* PLTL*-structure* M *such that* $M \models \{p\mathcal{U}q, \Diamond\neg q\}$*:* $V_M(s_0) = \{p\}$*,* $V_M(s_{2k-1}) = \{q\}$*,* $V_M(s_{2k}) = \{\}$ *for every* $k \geq 1$*. We represent* M *as* $\{p\}, \langle\{q\}, \{\ \}\rangle^w$*.*

2.2.2 Sequent Calculus

The NNF version of TTC sequent calculus introduced in [41] is presented in this subsection.

Sequent calculus were introduced by Gentzen [42]. Similar to axiomatic systems, a Gentzen system has axioms and inference rules. But, unlike a deductive system, the basic building blocks in a Gentzen system are expressions called sequents, not formulae. Sequents are formed by two sequences of formulae separated by $\vdash$: $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi_1, \psi_2, \ldots, \psi_m$ interpreted as

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

being $\rightarrow$ the classical implication connective. $\varphi_1, \varphi_2, \ldots \varphi_n$ is called the *antecedent* and the sequent $\varphi_1, \varphi_2, \ldots, \varphi_m$ its *consequent*.

The inference rules in a sequent calculus indicate that a sequent may be inferred from a set of sequents. In fact, TTC (Tait-style Temporal Calculus) provides sequent rules for inferring $false$. In Figure 2.16 there is a table which relates each tableau rule with its respective TTC rule. Note that TTC contains two axioms: (Ctd) and $(\mathbf{F})$.

	Tableau Rules	Sequent Calculus
(Ctd)	$\dfrac{\Sigma, \varphi, \sim\varphi}{\text{Closed branch}}$	$\dfrac{}{\Sigma, \varphi, \sim\varphi \vdash \mathbf{F}}$
$(\mathbf{F})$	$\dfrac{\Sigma, \mathbf{F}}{\text{Closed branch}}$	$\dfrac{}{\Sigma, \mathbf{F} \vdash \mathbf{F}}$
$(\wedge)$	$\dfrac{\Sigma, \varphi_1 \wedge \varphi_2}{\Sigma, \varphi_1, \varphi_2}$	$\dfrac{\Sigma, \varphi_1, \varphi_2 \vdash \mathbf{F}}{\Sigma, \varphi_1 \wedge \varphi_2 \vdash \mathbf{F}}$
$(\Box)$	$\dfrac{\Sigma, \Box\varphi}{\Sigma, \varphi, \circ\Box\varphi}$	$\dfrac{\Sigma, \varphi, \circ\Box\varphi \vdash \mathbf{F}}{\Sigma, \Box\varphi \vdash \mathbf{F}}$
$(\circ)$	$\dfrac{\Sigma, \circ(\Sigma_2)}{\Sigma_2}$	$\dfrac{\Sigma_1 \vdash \mathbf{F}}{\Sigma_2, \circ(\Sigma_1) \vdash \mathbf{F}}$
$(\vee)$	$\dfrac{\Sigma, \varphi_1 \vee \varphi_2}{\Sigma, \varphi_1 \mid \Sigma, \varphi_2}$	$\dfrac{\Sigma, \varphi_1 \vdash \mathbf{F} \qquad \Sigma, \varphi_2 \vdash \mathbf{F}}{\Sigma, \varphi_1 \vee \varphi_2 \vdash \mathbf{F}}$
$(\mathcal{R})$	$\dfrac{\Sigma, \varphi_1 \mathcal{R} \varphi_2}{\Sigma, \varphi_1, \varphi_2 \mid \Sigma, \varphi_2, \circ(\varphi_1 \mathcal{R} \varphi_2)}$	$\dfrac{\Sigma, \varphi_1, \varphi_2 \vdash \mathbf{F} \qquad \Sigma, \varphi_2, \circ(\varphi_1 \mathcal{R} \varphi_2) \vdash \mathbf{F}}{\Sigma, \varphi_1 \mathcal{R} \varphi_2 \vdash \mathbf{F}}$
$(\mathcal{U})$	$\dfrac{\Sigma, \varphi_1 \mathcal{U} \varphi_2}{\Sigma, \varphi_2 \mid \Sigma, \varphi_1, \circ(\varphi_1 \mathcal{U} \varphi_2)}$	$\dfrac{\Sigma, \varphi_2 \vdash \mathbf{F} \qquad \Sigma, \varphi_1, \circ(\varphi_1 \mathcal{U} \varphi_2) \vdash \mathbf{F}}{\Sigma, \varphi_1 \mathcal{U} \varphi_2 \vdash \mathbf{F}}$
$(\Diamond)$	$\dfrac{\Sigma, \Diamond\varphi}{\Sigma, \varphi \mid \Sigma, \circ\Diamond\varphi}$	$\dfrac{\Sigma, \varphi \vdash \mathbf{F} \qquad \Sigma, \circ\Diamond\varphi \vdash \mathbf{F}}{\Sigma, \Diamond\varphi \vdash \mathbf{F}}$
$(\mathcal{U})^+$	$\dfrac{\Sigma, \varphi_1 \mathcal{U} \varphi_2}{\Sigma, \varphi_2 \mid \Sigma, \varphi_1, \circ((\varphi_1 \wedge \sim\Sigma')\mathcal{U}\varphi_2)}$	$\dfrac{\Sigma, \varphi_2 \vdash \mathbf{F} \qquad \Sigma, \varphi_1, \circ((\varphi_1 \wedge \sim\Sigma')\mathcal{U}\varphi_2) \vdash \mathbf{F}}{\Sigma, \varphi_1 \mathcal{U} \varphi_2 \vdash \mathbf{F}}$
$(\Diamond)^+$	$\dfrac{\Sigma, \Diamond\varphi}{\Sigma, \varphi \mid \Sigma, \circ((\sim\Sigma')\mathcal{U}\varphi)}$	$\dfrac{\Sigma, \varphi \vdash \mathbf{F} \qquad \Sigma, \circ((\sim\Sigma')\mathcal{U}\varphi) \vdash \mathbf{F}}{\Sigma, \Diamond\varphi \vdash \mathbf{F}}$

Figure 2.16: Sequent calculus associated with the tableau rules.

Thanks to the correspondence between tableau rules and sequents, it is possible to obtain a refutational proof that is dual to a closed tableau. It starts from the closed leaves of the tableau, contradiction or $\mathbf{F}$, deriving $\mathbf{F}$ by applying the sequents (Ctd) or $(\mathbf{F})$ respectively. Then it continues applying the different rules until it arrives to the root of the tableau. Finally the root derives $\mathbf{F}$.

In Figure 2.17 there is a refutational proof that corresponds to the closed tableau in Figure 2.14. In Figure 2.18 there is a refutational proof that corresponds to the closed tableau in Figure 2.13.

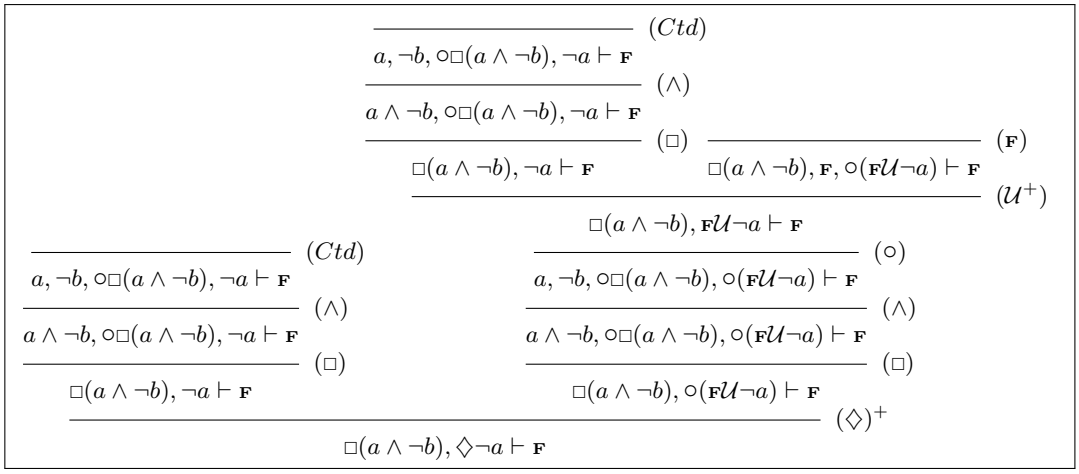


Figure 2.17: Sequent proof for $\Box(a \wedge \neg b), \Diamond\neg a$

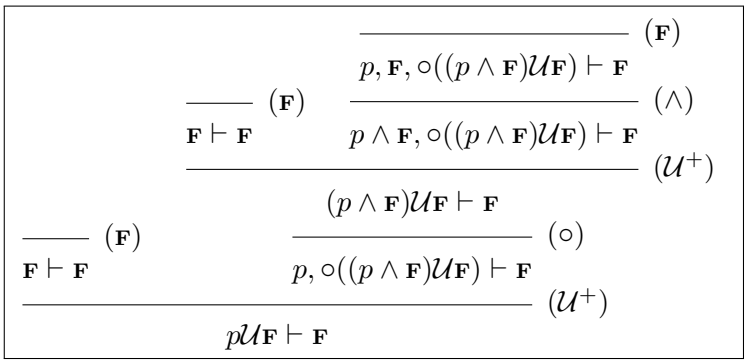


Figure 2.18: Sequent proof for $p\mathcal{U}\mathbf{F}$

2.3 Sequent Calculus in Isabelle

We have codified a version of the calculus TTC in Section 2.2.2 (for formulae in NNF) as an Isabelle theory ([75]). Isabelle is a generic theorem prover, designed for interactive reasoning in a variety of formal theories. All Isabelle files can be found at `http://github.com/alexlesaka/OnePassTableau`. The file `TTC_Calculus.thy` contains the encoding of the sequent rules introduced in Figure 2.16, and another file `TTC_Soundness.thy` provides the soundness proof of them.

We use a datatype PLTL_formula to define the syntax of the considered formulae, which are the two boolean constants ($\mathbf{T}$ and $\mathbf{F}$), the atoms (strings preceded by constructor Var, or shortly V), classical connectives (preceded by a dot, to avoid conflicts) of negation ($.\neg$), conjunction ($.\wedge$), disjunction ($.\vee$), along with temporal operators for next, until, release, eventually and always. To automate the Isabelle proofs, we add two extra connectives $\ddot{\mathcal{U}}$ (the until operator surrounded by dieresis) to denote the selected eventuality in goals and $\ddot{\circ}$ to mark the sequents formed by elementary formulae. In addition, we include the implication connective ($.\longrightarrow$) to facilitate the readability of the proofs.

```
datatype PLTL_formula =
  F
| T
| Var string ("(V _)" [80] 80)
| Not PLTL_formula ("(.¬ _)" [80] 80)
| And  PLTL_formula  PLTL_formula (infixl ".∧" 75)
| Or PLTL_formula  PLTL_formula (infixl ".∨" 73)
| Imp PLTL_formula  PLTL_formula (infixr ".⟶" 70)
| X PLTL_formula ("(○ _)" [80] 80)
| U PLTL_formula  PLTL_formula (infixr "𝒰" 72)
| R PLTL_formula  PLTL_formula (infixr "ℛ" 72)
| Alw PLTL_formula ("(□ _)" [80] 80)
| Evt PLTL_formula ("(♢ _)" [80] 80)
(* Selected Next and Until,
   Extra-logic connectives for marking*)
| SelX PLTL_formula ("(¨○¨ _)" [80] 80) (* Marked Next *)
| SelU PLTL_formula  PLTL_formula (infixr "¨𝒰¨" 72)
(* Selected Until *)
```

We also have a function that transforms any formula into its negation in NNF.

```
fun notInNNF  :: "PLTL_formula ⇒ PLTL_formula" ("∼nnf")  where
  "∼nnf F = T" |
  "∼nnf T = F" |
  "∼nnf(V x) = (.¬(V x))" |
  "∼nnf(.¬ phi) =  phi" |
  "∼nnf(phi .∧ psi) = ((∼nnf phi) .∨ (∼nnf psi))" |
  "∼nnf(phi .∨ psi) = ((∼nnf phi) .∧ (∼nnf psi))" |
  "∼nnf(phi .⟶ psi) = ((phi .∧ (∼nnf psi))" |
  "∼nnf(○ phi) = ○(∼nnf phi)" |
  "∼nnf(¨○¨ phi) = ¨○¨(∼nnf phi)" |
  "∼nnf(phi 𝒰 psi) = (∼nnf phi) ℛ (∼nnf psi)" |
  "∼nnf(phi ¨𝒰¨ psi) = (∼nnf phi) ℛ (∼nnf psi)" |
  "∼nnf(phi ℛ psi) = (∼nnf phi) 𝒰 (∼nnf psi)" |
  "∼nnf(□ phi) = ♢ (∼nnf phi)" |
  "∼nnf(♢ phi) = □ (∼nnf phi)"
```

We define the TTC rules by an inductive binary relation (predicate) TTC_proves, which is denoted "⊢" in infix notation. The first argument of ⊢ is a set of PLTL formulae (implemented as an ordered list without repeated elements) and the second is a PLTL formula. By the construction of the calculus, the consequents of the sequents are always the constant $\mathbf{F}$, but (for clarity) we prefer to keep ⊢ as a binary relation, and to explicitly represent that falsehood. The Isabelle definition of ⊢ includes the two axioms:

$$\textit{TTC\_Ctd1}: \quad \varphi \mathbin{.\in} \Delta \Longrightarrow (\sim\text{nnf}\ \varphi) \mathbin{.\in} \Delta \Longrightarrow \Delta \vdash \mathbf{F}$$
$$\textit{TTC\_Ctd2}: \quad \mathbf{F} \mathbin{.\in} \Delta \Longrightarrow \Delta \vdash \mathbf{F}$$

where $.\in$ is the user-defined infix operator for member of a list, and the user-defined infix function ∼nnf computes the negation normal form of a negated formula. We also

encode the traditional rules for classical connectives and temporal operators:

$$
\begin{array}{ll}
\mathit{TTC\_T}: & \Delta \vdash \mathbf{F} \Longrightarrow \mathbf{T} \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_And}: & \varphi \bullet \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi \mathbin{.\wedge} \psi) \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_Or}: & \varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi \mathbin{.\vee} \psi) \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_Imp}: & (\sim \mathrm{nnf}\ \varphi) \bullet \Delta \vdash \mathbf{F} \Longrightarrow \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi \mathbin{.\longrightarrow} \psi) \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_U}: & \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \varphi \bullet \circ(\varphi \mathcal{U} \psi) \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi \mathcal{U} \psi) \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_R}: & \varphi \bullet \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \psi \bullet \circ(\varphi \mathcal{R} \psi) \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi \mathcal{R} \psi) \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_Alw}: & \varphi \bullet \circ\Box\varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\Box\varphi) \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_Evt}: & \varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \circ\Diamond\varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\Diamond\varphi) \,\#\, \Delta \vdash \mathbf{F}
\end{array}
$$

where # is the standard *cons* constructor of lists and $\bullet$ is our user-defined operator for insert an element φ in the correct position of an ordered list Δ (if φ is already in Δ, then the result is Δ itself). For automating proofs, we have defined an order on the set of PLTL formulae where the minimal formulae are literals, and formulae with connectives are lexicographic ordered according to the following order (from lowest to highest): $\ddot{\circ}\ddot{}$, $\circ$, $.\wedge$, $.\vee$, $\Box$, $\mathcal{R}$, $\Diamond$, $\mathcal{U}$, $\ddot{\mathcal{U}}\ddot{}$. We order the antecedent of the sequent, in decreasing order, at the beginning of any proof using the following extra rule:

$$\mathit{TTC\_Interchange}: \quad (\mathsf{sort}\ \Delta) \vdash \mathbf{F} \Longrightarrow \Delta \vdash \mathbf{F}$$

Then, every application of a rule preserves the order in the generated subgoals by means of the operator $\bullet$ that inserts each formula in the correct place. As a consequence, the first formula of any sequent is a *non-elementary* formula, if there exists at least one. Moreover, the first formula is the eventuality, if there exists at least one, and it is the selected eventuality whenever the selection has already been done. The rules applied to the selected eventuality are:

$$
\begin{array}{ll}
\mathit{TTC\_Evt\_Plus}: & \varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \circ((\mathsf{negCtxt}\ \Delta)\mathcal{U}\varphi) \bullet \Delta \vdash \mathbf{F} \\
& \Longrightarrow \Diamond\varphi \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_U\_Plus}: & \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \varphi \bullet \circ((\varphi \mathbin{.\sqcap.} (\mathsf{negCtxt}\ \Delta))\ddot{\mathcal{U}}\psi) \bullet \Delta \vdash \mathbf{F} \\
& \Longrightarrow \varphi\mathcal{U}\psi \,\#\, \Delta \vdash \mathbf{F} \\
\mathit{TTC\_U\_Sel}: & \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \varphi \bullet \circ((\varphi \mathbin{.\sqcap.} (\mathsf{negCtxt}\ \Delta))\ddot{\mathcal{U}}\psi) \bullet \Delta \vdash \mathbf{F} \\
& \Longrightarrow \varphi\ddot{\mathcal{U}}\psi \,\#\, \Delta \vdash \mathbf{F}
\end{array}
$$

where $(\mathsf{negCtxt}\ \Delta)$ is the negation of the context Δ and the operator $.\sqcap.$ is a conjunction up to subsumption, hence we avoid adding subsumed disjunctions, in particular, adding duplicated disjunctions. Note that the premises of *TTC_U_Sel* are really a copy of the premises of *TTC_U_Plus*. Consequently, *TTC_U_Plus* is applied at the first time when the eventuality has been just selected, whereas *TTC_U_Sel* is applied after that, while it is kept selected. When all formulae in the sequent are elementary we apply the following rule:

$$\mathit{TTC\_Next\_State}: \quad (\mathsf{next\_state}\ \Delta) \vdash \mathbf{F} \Longrightarrow \Delta \vdash \mathbf{F}$$

This rule applies when all the formulae in Δ are elementary, in such a way that Function next_state filters all formulae in Δ starting by the $\circ$ operator, removing from them this operator.

The soundness proof of each TTC rule is included in the file `TTC_Soundness.thy`. These proofs need prior definitions and lemmas. Some of them are presented below.

The first example is `Structure`, which corresponds to Definition 2. The `suffix` definition states that the infinite sequence (`suffix` σ n) is σ without its first n states. Then auxiliary lemmas like `suffix0` and `suffix_add` are automatically proved by Isabelle using the `suffix` definition.

```
type_synonym Structure = "nat ⇒ string set"

definition suffix :: "Structure ⇒ nat ⇒ Structure"
  where "suffix σ n = (λi.(σ (n+i)))"

lemma suffix0 [simp]: "suffix σ 0 = σ"
  by (simp add: suffix_def)

lemma suffix_add [simp]: "suffix (suffix σ m) n = suffix σ (m+n)"
  by (simp add: suffix_def add_ac)
```

`Holds` corresponds to Definition 3. In the case of the formalisation in Isabelle, we chose $j = 0$, i.e., the state s_j of the structure M is s_0.

`Models` defines whether a structure is a model of a list of PLTL formulae.

```
primrec holds :: "Structure ⇒ PLTL_formula ⇒ bool"  ("(_ ⊨ _)"
    [60,60] 59)
where
  "σ ⊨ F = False" |
  "σ ⊨ T = True" |
  "σ ⊨ V s = (s ∈ (σ 0))" |
  "σ ⊨ .¬ φ  = (¬(σ ⊨ φ))" |
  "σ ⊨ φ .∧ ψ = (σ ⊨ φ ∧  σ ⊨ ψ)" |
  "σ ⊨ φ .∨ ψ = (σ ⊨ φ ∨  σ ⊨ ψ)" |
  "σ ⊨ φ .⟶ ψ  = (σ ⊨ φ  ⟶ σ ⊨ ψ)" |
  "σ ⊨ □ φ  = (∀i. suffix σ i  ⊨ φ)" |
  "σ ⊨ ○ φ  = (suffix σ 1  ⊨ φ)" |
  "σ ⊨ ¨○¨ φ  = (suffix σ 1  ⊨ φ)" |
  "σ ⊨ φ 𝒰 ψ = (∃i. suffix σ i  ⊨ ψ ∧ (∀j<i. suffix σ j  ⊨ φ))" |
  "σ ⊨ ◇ φ  = (∃i. suffix σ i  ⊨ φ)" |
  "σ ⊨ φ ℛ ψ = (¬(∃i.(¬(suffix σ i  ⊨ ψ) ∧
    (∀j<i. ¬(suffix σ j  ⊨ φ))))" |
  "σ ⊨ φ ¨𝒰¨ ψ = (∃i. suffix σ i  ⊨ ψ ∧ (∀j<i. suffix σ j  ⊨ φ))"

definition models :: "Structure ⇒ PLTL_formula list ⇒ bool"
    ("(_ |⊨ _)" [60,60] 59)
  where "models σ Φ = (∀φ. (φ .∈ Φ⟶σ ⊨ φ))"
```

The definitions of Sat and UnSat, which correspond to Definition 4, are as follows.

```
definition sat :: "PLTL_formula list ⇒ bool"
  where "sat Δ = (∃σ. σ |⊨ Δ)"

definition unSat :: "PLTL_formula list ⇒ bool"
  where "unSat Δ = (¬(sat Δ))"
```

Finally we present an example of an auxiliary lemma that is used by the soundness proofs. Isabelle automatically proves this lemma by induction.

```
lemma equiv_NNF [simp]:
```

```
      shows "(¬(σ ⊨ φ)) = (σ ⊨ ∼nnf φ)"
      by (induct φ arbitrary: σ) auto
```

Once the necessary definitions and lemmas are established, the proof of soundness of each inference rule can be made by refutation. Next we present the proofs of the rules `TTC_Ctd1` and `TTC_U`.

```
(******************** TTC_Ctd1 soundness ***************)
lemma TTC_Ctd1_Sound:
  assumes "φ .∈ Δ"
  assumes "∼nnf φ .∈ Δ"
  shows "unSat Δ"
proof (rule ccontr)
  assume "¬(unSat Δ)"
  hence "∃σ. σ |⊨ Δ "  by (simp add: models_def)
  then have "∃ σ. σ ⊨ ∼nnf φ  ∧ σ ⊨ φ" using models_def
    by auto
  then show "False" using equiv_NNF
qed
```

```
(******************** TTC_U soundness ****************)

lemma TTC_U_Sound [simp]:
  assumes "unSat(ψ • Δ)"
  assumes "unSat(φ • (○(φ 𝒰 ψ)) • Δ)"
  shows "unSat((φ 𝒰 ψ) # Δ)"
proof (rule ccontr)
  assume "¬(unSat((φ 𝒰 ψ) # Δ))"
  then obtain σ where "σ |⊨((φ 𝒰 ψ) # Δ)" using sat_def unSat_def
  then obtain i:: nat where hyp:
        "suffix σ i  ⊨ ψ ∧ (∀j<i. suffix σ j  ⊨ φ) ∧ σ |⊨ Δ"
            using models_def by auto
  then have "sat(ψ • Δ) ∨ sat(φ • (○(φ 𝒰 ψ)) • Δ)"
  proof (cases)
    assume  "i = 0"
    then show ?thesis by (auto simp add: models_def sat_def)
  next
    assume "¬(i = 0)"
    then have "i > 0 ∧ σ ⊨ φ ∧
               suffix (suffix σ (i-1)) 1 ⊨ ψ ∧
               (∀j<i. suffix σ j  ⊨ φ) ∧
                σ |⊨ Δ" using hyp by auto
    then have  "σ ⊨ φ ∧ σ ⊨  ○(φ 𝒰 ψ) ∧ σ |⊨ Δ" by auto
    thus ?thesis  by (auto simp add: models_def sat_def)
  qed
  with assms show False using sat_def unSat_def
qed
```

3. CERTIFIED MODEL CHECKING FOR PLTL

In Symbolic Model Checking, since the system to be tested is represented by a formula, and nothing forces the user to give an specification with a unique model, the formula to be handled is often satisfied by a collection of 'logical' models. For a simple example, in a symbolic model checker such as NuSMV one can specify that the initial state satisfies the property a and that we have a unique transition from any state that satisfies a to a state that satisfies b. Then, the formula that represents this transition system is equivalent to the temporal formula $a \ \wedge \ \Box(a \rightarrow \circ b)$. This formula has more than one model in temporal logic. Indeed, if one asks a symbolic model checker whether the property $\Diamond\Box b$ is satisfied or not, it returns the counterexample $\langle\{a\},\{b\}\rangle^{\omega}$; and for the question $\Box\Diamond a$ the returned counterexample is $\{a\},\langle\{b\}\rangle^{\omega}$. In other words, from the logical point of view, symbolic model checking really decides whether a property (temporal formula) φ is satisfied in all the models of the given specification (a set of premises) Φ, that is $\Phi \models \varphi$. Hence, the underlying metalogical concept is logical consequence that corresponds with derivability (since PLTL is a complete logic). The crucial fact is that the allowed formulae for specifying the transition system are a syntactical restriction of general temporal formulae. In other words, model checking is a particular case of temporal deduction, i.e. the deduction of a (general) temporal formula from a set of (non-general) temporal formulae as premises.

When the specification of the transition system is very complex, the user may not be sure if the specification is well written. Here it is important to have techniques that not only provide a counterexample, but also a certificate that the transition system meets the property. This is exactly what the *Certified Model Checking (CMC)* has been introduced for. In this scenario a number of techniques have been previously proposed. Some of the techniques that deal with finite-state systems can be found in [59, 74]. In [59] an automata-theoretic approach to model checking is addressed. In [74] a deductive proof system was introduced for verifying branching time properties expressed in the mu-calculus. For infinite-state systems, Mebsout et al. [72] presented a new technique for generating and verifying proof certificates for SMT-based model checkers, focusing on proofs of invariant properties. The use of invariants has been exploited in [49, 58], where the proof is generated from the inductive invariant obtained with the k-liveness algorithm [23]. The resulting approach can be implemented as a model checker based on the combination of k-liveness with an engine for invariant properties that is capable of producing inductive invariants. A drawback of this approach is that, although it is very competitive, the task of finding counterexamples and the task of generating proofs (in this case via finding invariants) are very different, requiring for the latter the addition of extra mechanisms to the own model checker.

The CMC (see Figure 3.1) differs from the traditional model checking in providing a proof of the satisfiability of the given property, and not only a counterexample. Incor-

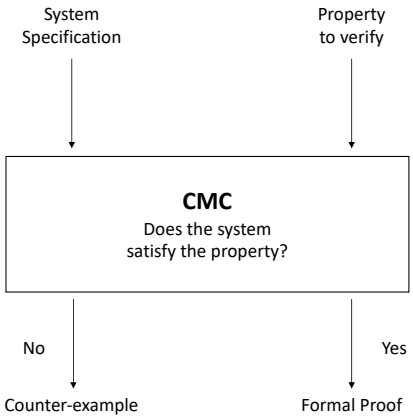


Figure 3.1: General schema of CMC

porating the notation of [56] (and slightly modifying it) we represent the methodology of the certified model checking as the following signature:

$$\mathcal{C}MC :: S \times \varphi \longrightarrow \mathbb{B} \times (Proof \mid Counter\text{-}example)$$

where, given a specification of a transition system, S, and a property, φ, a certified model checking produces a Boolean result, $\mathbb{B}$, indicating whether S satisfies φ, along with

- a proof (or certification), in the positive case, or
- a counterexample, in the negative case.

However, we believe that to take the full advantage of CMC, and enabling its industrial application to real systems, we need to ensure that CMC meets the following requirements.

(i) Proofs should be generated automatically.

(ii) An CMC needs to offer a CMC user sufficient information to understand the proof without additional 'costs' related to specialist knowledge of the underlying proof technique.

(iii) The presentation of the proof should enable the CMC users to easily navigate through its trace. This becomes particularly important when the system is badly defined – here the navigation through the trace can help to detect errors.

(iv) Finally, when developing a safety critical system following a safety process (e.g. processes of standard ISO26262 [52], IEC61508 [51] or EN50128 [20]), it is mandatory to analyse how a bug in a tool (a model checker in our case) may affect

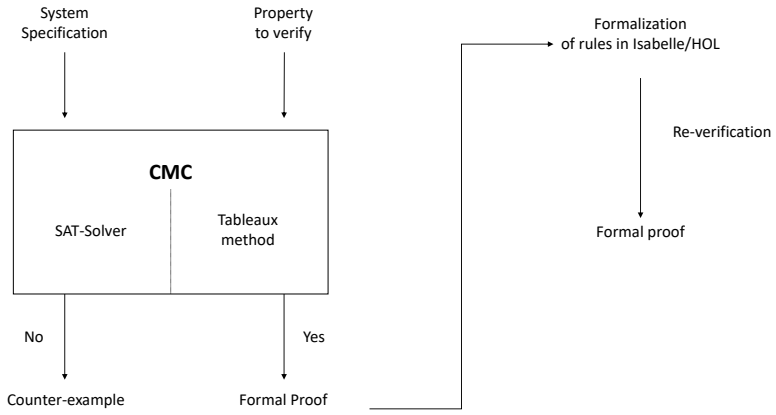


Figure 3.2: General schema of the proposal

the safety of the developed system. Depending on the level of these effects, some actions are required to increase our confidence in the model checker. One of the mechanisms to increase the probability of finding a failure is to re-evaluate the outputs of the CMC by an independent tool developed by an independent team.

The dual tableaux method introduced in Section 2.2 can be used to perform certified model checking by taking advantage of the following benefits: the tableau is built on-the-fly; due to the use of the contexts, the eventualities are satisfied as soon as possible. In terms of the implementation, all branches are completely independent so they can be parallelised without shared data. Regarding storage problems, we have a potential memory improvement by only requiring to keep traces (of the branch that we deal at the moment). Moreover, the use of sequent calculus provides a certificate in case the tableaux close, completing the Certified Model Checking scenario.

We propose a particular CMC method of realising the CMC philosophy (see Figure 3.2) meeting the characteristics (i)-(iv) above while maintaining the common (for the traditional Model Checker) functionality. On the one hand, our method is centered on the context-based temporal one-pass tableaux technique (introduced in Section 2.2) to perform traditional model checking optimised by a SAT solver. On the other hand, the dual sequent calculus of the tableaux method is used to generate a certificate in the case where the property is fulfilled.

The chapter is divided into the following sections:

- Section 3.1 describes everything related to performing model checking using the tableaux method: first the use of the tableaux is explained (3.1.1). Afterwards, the optimisation using SAT solvers is described (3.1.2). Finally, the algorithm integrating these tools is shown (3.1.3).

- Section 3.2 shows different examples illustrating what was explained in Section 3.1.

- Section 3.3 describes the use of certifications. We present two types of proofs with different degrees of granularity: the small-step proofs and the big-step ones (3.3.1).

3.1 Model Checking

This section discusses how to carry out Model Checking (without taking into account certificates) using the tableaux method explained in Section 2.2.

Our model checker receives as input a specification, S, of a transition system and a property *prop*, both written in the PLTL language.

3.1.1 Model Checking using Context-Based Tableaux

Whereas *prop* is any PLTL formula, transition system specifications are restricted to a sub-language.

The specification S consists of a set *Init* that is a non-temporal (or classical) formula and a conjunction (set) TR of formulae of the form $\Box\rho$, where ρ is a boolean combination of literals, and $\circ\ell$,where ℓ is a literal. *Invariant* is a set of *always-formulae*. $S = Init \wedge \text{TR} \wedge Invariant$ specifies the transition systems such that:

- A state is initial if and only if it satisfies the formula *Init*.
- Any pair '(current state, next state)' is in the transition relation if, and only if, it satisfies TR.
- *Invariant* contains any other always formula that every state in the transition system satisfies.

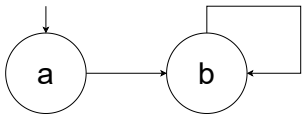


Figure 3.3: Example of a transition system

Example 5. *One possible representation of the transition system described in Figure 3.3 is the following one:* $Init = \{a\}$, $TR = \{\Box(a \rightarrow \circ b), \Box(b \rightarrow \circ b)\}$, $Invariant = \{\Box(a \rightarrow \neg b), \Box(b \rightarrow \neg a)\}$.

Then, given a specification S and a property *prop*, the model checker decides whether any model of S satisfies *prop*, by deciding if $S \cup \{\neg prop\}$ is unsatisfiable or not. The context-based tableau method is suitable for deciding the (un)satisfiability problem $S \cup \{\neg prop\}$, but to use it we need to transform the input into NNF. It explicitly tries to generate a model of $S \cup \{\neg prop\}$. The results in any particular tableau are interpreted as follows:

- If a cycle is found in a tableau branch, this branch is open, and represents a model that satisfies the set of formulae with which the tableau has been called. Consequently, this is a counterexample proving that the system S does not satisfy the property *prop*.
- If the tableau closes, i.e all its leaves are inconsistent sets, it means that the tableau input is unsatisfiable, hence, all models of S satisfy the property *prop*.

3.1.2 Optimization using a SAT Solver

A large proportion of the computational effort of the context-based tableaux is spent on classical propositional reasoning. Since the specification, S, of the system is the most determining factor, the use of the method is especially inefficient when the specified system is large. However, S involves very simple temporal formulae: always-formulae with arguments that are boolean combinations of literals and literals preceded by 'next'.

Definition 13 (Quasi-propositional and permanent formulae)**.** *Any boolean combination of literals and literals preceded by 'next' is a quasi-propositional formula. Formulae of the form $\Box\rho$, where ρ is a quasi-propositional formula are said to be permanent formulae.*

Definition 14 (Pre-next state)**.** *A node whose label is a set consisting only of quasi-propositional formulae, next formulae, and permanent formulae is called a pre-next state.*

Formulae in TR and most of the formulae in *Invariant* are permanent formulae. Permanent formulae are of the form $\Box\rho$, and according to the tableau rules, ρ and $\Box\rho$ are maintained in all states. Therefore, renaming in ρ the formulae $\circ\ell$ by fresh literals ℓ', we make ρ purely propositional. Hence, for a more efficient implementation of the context-based tableaux, we propose to add a SAT solver to carry out the quasi-propositional reasoning in the tableau.

Initially, we take all permanent formulae, extracted from TR and *Invariant* and construct the set P with them. We initialise the tableau with P, *Init*, the remaining formulae of *Invariant* and the negated property. Next, we apply the tableau rules to the temporal formulae that are neither permanent nor quasi-propositional. Subsequently, at each node that is a pre-next state (according to Definition 14), the SAT solver is called. The input of the SAT solver is the set of all quasi-propositional formulae together with the permanent formulae duly transformed into propositional ones. To do so, the set P is taken together with all formulae that appear new in the node and are permanent. Then, their $\Box$ operators are removed. In addition, each formula of the form $\circ l$ are renamed by l'. When the SAT solver receives this input, it returns propositional models (atoms and atoms with prime apostrophe) defining all possible transitions to the next state that do not contradict (yet) $\neg prop$. To get each of the next states, the next-state rule is applied once the variables are renamed back from ℓ' to $\circ\ell$.

In summary, the tableau deals with the temporal reasoning of the initial set of formulae. In contrast, the SAT solver deals with the quasi-propositional reasoning.

3.1.3 Algorithm

In this section the Algorithm `Momo_pltl` is presented (Algorithm 5). This algorithm is based on Algorithm 1. It includes the necessary adaptations for the optimizations presented in Section 3.1.2.

The first two if-elseif cases of the Algorithms 5 are the same as Algorithm 1: if Δ is not syntactically consistent the algorithm returns an empty model and $false$. If there is an ancestor that contains all the formulae of the current node and all eventualities in the path has been selected at least once, the algorithm returns a model and $true$.

Algorithm 5: `Momo_PLTL`
Input = Δ: set of formulae, `selectedFormula`: formula
Output = M: Model, b: boolean

if Incons(Δ) **then**
  $M, b := \emptyset$, *false*;
else if there exists Δ', an ancestor of Δ, such that $\Delta \subseteq \Delta'$ and all eventualities in the branch has been already selected **then**
  ▷ where *model* is the model constructed from the branch where Δ' and Δ are.
  $M, b :=$ *model*, *true*;
else if `is_pre-next_state(`Δ`)` **then**
  M, b `:= apply-SAT-Next-rule(`Δ`,selectedFormula);`
else
  `selectedFormula = select_formula(`Δ`);`
  if `selectedFormula` $\neq$ `F` **then**
    M, b `:= apply-`β^+`-rule(`Δ`,selectedFormula);`
  Let ψ be any temporal formula in Δ that is neither permanent nor quasi-propositional;
  if α`_is_applicable(`ψ`)` **then**
    M, b `:= apply-`α`-rule(`ψ`,`Δ`,selectedFormula);`
  else `//` β`_is_applicable(`ψ`)`
    M, b `:= apply-`β`-rule(`ψ`,`Δ`,selectedFormula);`
  end
end

Algorithm 6: `apply-SAT-Next-rule`
Input = Δ: set of formulae, `selectedFormula`: formula
Output = M: Model, b: boolean

Let P be the set of permanent formulae in Δ;
$\alpha =$ `PLTL2SAT(`$P \cup$ {`quasi-propositional formulae in` Δ}`);`
is_sat`,`B `:= Solver(`α`);`
is_closed `:=` *true*`;`
while *is_sat* and *is_closed* **do**
  Δ `= SAT2PLTL(`Δ`,`B`);`
  Δ' `=` $P \cup$ `apply-Next-rule(`$\Delta \setminus P$`);`
  model`,`*is_closed* `:= Momo_PLTL(`Δ'`, selectedFormula);`
  if *is_closed* **then**
    $\alpha := \alpha \wedge \neg B$;
    is_sat`,`B `:= Solver(`α`);`
  end
end
if not *is_closed* **then**
  $M, b :=$ *model*, *is_closed*;
else
  $M, b := \emptyset$, *false*;
end

In Momo_pltl the functions apply-α-rule, apply-β-rule and apply-β^+-rule are the same as in Algorithm 3, 4, and 2. However, unlike Tableau (Algorithm 1), the α and β rules apply only to temporal formulae that are neither permanent nor quasi-propositional. These types of formulae are treated in pre-next states with the SAT solver support.

The function apply-SAT-Next-rule (Algorithm 6) removes the $\Box$ operator from the permanent formulae in Δ and transforms them into quasi-propositional formulae. A boolean formula α is then constructed from these quasi-propositional formulae in Δ. The function PLTL2SAT performs this task (line 2). Next, the solver is used to check whether α is satisfiable or not. In the affirmative case, a propositional model B is obtained (line 3), whose information is added to Δ when back translation SAT2PLTL is performed (line 6). The next-state rule ($\circ$) is then applied to Δ excluding its permanent formulae (line 7). Consequently, a new state with label Δ' is obtained where the previous permanent formulae are included again.

The process goes on by a recursive call (line 8). As soon as *is_closed* gets the value $false$ in this recursive call, the iteration stops and M is returned as a model of Δ. Otherwise, the iteration stops when the query to the solver about α does not give new propositional models. Note that, at each iteration step, we conjunctively add to α the negation of the previous propositional model (line 10).

Example 6 illustrates the use of the SAT solver.

Example 6 (Use of the SAT solver)**.** *Suppose that* Momo_pltl *receives as input the specification of Example 5 plus the formula* $\Diamond(\circ b \wedge \Box\neg a)$. *Then, the root node* (n_1) *contains* $Init = \{a\}, P$, *and* $\Diamond\Box(\circ b \wedge \neg a)$, *where* P *is the set of permanent formulae* $Invariant \cup TR$. *Therefore,* $P = \{\Box(a \rightarrow \circ b), \Box(b \rightarrow \circ b), \Box(a \rightarrow \neg b), \Box(b \rightarrow \neg a)\}$.

The tableaux construction is shown in Figure 3.4. The $(\Diamond)^+$ *rule is the first one to be applied creating two nodes. The left one* (n_2), *after the application of the* $(\wedge)$*-rule, produces the node* n_3, *which contains a new permanent formula* $\Box\neg a$ *and is a pre-next state node. The algorithm executes* apply-SAT-Next-rule *and the function* PLTL2SAT *transforms* $\{a, P, \circ b, \Box\neg a\}$ *into a propositional set, which is passed to the SAT solver. Its answer is empty (unsatisfiable) and the branch closes. The right node* (n_4) *is created using the negated context. This node is directly a pre-next state and the* apply-SAT-Next-rule *function is executed. This time the input of the SAT solver is the translation of the set* $\{P, a\}$ *to a propositional formula. The SAT solver returns the model* $\{a \leftarrow \mathbf{T}, b \leftarrow \mathbf{F}, b' \leftarrow \mathbf{T}\}$. *The model information and its translation back provide a new node* (n_5), *whose label is* $\{P, a, \neg b, \circ b, \circ(\neg a \mathcal{U}(\circ b \wedge \Box\neg a)\}$. *The tableau applies the next-state rule and makes a recursive call to repeat the whole process. At a given time, it reaches a new pre-next state node* (n_8) *and* apply-SAT-Next-rule *is executed. The SAT solver returns a model and the pre-next state node* n_9 *is obtained. When the next-state rule is applied to it,* Momo_pltl *finds out a cycle and the process ends up with an open branch.*

The PLTL*-structure provided by this open branch is* $\{a\}, \langle\{b\}\rangle^w$.

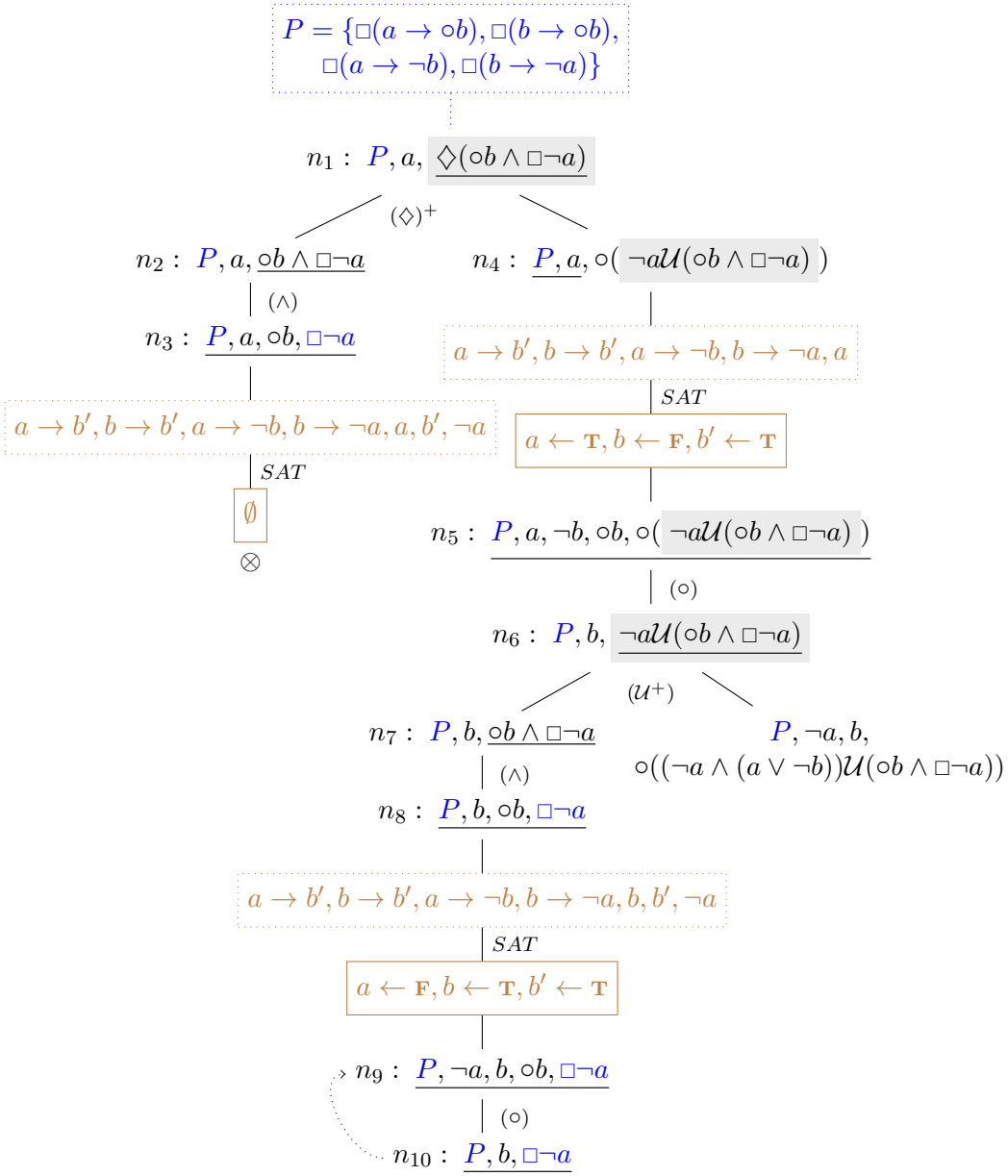


Figure 3.4: Example of SAT solver use.

3.2 Running Examples

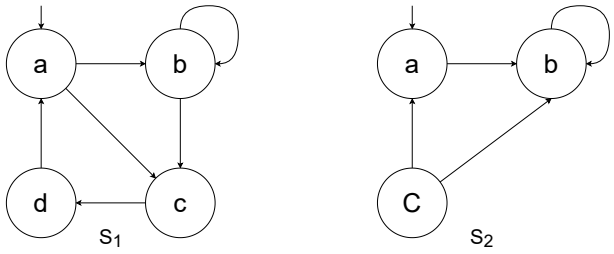


Figure 3.5: Two transition Systems S_1 and S_2.

This section shows how the algorithm works by means of two examples of checking where a transition system (that represents a directed graph) contains a cycle starting at some particular vertex. For example, in Figure 3.5, the left graph S_1 contains a cycle where the node a is reachable from a, while the right one, S_2, does not contain any cycle where a is reachable.

In order to compare the representation of the systems and the use of the model checker, we present the solution for both systems in NuSMV in parallel with our solution. First, we introduce the codification of the two problem-instances in NuSMV. There exist many ways of encoding directed graphs onto transition systems depending on the nature of the problem to be solved [10]. However, all of them simulate the adjacency matrix for a given graph. Essentially, states of the transition system are the vertices of the graph and the transitions themselves are the edges of the graph.

The transition system with cycles

In the case of S_1 the adjacency matrix is as follows.

$$\begin{aligned}\mathrm{TR}_1 = \{\ & \Box(a \rightarrow ((\circ\neg a \wedge \circ b \wedge \circ\neg c \wedge \circ\neg d)\ \vee\ (\circ\neg a \wedge \circ\neg b \wedge \circ\neg c \wedge \circ d))),\\ & \Box(b \rightarrow ((\circ\neg a \wedge \circ b \wedge \circ\neg c \wedge \circ\neg d)\ \vee\ (\circ\neg a \wedge \circ\neg b \wedge \circ\neg c \wedge \circ d))),\\ & \Box(c \rightarrow (\circ a \wedge \circ\neg b \wedge \circ\neg c \wedge \circ\neg d)),\\ & \Box(d \rightarrow (\circ\neg a \wedge \circ\neg b \wedge \circ c \wedge \circ\neg d))\ \}\end{aligned}$$

To make the transition system above more realistic (in terms of directed graphs behavior), we add an *Invariant* formula that forces to stay on a single vertex each time:

$$\Box(\,((a \rightarrow (\neg b \wedge \neg c \wedge \neg d))\ \wedge\ (b \rightarrow (\neg a \wedge \neg c \wedge \neg d))\ \wedge\ (c \rightarrow (\neg a \wedge \neg b \wedge \neg d))\ \wedge\ (d \rightarrow (\neg a \wedge \neg b \wedge \neg c)))\,)$$

Finally, the initial vertex in our example is a. Hence, $Init = \{a\}$.

Figure 3.6 shows the module for S_1 in NuSMV as well as the temporal formula that states 'Vertex a is unreachable in the future'. That formula is written at the end of the module, in the part starting with the reserved word LTLSPEC. The corresponding temporal formula in our syntax is

$$\circ\Box\neg a \tag{3.1}$$

The formulae $Init$, TR_1, and $Invariant$ are written in the NuSMV file under the INIT, TRANS, and INVAR keywords respectively. The syntax of boolean connectives and temporal operators in NuSMV is as follows.

- The propositional connectives are & for $\wedge$, | for $\vee$, ! for $\neg$.
- The temporal operators are X for $\circ$, G for $\Box$, F for $\Diamond$, and U for $\mathcal{U}$.

NuSMV system negates the LTLSPEC formula and tries to construct a model. It successes and finds out a cycle in the graph. Namely, it returns the model of Figure 3.6, which is the PLTL-structure $\langle\{a\},\{d\},\{c\}\rangle^w$.

S1.smv

```
MODULE main

VAR
a:boolean; b:boolean; c:boolean; d:boolean;

INVAR
(a -> (!b & !c & !d)) & (b -> (!a & !c & !d)) &
(c -> (!a & !b & !d)) & (d -> (!a & !b & !c))

INIT
a = TRUE

TRANS
(a -> ((next(!a) & next(b) & next(!c) & next(!d)) |
   (next(!a) & next(!b) & next(!c) & next(d))))
&
(b -> ((next(!a) & next(b) & next(!c) & next(!d)) |
   (next(!a) & next(!b) & next(!c) & next(d))))
&
(c -> (next(a) & next(!b) & next(!c) & next(!d)))
&
(d -> (next(!a) & next(!b) & next(c) & next(!d)))

LTLSPEC X(G(!a))
```

```
-- specification  X ( G !a)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 1.1 <-
    a = TRUE
    b = FALSE
    c = FALSE
    d = FALSE
  -> State: 1.2 <-
    a = FALSE
    d = TRUE
  -> State: 1.3 <-
    c = TRUE
    d = FALSE
  -> State: 1.4 <-
    a = TRUE
    c = FALSE
```

Figure 3.6: NuSMV encoding of S_1 and the output provided by NuSMV

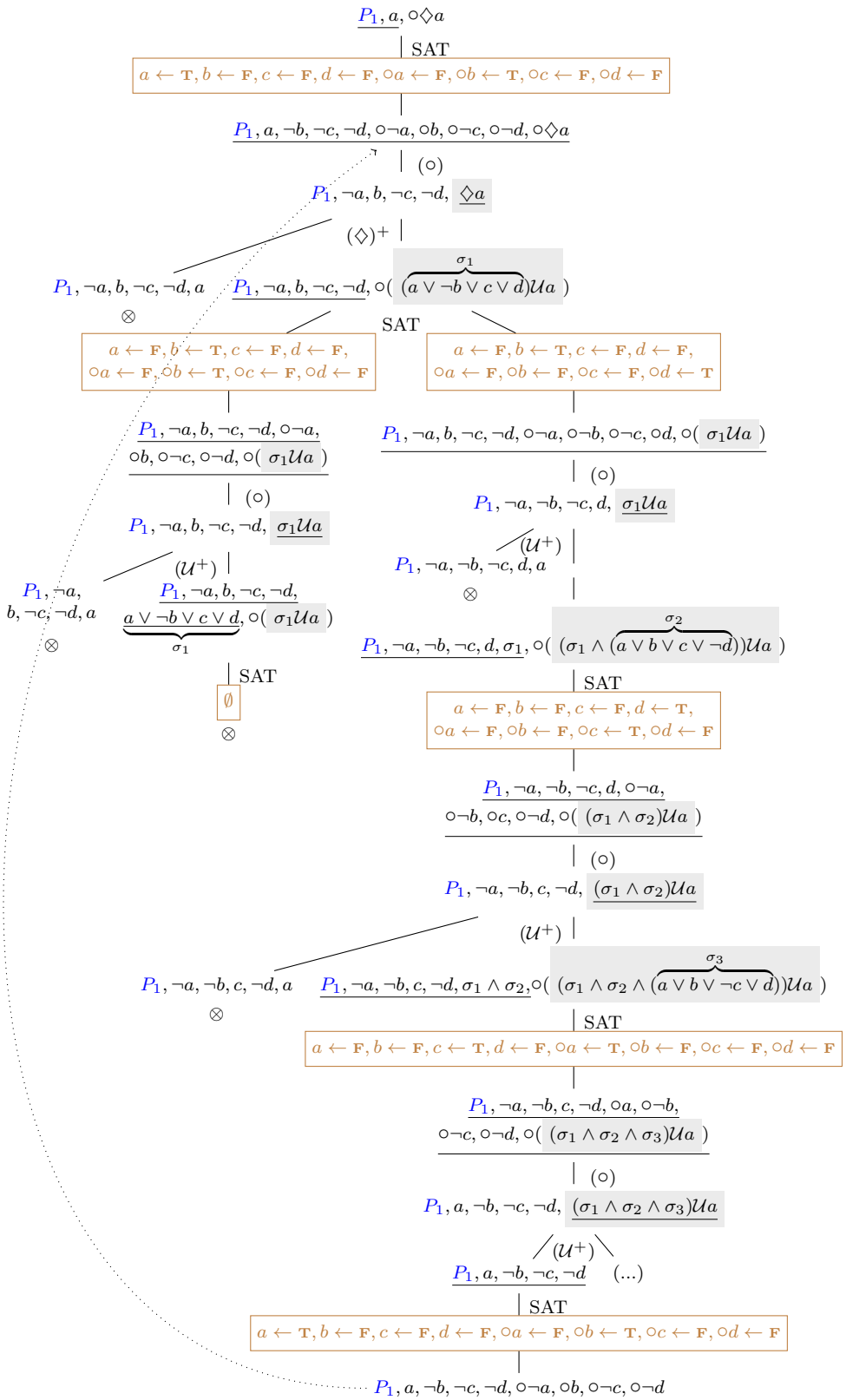


Figure 3.7: Open Tableau for S_1

Figure 3.7 is the tableau constructed by our algorithm. Remember that the selected eventualities are in gray boxes. The root node contains the set $\{a, \circ\Diamond a\}$. The formula $\circ\Diamond a$ is the negation of formula (3.1). The root also contains the set of permanent formulae $P_1 = Invariant \cup \mathrm{TR}_1$. Since the root node is a pre-next state, the tableau calls to the SAT solver, which receives as input a in conjunction with the translation of P_1 into this set of formulae.

$$\begin{aligned}\mathrm{PLTL2SAT}(P_1) = \{\ & a \rightarrow ((\neg a' \wedge b' \wedge \neg c' \wedge \neg d') \ \vee\ (\neg a' \wedge \neg b' \wedge \neg c' \wedge d))),\\ & b \rightarrow ((\neg a' \wedge b' \wedge \neg c' \wedge \neg d') \ \vee\ (\neg a' \wedge \neg b' \wedge \neg c' \wedge d'))),\\ & c \rightarrow (a' \wedge \neg b' \wedge \neg c' \wedge \neg d')),\\ & d \rightarrow (\neg a' \wedge \neg b' \wedge c' \wedge \neg d')),\\ & a \rightarrow (\neg b \wedge \neg c \wedge \neg d), b \rightarrow (\neg a \wedge \neg c \wedge \neg d)\ ,\\ & c \rightarrow (\neg a \wedge \neg b \wedge \neg d), d \rightarrow (\neg a \wedge \neg b \wedge \neg c)\ \}\end{aligned}$$

The algorithm stops when the first open branch (in Figure 3.7, the right most one) is found. Then, it returns the same PLTL-structure as the one built by NuSMV.

The transition system without cycles

S2.smv

```
MODULE main

VAR
a:boolean;  b:boolean;  c:boolean;

INVAR
(a -> !b & !c) & (b -> !a & !c) & (c -> !a & !b)

INIT
a = TRUE

TRANS
((a -> (next(!a) & next(b) & next(!c)))
&
(b -> (next(!a) & next(b) & next(!c)))
&
(c -> ((next(a) & next(!b) & next(!c)) |
       (next(!a) & next(b) & next(!c)))))

LTLSPEC X(G(!a))
-------------------------------------------------------------------------------
-- specification  X ( G !a)  is true
```

Figure 3.8: NuSMV encoding of S_2

The graph S_2 of Figure 3.5 can be represented with $Init = \{a\}$,

$$Invariant = \Box(\ (a \rightarrow (\neg b \wedge \neg c)) \wedge (b \rightarrow (\neg a \wedge \neg c)) \wedge (c \rightarrow (\neg a \wedge \neg b))\ )$$

$$\mathrm{TR}_2 = \{\ \Box(a \rightarrow (\circ\neg a \wedge \circ b \wedge \circ\neg c)),$$

$$\Box(b \rightarrow (\circ\neg a \wedge \circ b \wedge \circ\neg c)),$$
$$\Box(c \rightarrow ((\circ a \wedge \circ\neg b \wedge \circ\neg c) \ \vee \ (\circ\neg a \wedge \circ b \wedge \circ\neg c))) \ \}$$

So, the set of permanent formulae is $\mathrm{TR}_2 \cup Invariant$. We denote it as P_2. The translation of P_2 into boolean formulae is performed by omitting the $\Box$ operator and renaming $\circ a, \circ b$, and $\circ c$ with a', b', and c' respectively.

Figure 3.8 shows the module for S_2 in NuSMV. When NuSMV system negates the LTLSPEC formula $\circ\Box\neg a$, it is not able to find a model and returns the following answer: `specification X(G!a) is true`. Therefore, there is no way to reach vertex a once we move to any other vertex.

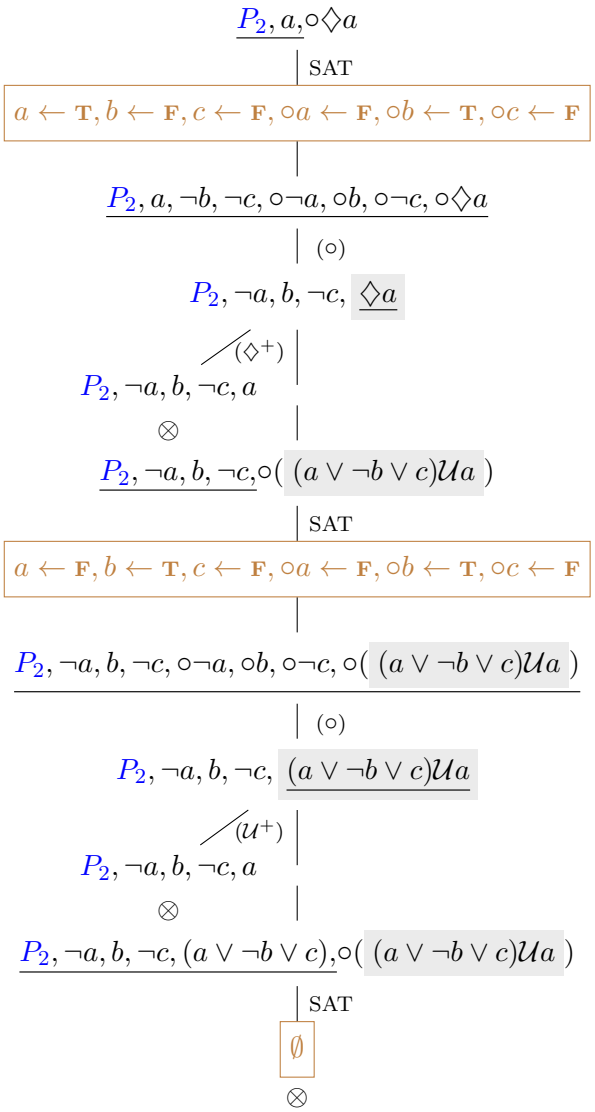


Figure 3.9: Closed Tableau for S_2

Figure 3.9 is the tableau constructed by our algorithm. The root node is the set of permanent formulae P_2 together with $\{a, \circ\Diamond a\}$. All its branches are closed as expected.

Hence, the constructed tableau proves the fact that 'vertex a is unreachable in the future'.

3.3 Generation of a Certification

The specialization of the tableaux method for model checking explained in Section 3.1 has the same functionalities as other model checkers (as seen in Section 3.2). If the transition system satisfies the property, there is a confirmation of it (closed tableau). On the other hand, if the system does not satisfy the property, there is a counterexample (the model).

When the properties are not fulfilled, it is easy to navigate over the transition system S to follow the counterexample and the errors may be detected. In contrast, in the case where the property being checked is fulfilled, the user has only a confirmation. In that case it is assumed that the implementation of the model checker has not failed and that the representation of the transition system is correct but there is no evidence of that. Example 7 illustrates an error in system modelling.

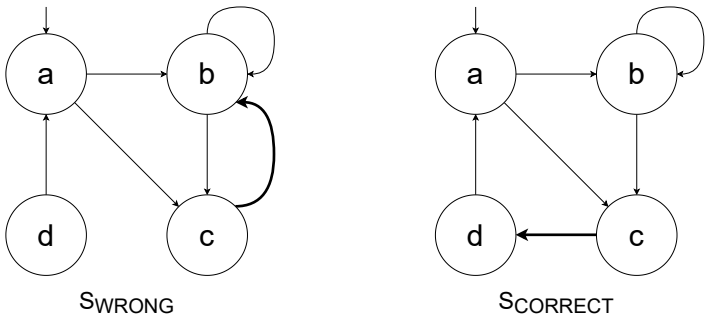


Figure 3.10: Example of how the approach can help to detect errors.

Example 7. *Figure 3.10 shows two transition systems. Let us suppose that we pretend to represent system $S_{CORRECT}$. Due to a specification error, we write S_{WRONG}. That is, we mistakenly specify that there is a transition 'from c to b' instead of the intended one: 'from c to d'. The resulting specification is:*

$$
\begin{aligned}
Init &= \{\ a,\ \neg b,\ \neg c,\ \neg d\ \} \\
TR &= \{\ \Box(a \rightarrow (\circ\neg a \wedge \circ b \wedge \circ\neg c \wedge \circ\neg d)), \\
&\qquad \Box(b \rightarrow ((\circ\neg a \wedge \circ b \wedge \circ\neg c \wedge \circ\neg d)\ \vee\ (\circ\neg a \wedge \circ\neg b \wedge \circ c \wedge \circ\neg d))), \\
&\qquad \Box(c \rightarrow (\circ\neg a \wedge \underline{\circ b} \wedge \circ\neg c \wedge \underline{\circ\neg d})), \\
&\qquad \Box(d \rightarrow (\circ a \wedge \circ\neg b \wedge \circ\neg c \wedge \circ\neg d))\ \}
\end{aligned}
$$

Note that S_{WRONG} only consists of $Init$ and TR. Now, suppose that we want to check if S satisfies the PLTL *formula $\circ\Box\neg a$. Its negation in* NNF *is $\circ\Diamond a$. Then, we call* `Momo_pltl` *with the following input - the label of the initial node - $Init \cup P \cup \{\circ\Diamond a\}$. Here P is exactly the set TR. Thinking on the system $S_{CORRECT}$, we expect to get a model like $\langle\{a\},\{b\},\{c\},\{d\}\rangle^w$ or $\langle\{a\},\{c\},\{d\}\rangle^w$ as a counterexample, since they are models of $\circ\Diamond a$ in the correct system $S_{CORRECT}$. However, $S_{WRONG} \cup \{\circ\Diamond a\}$ is unsatisfiable and our model checker generates a closed tableau for it.*

Closed tables are refutation proofs that, in our setting, explain why a transition system satisfies a particular property. However, the explanation provided by the proof can be very difficult to follow. To solve this, we define a 'big-step' version of the closed tableaux. This big-step version enables an easy follow-up of the run across the transitions systems because it only shows the pre-next states and the next states in the branches. Figure 3.11 depicts the big-step version of the closed tableau for $S_{WRONG} \cup \{\circ\Diamond a\}$.

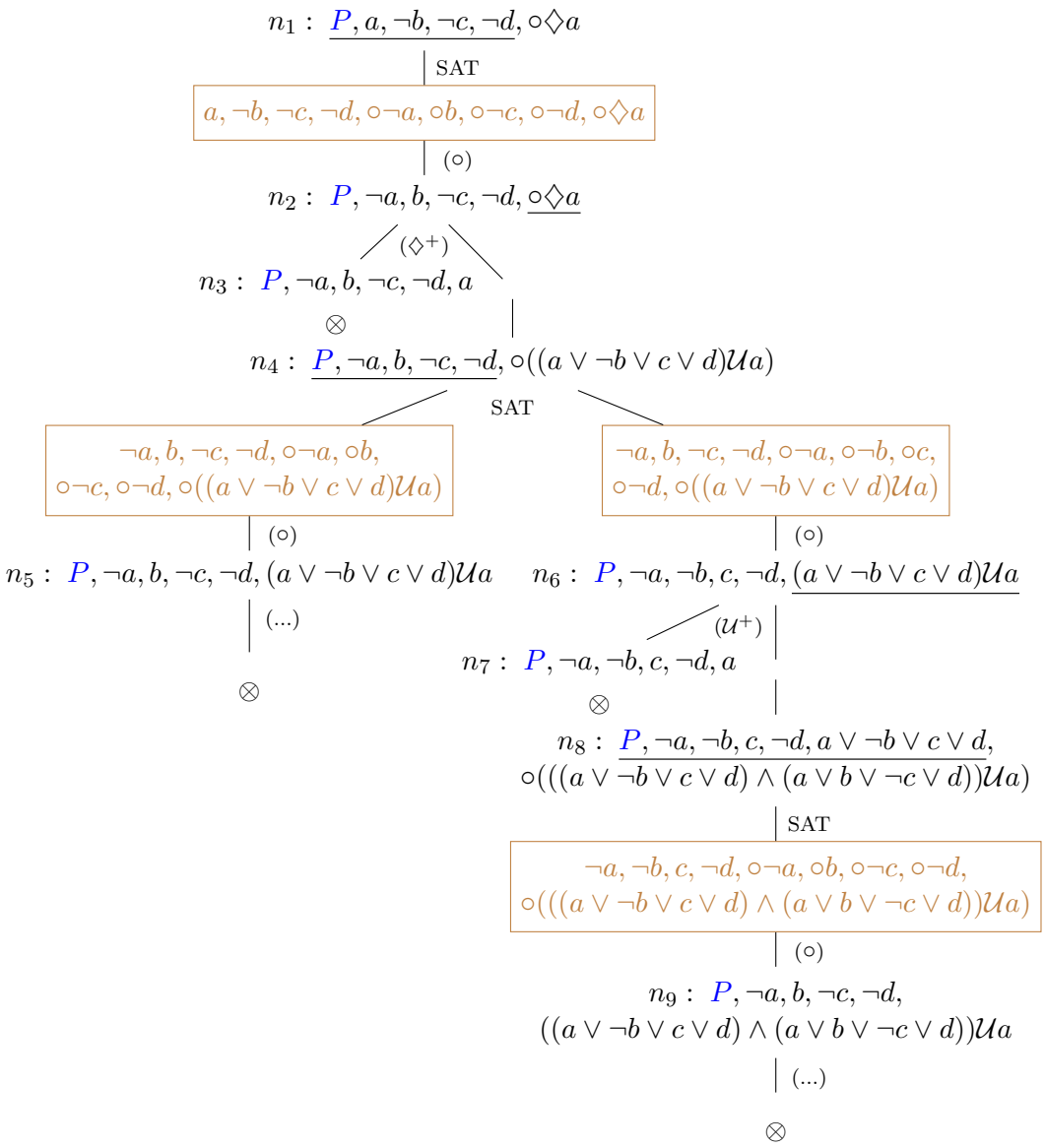


Figure 3.11: A big-step representation of the closed tableau for $S_{WRONG} \cup \{\circ\Diamond a\}$.

The root of the tableau in Figure 3.11 contains the $Init = \{a, \neg b, \neg c, \neg d\}$, the negated property, $\circ\Diamond a$, and P, which is exactly TR. The tableau evolves by the application of rules until a pre-next state is reached. It is now, when the solver gives us, one by one, the different models that satisfy the (translated) permanent and quasi-

propositional formulae. Each model is transformed back into a PLTL formulae and the next-state rule is applied. The tableau continues to work and if it closes, a different model is supplied by the solver, when available. For example, for node n_4 in the tableau, two different propositional models are provided. First, the solver returns a model in which only b' (that is, $\circ b$) is true. It applies the next rule and all the subsequent branches close. Returning to node n_4, the solver finds a new propositional model where c' (i.e. $\circ c$) is true. However, this model also produces a closed branch and no other model is available. Therefore, the tableau is closed.

If users analyse the tableau in Figure 3.11, they can see the reason why the property is not satisfied: the trace provided by nodes n_1, n_2, n_6 and n_9 describes the transitions (omitting the negated literals) $a \to b \to c \to b$. By comparing that trace with the intended specification $S_{CORRECT}$, the user will be able to find the error.

3.3.1 One Step and Big Step Proof in Isabelle

In case of closed tableaux, a certificate guaranteeing the unsatisfiability of the initial set of formulae can be obtained. This can be done using the sequent calculus presented in Section 2.2. Example 8 shows the Isabelle representation of the system S_{WRONG}. Then the negated property is written and using Isabelle the user can navigate through the proof until the root derives $\mathbf{F}$.

Example 8. *The specification S_{WRONG} on the Figure 3.10 is defined as the list TR = [$T1$, $T2$, $T3$, $T4$] of* PLTL *formulae.*

$$
\begin{array}{rcl}
T1 & = & \Box(\,(V\ a)\ .\longrightarrow\ (\ \circ(V\ b)\ .\wedge\ \circ(\ .\neg\ (V\ a))\ .\wedge\ \circ(\ .\neg\ (V\ c))\ .\wedge\ \circ(\ .\neg\ (V\ d)))\ ) \\
T2 & = & \Box(\,(V\ b)\ .\longrightarrow\ (\ (\circ(V\ b)\ .\wedge\ \circ(\ .\neg\ (V\ a))\ .\wedge\ \circ(\ .\neg\ (V\ c))\ .\wedge\ \circ(\ .\neg\ (V\ d)))\ .\vee \\
 & & \qquad (\circ(V\ c)\ .\wedge\ \circ(\ .\neg\ (V\ a))\ .\wedge\ \circ(\ .\neg\ (V\ b))\ .\wedge\ \circ(\ .\neg\ (V\ d)))\ ) \\
T3 & = & \Box(\,(V\ c)\ .\longrightarrow\ (\ \circ(V\ b)\ .\wedge\ \circ(\ .\neg\ (V\ a))\ .\wedge\ \circ(\ .\neg\ (V\ c))\ .\wedge\ \circ(\ .\neg\ (V\ d)))\ ) \\
T4 & = & \Box(\,(V\ d)\ .\longrightarrow\ (\ \circ(V\ a)\ .\wedge\ \circ(\ .\neg\ (V\ b))\ .\wedge\ \circ(\ .\neg\ (V\ c))\ .\wedge\ \circ(\neg\ (V\ d)))\ )
\end{array}
$$

along with $Init = (V\ a)\ .\wedge\ .\neg\ (V\ b)\ .\wedge\ .\neg\ (V\ c)\ .\wedge\ .\neg\ (V\ d)$.

We have implemented, with the help of the Eisbach tools [70], two prototypes of automatic solvers: one big-step and one small-step.

These two solvers print into a text file the 'apply' instructions of the Isabelle proof, at the same time that they prove the lemma runningExample_bigStep_proof. The proof is given in Figure 3.12. It proves the property S @ $[\circ\Diamond(V\ a)] \vdash \mathbf{F}$ by a list of 'apply' instructions. S is the specification S_{WRONG}, defined as the list of `[T1, T2, T3, T4] @ [Init]`. The proof can be found in file `ProofGeneration.thy` in `http://github.com/alexlesaka/OnePassTableau` and it is a big-step proof that enables the user to check whether S satisfies the property $\circ\Box\neg a$. That means, checking the unsatisfiability of $S \cup \{\circ\Diamond a\}$, in fact checking the derivability of the sequent $S, \circ\Diamond a \vdash \mathbf{F}$.

The (proof) method one_step_solver systematically applies the TTC_Calculus rules until it obtains a set of non-proved subgoals with antecedents exclusively formed by elementary formulae. Hence, the rule TTC_Next_State is applied to all subgoals, which depict (as 'Proof state') all subgoals related to a possible next state of the system, that is, after all possible transitions from the current state. For clarity, we fold the transition

proofGeneration.thy

```
lemma runningExample_bigStep_proof: "S @ [()(<>(V "a"))]  |-  F"
 1. apply (rule TTC_Interchange, simp add: S_def TR_def T1_def T2_def T3_def T4_def Init_def)
 2. apply one_step_solver
 3. apply (all <rule TTC_Next_State>; simp)
 4. apply (fold T1_def T2_def T3_def T4_def)
 5. apply (simp add: T1_def T2_def T3_def T4_def Init_def)
 6. apply one_step_solver
 7. apply (all <rule TTC_Next_State>; simp)
 8. apply (fold T1_def T2_def T3_def T4_def)
 9. apply (simp add: T1_def T2_def T3_def T4_def Init_def)
10. apply one_step_solver
11. apply (all <rule TTC_Next_State>; simp)
12. apply (simp add: T1_def T2_def T3_def T4_def Init_def)
13. apply one_step_solver
14. apply (all <rule TTC_Next_State>; simp)
15. apply (fold T1_def T2_def T3_def T4_def)
16. apply (simp add: T1_def T2_def T3_def T4_def Init_def)
17. apply one_step_solver
18.   done
```

Figure 3.12: The big-step lemma proof.

relations to their names. Hence, after the 'apply' in line 3 (Figure 3.12), the user can see that there is only one subgoal:

$$[\Diamond(V\ a),\ T4,\ T2,\ T3,\ T1,\ .\neg(V\ d),\ .\neg(V\ c),\ .\neg(V\ a),\ (V\ b),\ ] \vdash \mathbf{F}$$

This means the only state that is reachable from the initial one is the state that satisfies b. That corresponds to the node n_2 in Figure 3.11. After the 'apply', line 7, there are two subgoals that correspond, respectively, to the nodes n_5 and n_6 in Figure 3.11. Thus, from the state, which satisfies exactly b, the system can reach either a state that again satisfies b or a state satisfying exactly c. In both cases, the property to check is $(\neg b \vee a \vee c \vee d)\ \mathcal{U}\ a$. The goal corresponding to node n_5 is proved, whereas the 'apply' in line 11, (Figure 3.12) generates the subgoal for node n_6. This subgoal corresponds to the transition from the state that satisfies b to the state that satisfies c. The property to check at this state is $(\neg b \vee a \vee c \vee d)\ \mathcal{U}\ a$. After the 'apply' in line 14, the subgoal which corresponds to the node n_9 in Figure 3.11 is reached; here the property to check is $((\neg b \vee a \vee c \vee d) \wedge (\neg c \vee a \vee b \vee d))\ \mathcal{U}\ a$. The proof of this subgoal completes the proof of the lemma, that has explored the two possible runs $\{a\}, \langle\{b\}\rangle^w$ and $\{a\}, \{b\}, \{c\}, \langle\{b\}\rangle^w$, where the property $\circ\Diamond a$ is not satisfied. This shows the inability of the transition system to reach the state that satisfies d, which reveals an error in the specification that makes unreachable the state that satisfies a.

The 'apply(fold ...)' instructions in Figure 3.12 are only to show, in subgoals, the names ($T1$, $T2$, $T3$, and $T4$) instead of the corresponding PLTL formulae defining the transitions, for brevity and clarity. After that, we must use 'apply(simp add: ...)' to enable the application of the rules.

We have also implemented a method one_step_solver_print which prints into a file of text all applications of the TTC rules that are hidden in the big-step proof. Indeed, it is a printing version of the method one_step_solver. Calling one_step_solver_print,

instead of one_step_solver, we can generate a small-step proof (see lemma runningExample_smallStep_proof) for the user wishing to check the Isabelle's subgoals step by step. This proof is the result of the substitution, in the big-step proof, of each of the five occurrences of apply one_step_solver by the list of 'apply' instructions of TTC_Calculus rules. The method one_step_solver_print prints such a list, in a text file, while it is solving the goal.

Let us conclude by summing up the information contained in Figure 3.11. If one follows the big-step tableau in the figure, you can see how from the state $a, \neg b, \neg c, \neg d$ you go to the state $\neg a, b, \neg c, \neg d$. At that point there are two possible next states. In the first case, the tableau closes. On the other hand, in the second one, it goes to the state $\neg a, \neg b, c, \neg d$. At this point the new state is $\neg a, b, \neg c, \neg d$. Following the transition system $S_{CORRECT}$, the user finds inconsistencies in the modelling. In a large system it is impossible to go through absolutely all the traces, but in case of doubts about specific parts, a tableau helps to visualise them. The reason is that a tableau corresponds exactly to the transitions of the model and this makes it easier to navigate through the traces.

4. BRANCHING TEMPORAL LOGICS: CTL AND ECTL

This chapter continues the development of context-based tableau methods for temporal logics [41, 16, 3, 4] with the intention of solving both the satisfiability problem and the model checking problem. With respect the latter problem, our aim is to extend the CMC approach, which we introduced for PLTL, to be 'generic' for the whole variety of branching-time logics. As we have already explained, the crucial aspect of our approach is based on the development of deductive techniques that apply the same reasoning mechanism for both the tasks of generating counterexamples and providing formal proofs.

We focus here on the two logics CTL and ECTL. The CTL (resp. ECTL) satisfiability problem cannot be reduced to the CTL (resp. ECTL) model checking problem. Hence, model checking algorithms for CTL (resp. ECTL) cannot be adapted to decide CTL (resp. ECTL) satisfiability. However, any satisfiability decision procedure for any logic (in particular for CTL and ECTL) can perform the model checking task. This is the reason why the tableaux we are going to present in this chapter refer to the satisfiability problem for CTL and ECTL. Their specialisation for the model checking case would be similar to the one introduced for PLTL in Chapter 3.

A context-based tableau method for the branching-time logic $\mathsf{ECTL}^{\#}$ (which extends CTL and ECTL) was developed in [16]. This logic allows to reason about fairness constraints that use the temporal operator 'until'. These tableaux for $\mathsf{ECTL}^{\#}$ has rather complex rules and they have a distinguished (and unavoidable) feature: the utilisation of two types of contexts. The first type of context, called 'outer' (similar to PLTL), is a collection of state formulae and the second, called 'inner' context, is a collection of path formulae. The context-based tableaux for CTL and ECTL only need the 'outer' context and, therefore, it seems reasonable to develop simpler specific methods for CTL and ECTL instead of applying the general tableau method for $\mathsf{ECTL}^{\#}$, which would be too costly and 'non-intuitive'. So, we introduce here new context-based tableau methods for CTL and ECTL along with their dual sequent calculi. We prove soundness and completeness of the methods, and illustrate how they provide formal proofs and models. All these bring us one step closer to formulating a 'generic' approach covering all CTL-type branching-time logics.

This chapter starts with Section 4.1 showing an overview of CTL-type Branching-time logics. In Section 4.2 those logics are restricted to CTL and ECTL. In Section 4.3 the new dual tableau and sequent method for CTL is presented. Finally, Section 4.4 extends the dual method to ECTL.

4.1 Overview of CTL-type Branching-time Logics

The hierarchy of CTL-type family of Branching-time logics (BTL) is defined by releasing restrictions on the concatenations of temporal operators and paths quantifiers which define classes of admissible state formulae distinguished for these logics. As in CTL [25] every temporal operator must be preceded by a path quantifier, this logic cannot express fairness which requires at least the concatenation of $\Box$ and $\diamondsuit$. These are tackled by ECTL [34] which enables simple fairness constraints but not their Boolean combinations. ECTL$^+$ [35] further extends the expressiveness of ECTL allowing Boolean combinations of temporal operators and ECTL fairness constraints (but not permitting their nesting). The logic ECTL$^\#$ [16] extends ECTL$^+$ by allowing the combinations $\Box(A\mathcal{U}B)$ or $A\mathcal{U}\Box B$, referred to as modalities $\Box\mathcal{U}$ and $\mathcal{U}\Box$. The logic CTL*, often considered as the full branching-time logic overcomes all these restrictions on syntax allowing any arbitrary combinations of temporal operators and path quantifiers.

As all logics we are interested in are subsumed by CTL*, for the sake of generality, we first present the CTL* syntax (Definition 15) and then, by restricting it, derive the syntax for each of ECTL$^\#$, ECTL$^+$, ECTL and CTL.

Definition 15 (Syntax of CTL*)**.** *Given* Prop *is a fixed set of propositions, and* $p \in$ Prop*, we define sets of state* (σ) *and path* (π) CTL* *formulae over* Prop *as follows:*

$$\begin{array}{lll} \sigma & ::= & \mathbf{T} \mid p \mid \neg\sigma \mid \sigma_1 \wedge \sigma_2 \mid \mathsf{E}\pi \\ \pi_{\mathsf{CTL}^\star} & ::= & \sigma \mid \neg\pi \mid \pi_1 \wedge \pi_2 \mid \circ\pi \mid \pi_1 \mathcal{U} \pi_2 \end{array}$$

Two observations are needed here. Definition 15 above introduces the minimal CTL* grammar: we only use one path quantifier - E, and two temporal modalities - $\circ$ and $\mathcal{U}$. From this combination we can derive a richer syntax, which is often more appropriate to use when we speak about the intuitive interpretation of formal specification of systems: the other path quantifier - A and the remaining temporal operators ($\diamondsuit$, $\Box$ and $\mathcal{R}$). In particular, the 'falsehood' constant $\mathbf{F} \equiv \neg\mathbf{T}$ and the classical disjunction operator is defined as $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$. The 'for all paths' quantifier $\mathsf{A}\varphi \equiv \neg\mathsf{E}\neg\varphi$ and the remaining temporal operators are defined as follows: $\diamondsuit\varphi \equiv \mathbf{T}\mathcal{U}\varphi$, $\Box\varphi \equiv \neg\diamondsuit\neg\varphi$, and $\varphi_1\mathcal{R}\varphi_2 \equiv \neg(\neg\varphi_1\mathcal{U}\neg\varphi_2)$.

Second, observe that in Definition 15 for the set of path formulae, $\pi_{\mathsf{CTL}^\star}$, we deliberately used an index CTL* to indicate that this grammar introduces a set of path formulae specifically for CTL*. At the same time we did not use any index for the set of state formulae. This reflects the tradition in defining the grammar for BTL logics in a way that the grammar for the path formulae determines relevant changes in the grammar for the set of state formulae. For CTL*, 'no restrictions' on the construction of path formulae determine 'no restriction' on the construction of the arguments of the 'path' quantifiers. For each of CTL* sublogics that we will define later, CTL and ECTL, their specific restrictions on the construction of path formulae will determine relevant classes of their state formulae (however, the grammar scheme to generate state formulae remains as presented here for CTL* formulae).

For interpreting CTL* formulae, we invoke Kripke structures that are labelled directed graphs corresponding to Emerson's R-generable structures, i.e. the transition relation R is suffix, fusion and limit closed [32].

Definition 16 (Labelled Kripke Structure)**.** *A Kripke structure, $\mathcal{K}$, is given by a quadruple (S, R, I, L) where $S \neq \emptyset$ is a set of states, $R \subseteq S \times S$ is a total binary relation, called the transition relation, I is a finite non-empty set of initial states, and $L : S \to 2^{\mathsf{Prop}}$ is a labelling function.*

A *path* x through a Kripke structure $\mathcal{K}$ is an infinite sequence of states $s_i, s_{i+1}, s_{i+2} \ldots$ $(i \geq 0)$ such that $(s_j, s_{j+1}) \in R$ for any $j \geq i$. Note that the totality of the transition relation R causes the infinity of paths. It is possible that a path in a Kripke structure, from some point onward, contains a repetitive sequence of states - a cycle. Later, in Definition 21, we introduce the notion of a cyclic Kripke structure. A *fullpath* x through a Kripke structure $\mathcal{K}$ is an infinite sequence of states $s_0, s_1, s_2 \ldots$, where $s_0 \in I$. By $\mathsf{fullpaths}(\mathcal{K})$ we denote the set of all fullpaths in $\mathcal{K}$. Given a fullpath $x \in \mathsf{fullpaths}(\mathcal{K})$ such that $x = s_0, s_1 \ldots$, the state s_i $(0 \leq i)$ is denoted by $x(i)$.

Given a path $x = s_i, s_{i+1}, \ldots$ and $k \geq 0$, we denote by $x^{<k}$ a finite prefix of x of length k and by $x^{\geq k}$ we denote its infinite suffix starting at state s_{i+k}. Hence $x^{<k} = s_i, s_{i+1} \ldots, s_{i+k-1}$ and $x^{\geq k} = s_{i+k}, s_{i+k+1}, \ldots$. Note that, $x = x^{<k}, x^{\geq k}$. Note that, $x = x^{<k}, x^{\geq k}$ (which means that this path x starts with prefix $x^{<k}$ to the k-th state and then, from this k-th state, continues with the suffix $x^{\geq k}$). Given a Kripke structure $\mathcal{K} = (S, R, I, L)$ and a state $s \in S$, let $\mathcal{K}' = (S', R, \{s\}, L)$ be a Kripke structure obtained from $\mathcal{K}$ by restricting S to S' such that S' is the set of all states of S that are R-reachable from s, we will denote $\mathcal{K}'$ by $\mathcal{K} \restriction s$. Note that if $I \neq \{s\}$ then $\mathcal{K} \restriction s$ is a proper substructure of $\mathcal{K}$ with the unique initial state S. Intuitively, $\mathcal{K} \restriction s$ represents the behaviour of a system from state s 'forward'. The set I helps modelling systems whose initial state is not definitely determined, for each $s \in I$, $\mathcal{K} \restriction s$ represents the behaviour of the system for the initial state s.

For a given $x \in \mathsf{fullpaths}(\mathcal{K})$ such that $x = s_0, s_1 \ldots$, and given $i \geq 0$, $\mathcal{K} \restriction x(i)$ is the Kripke structure that allows us to express path's *fusion closure*: if $y \in \mathsf{fullpaths}(\mathcal{K} \restriction x(i))$ then $x^{\leq i-1}, y \in \mathsf{fullpaths}(\mathcal{K})$. a prefix $x^{\leq i-1}$ of path x with any fullpath of $\mathcal{K} \restriction x(i)$, would result in a fullpath of $\mathcal{K}$.

For the convenience of the subsequent presentation (as we will be presenting the context-based tableaux and the dual sequent calculi for the sublogics of $\mathsf{CTL}^\star$), in Definition 17 below we introduce the evaluation of $\mathsf{CTL}^\star$ state and path formulae constructed with the extended grammar - with both classical $\mathbf{F}$ and $\vee$, with both path quantifiers, and the full set of temporal operators - $\circ$, $\Diamond$, $\Box$, $\mathcal{U}$ and $\mathcal{R}$. In our formulation of the $\mathsf{CTL}^\star$ semantics below we will also label the conditions related to the evaluation of state $\mathsf{CTL}^\star$ formulae by 's' followed by the reference to the relevant constraint (for example, '$(s_\neg)$' labels the condition evaluating a state formula $\neg\sigma$ in some state). Similarly, we label the conditions related to the evaluation of path $\mathsf{CTL}^\star$ formulae by 'p' followed by the reference to the relevant constraint (for example, '$(p_\vee)$' labels the condition evaluating a path formula $\pi_1 \vee \pi_2$ along some path). Recall that any $\mathsf{CTL}^\star$ state formula is also a path formula and that any rule (p_σ) applies when the path formula is really an state formula.

Definition 17 (Models)**.** *Given the structure $\mathcal{K} = (S, R, I, L)$, the relation $\models$, which evaluates path formulae in a given path x and state formulae at the state index i of the given path x, is inductively defined as follows.*

$(s_{\mathbf{T}})$	$\mathcal{K}, x, i \models \mathbf{T}$		
$(s_{\mathbf{F}})$	$\mathcal{K}, x, i \not\models \mathbf{F}$		
(s_{prop})	$\mathcal{K}, x, i \models p$	*iff*	$p \in L(x(i))$.
$(s_{\neg})$	$\mathcal{K}, x, i \models \neg\sigma$	*iff*	$\mathcal{K}, x, i \models \sigma$ *does not hold.*
$(s_{\vee})$	$\mathcal{K}, x, i \models \sigma_1 \vee \sigma_2$	*iff*	$\mathcal{K}, x, i \models \sigma_1$ *or* $\mathcal{K}, x, i \models \sigma_2$.
$(s_{\wedge})$	$\mathcal{K}, x, i \models \sigma_1 \wedge \sigma_2$	*iff*	$\mathcal{K}, x, i \models \sigma_1$ *and* $\mathcal{K}, x, i \models \sigma_2$.
(s_{E})	$\mathcal{K}, x, i \models \mathsf{E}\pi$	*iff*	*there exists* $y \in \mathsf{fullpaths}(\mathcal{K} \restriction x(i))$ *such that* $\mathcal{K}, y \models \pi$.
(s_{A})	$\mathcal{K}, x, i \models \mathsf{A}\pi$	*iff*	$\mathcal{K}, y \models \pi$ *holds for all* $y \in \mathsf{fullpaths}(\mathcal{K} \restriction x(i))$.
(p_{σ})	$\mathcal{K}, x \models \sigma$	*iff*	$\mathcal{K}, x, 0 \models \sigma$.
$(p_{\neg})$	$\mathcal{K}, x \models \neg\pi$	*iff*	$\mathcal{K}, x \models \pi$ *does not hold.*
$(p_{\vee})$	$\mathcal{K}, x \models \pi_1 \vee \pi_2$	*iff*	$\mathcal{K}, x \models \pi_1$ *or* $\mathcal{K}, x \models \pi_2$.
$(p_{\wedge})$	$\mathcal{K}, x \models \pi_1 \wedge \pi_2$	*iff*	$\mathcal{K}, x \models \pi_1$ *and* $\mathcal{K}, x \models \pi_2$.
$(p_{\bigcirc})$	$\mathcal{K}, x \models \bigcirc\pi$	*iff*	$\mathcal{K}, x^{\geq 1} \models \pi$.
$(p_{\Diamond})$	$\mathcal{K}, x \models \Diamond\pi$	*iff*	*there exists* $j \geq 0$ *such that* $\mathcal{K}, x^{\geq j} \models \pi$.
$(p_{\mathcal{U}})$	$\mathcal{K}, x \models \pi_1 \mathcal{U} \pi_2$	*iff*	*there exists* $k \geq 0$ *such that* $\mathcal{K}, x^{\geq k} \models \pi_2$ *and* $\mathcal{K}, x^{\geq j} \models \pi_1$ *for all* $0 \leq j < k$.
$(p_{\Box})$	$\mathcal{K}, x \models \Box\pi$	*iff*	$\mathcal{K}, x^{\geq j} \models \pi$ *holds for all* $j \geq 0$.
$(p_{\mathcal{R}})$	$\mathcal{K}, x \models \pi_1 \mathcal{R} \pi_2$	*iff*	*either* $\mathcal{K}, x^{\geq k} \models \pi_2$ *holds for all* $k \geq 0$, *or there exists some* $k \geq 0$ *such that* $\mathcal{K}, x^{\geq k} \models \pi_1 \wedge \pi_2$ *and* $\mathcal{K}, x^{\geq j} \models \pi_2$ *for all* $0 \leq j \leq k$.

Given a Kripke structure $\mathcal{K} = (S, R, I, L)$. For a state formula φ, we say that $\mathcal{K} \models \varphi$ (in words, $\mathcal{K}$ models φ) if and only if $\mathcal{K} \restriction s_0 \models \varphi$ for all $s_0 \in I$. For a set of state formulae Σ, $\mathcal{K} \models \Sigma$ if and only if $\mathcal{K} \models \varphi$ holds for every $\varphi \in \Sigma$. $\mathsf{Mod}(\varphi)$ (resp. $\mathsf{Mod}(\Sigma)$) denotes the set of all models of φ (resp. Σ).

To illustrate the semantics, we present the Example 9.

Example 9. *We will consider the following* CTL* *formula*

$$\mathsf{A}\Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p) \tag{4.1}$$

To show that this formula does not have a model, let us reason by contradiction assuming that there exists a model, say $\mathcal{K}$, for this formula. We will show that this assumption leads us to a contradiction when trying to build such a model. To start with, we pick a state s_0 from the set of the initial states of $\mathcal{K}$ and assume that formula (4.1) is true at $\mathcal{K} \restriction s_0$. Note that $\mathsf{fullpaths}(\mathcal{K} \restriction s_0) \subseteq \mathsf{fullpaths}(\mathcal{K})$. This means that for any fullpath, $x \in \mathsf{fullpaths}(\mathcal{K})$, $\mathcal{K}, x \models \Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$. Now pick a path from $\mathsf{fullpaths}(\mathcal{K})$, say x_1. Among the states of x_1, let $x_1(i)$ $(i \geq 0)$ be the first state satisfying the condition $\mathcal{K}, x_1^{\geq i} \models \bigcirc p \wedge \mathsf{E}\bigcirc\neg p$. This state must exist following the CTL* *semantics. Therefore, both $\bigcirc p$ and $\mathsf{E}\bigcirc\neg p$ are satisfied along $x_1^{\geq i}$. This means that p itself is satisfied at the state $x_1(i+1)$, i.e. at the successor of $x_1(i)$ on the path x_1. Since $\mathcal{K}, x_1^{\geq i} \models \mathsf{E}\bigcirc\neg p$ there should be a path starting at state $x_1(i)$ such that $\bigcirc\neg p$ is satisfied along this path. As $\neg p$ can not be satisfied at the state $x_1(i+1) \in x_1^{\geq i}$ where p has been already satisfied, this new path, call it x_2, to satisfy $\bigcirc\neg p$ should differ from $x_1^{\geq i}$. Now we invoke the fusion*

closure property which we have already discussed in this section (after the Def 16). Due to the fusion closure property there is a fullpath in $\mathcal{K}$ with the prefix $x_1^{<i}$ and the suffix x_2, namely $y_1 = x_1^{<i}, x_2$. Since in the given formula (4.1), $\Diamond(\circ p \wedge \mathsf{E}\circ\neg p)$, is in the scope of the A path quantifier, any fullpath, hence, also y_1, must satisfy $\Diamond(\circ p \wedge \mathsf{E}\circ\neg p)$. Hence, we must have a state $x_2(j)$ such that $j > i$ and $\mathcal{K}, y_1^{\geq j} \models \circ p \wedge \mathsf{E}\circ\neg p$. Considering this state $x_2(j)$, we invoke the same reasoning as we applied to the state $x_1(i)$ evaluating $\mathsf{E}(\circ p \wedge \mathsf{E}\circ\neg p)$ on the path x_1. This leads us to the analogous conclusion, that there must be another path x_3 starting at $x_2(j)$ such that $\circ\neg p$ is true along it. Again, the path $y_2 = x_1^{<i}, x_2^{<j} \cdot x_3$ should satisfy the property $\mathsf{E}(\circ p \wedge \mathsf{E}\circ\neg p)$, hence we must have a state $x_3(k)$ such that $k > j$ and $\mathcal{K}, y_1^{\geq k} \models \circ p \wedge \mathsf{E}\circ\neg p$. Therefore (due to limit closure), there exists $y \in \mathsf{fullpaths}(\mathcal{K})$ formed by the finite prefixes $x_1^{<i}, x_2^{<j}, x_3^{<k} \ldots$, such that $\mathcal{K}, y \not\models \Diamond(\circ p \wedge \mathsf{E}\circ\neg p)$. So, our assumption on the satisfiability of (4.1) was wrong.

Note that limit closure of the underlying Kripke structures was important for the above proof. Without the limit closure, for example, for so called bundled structures [44], the situation would have been different as we would not be able to assemble this new path y.

The following Definition 18 introduces notions of satisfiability, validity and equivalence for $\mathsf{CTL}^\star$ formulae and generalises satisfiability and validity for the sets of $\mathsf{CTL}^\star$ formulae. These are based on the concept of Mod introduced in Def 17.

Definition 18 ($\mathsf{CTL}^\star$ Satisfiability, Validity and Logical Equivalence)**.**

- *A state formula φ is satisfiable (denoted $\mathsf{Sat}(\varphi)$) whenever $\mathsf{Mod}(\varphi) \neq \emptyset$, otherwise φ is unsatisfiable (denoted $\mathsf{UnSat}(\varphi)$).*
- *A state formula φ is valid whenever $\mathcal{K} \models \varphi$ for all $\mathcal{K}$.*
- *A set of state formulae Σ is satisfiable (denoted $\mathsf{Sat}(\Sigma)$) if $\mathsf{Mod}(\Sigma) \neq \emptyset$. Otherwise Σ is unsatisfiable (denoted $\mathsf{UnSat}(\Sigma)$).*
- *A set of state formulae Σ is valid whenever $\mathcal{K} \models \Sigma$ for all $\mathcal{K}$.*
- *State formulae φ and φ' are logically equivalent if $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\varphi')$ (denoted as $\varphi \equiv \varphi'$).*

For each of BTL logics - $\mathsf{ECTL}^\#$, ECTL^+, ECTL and CTL - which are sublogics of $\mathsf{CTL}^\star$, we define its syntax over a fixed set of propositions Prop, preserving the definition of state formulae from $\mathsf{CTL}^\star$ (Def. 15) and formulating in Definition 19 specific for these logics on the $\mathsf{CTL}^\star$ grammar that generate corresponding sets for path formulae. Note that similarly to the $\mathsf{CTL}^\star$ grammar, we also utilise here the minimal set of operators, for the sake of consistency of the presentation and its rigor.

Definition 19 (Path Formulae for $\mathsf{ECTL}^\#$, ECTL^+, ECTL and CTL)**.** *The classification of the different branching-time logics according to their path formulae is shown in the next table.*

Logic		*Inherited*		*Characteristic*
$\pi_{\mathsf{ECTL}^{\#}}$	$::=$	$\pi_{\mathsf{ECTL}^{+}}$	$\mid$	$\sigma_1 \mathcal{U} (\sigma_2 \wedge (\mathbf{T} \mathcal{U} \sigma_3)) \mid$
				$\neg(\mathbf{T}\mathcal{U}\neg(\mathbf{T}\mathcal{U}(\neg\sigma_1 \wedge (\mathbf{T}\mathcal{U}\neg\sigma_2)))) \mid$
				$\sigma_1 \mathcal{U} \neg(\mathbf{T}\mathcal{U}\neg\sigma_2) \mid \neg(\mathbf{T}\mathcal{U}\neg(\sigma_1\mathcal{U}\sigma_2))$
$\pi_{\mathsf{ECTL}^{+}}$	$::=$	π_{ECTL}	$\mid$	$\pi_1 \wedge \pi_2$
π_{ECTL}	$::=$	π_{CTL}	$\mid$	$\neg(\mathbf{T}\mathcal{U}\neg(\mathbf{T}\mathcal{U}\neg\sigma)) \mid \mathbf{T}\mathcal{U}\neg(\mathbf{T}\mathcal{U}\neg\sigma)$
π_{CTL}	$::=$			$\sigma \mid \neg\pi \mid \circ\sigma \mid \sigma_1 \mathcal{U} \sigma_2.$

Our aim is to highlight different kinds of path formulae that are generated in each sublogic of $\mathsf{CTL}^{\star}$ and those characteristic to it. For example, for ECTL, the characteristic path formulae are those expressing linear-time fairness - $\neg(\mathbf{T}\mathcal{U}\neg(\mathbf{T}\mathcal{U}\neg\sigma))$ and $\mathbf{T}\mathcal{U}\neg(\mathbf{T}\mathcal{U}\neg\sigma)$. Now, if, similar to the case of $\mathsf{CTL}^{\star}$, we extend this minimal grammar by the derivable constraints, these fairness constraints are read as $\Box\Diamond\sigma$ and $\Diamond\Box\sigma$. The characteristic path formulae for ECTL^{+} are $\pi_1 \wedge \pi_2$ - those that allow Boolean combination of linear-time temporal operators and fairness constraints. Finally, the characteristic path formulae for $\mathsf{ECTL}^{\#}$ are $\sigma_1\mathcal{U}(\sigma_2 \wedge \Diamond\sigma_3), \Box(\sigma_1 \vee \Box\sigma_2), \sigma_1\mathcal{U}(\Box\sigma_2)$ and $\Box(\sigma_1\mathcal{U}\sigma_2)$.

It is important to note that the nesting of 'pure path formulae', totally unrestricted in $\mathsf{CTL}^{\star}$, is restricted in its sublogics by relevant grammar cases for path formulae. Below, for each of the logics $\mathsf{CTL}^{\star}$, $\mathsf{ECTL}^{\#}$, ECTL^{+}, ECTL and CTL, we will provide an example of a formula which is expressible in this logic but is not expressible in its sublogic. The structures of these formulae reflect the characteristic path formulae for the considered sublogic. Hence, we will refer to these formulae as characteristic formulae for a dedicated logic under consideration. For example, an indicative $\mathsf{CTL}^{\star}$ formula $\mathsf{A}\Diamond(\circ p \wedge \mathsf{E}\circ\neg p)$ mentioned above (4.1) is not an $\mathsf{ECTL}^{\#}$ formula. Rewriting it as $\mathsf{A}(\mathbf{T}\mathcal{U}(\circ p \wedge \mathsf{E}\circ\neg p))$ we can see that $\circ p \wedge \mathsf{E}\circ\neg p$, the right-hand side argument of the $\mathcal{U}$ operator, does not meet the $\mathsf{ECTL}^{\#}$ criteria: it is neither a state formula of the form $\sigma_1 \wedge \Diamond\sigma_2$ nor $\Box\sigma$. Recall that the validity of (4.1) is directly linked to the limit closure property [32]. If we consider an $\mathsf{ECTL}^{\#}$ formula

$$\mathsf{A}((p\mathcal{U}\Box q) \wedge (s\mathcal{U}\Box\neg q)) \tag{4.2}$$

we can see that this is not an ECTL^{+} formula because ECTL^{+} only allows Boolean combinations of the fairness constraints. In this $\mathsf{ECTL}^{\#}$ formula, $p\mathcal{U}\Box q$ and $s\mathcal{U}\Box\neg q$, hence their conjunction, are not admissible ECTL^{+} formulae. Further, an ECTL^{+} formula (4.3) that does not belong to ECTL is

$$\mathsf{E}(\Box\Diamond q \wedge \Diamond\Box\neg q) \tag{4.3}$$

as $\Box\Diamond q \wedge \Diamond\Box\neg q$ is not an admissible ECTL path formula.

Finally, the fairness constraint (4.4) which is expressible in ECTL cannot be constructed in CTL syntax as every temporal operator

$$\mathsf{E}\Box\Diamond q \tag{4.4}$$

in a CTL formula must be preceded by a path quantifier. Obviously, writing, for example, an E quantifier before the $\Diamond q$ we obtain an admissible CTL structure $\mathsf{E}\Box\mathsf{E}\Diamond q$. However, comparing $\mathsf{E}\Box\mathsf{E}\Diamond q$ with $\mathsf{E}\Box\Diamond q$, we can see that the latter requires a model where there

exists a path, along which $\Box\Diamond q$ is satisfied, while the former requires a model where there exists a path where each state gives rise for a path along which $\Diamond q$ is satisfied.

Note that it is important to distinguish the problem if a formula of a superlogic belongs to a sublogic and the problem if a formula of a superlogic can be expressed in a sublogic. For example, $\mathsf{E}(\Box\Diamond q \vee \Diamond\Box\neg q)$, similarly to formula (4.3) does not belong to ECTL but unlike (4.3), it is expressible in this logic, as
$\mathsf{E}(\Box\Diamond q \vee \Diamond\Box\neg q) \equiv \mathsf{E}\Box\Diamond q \vee \mathsf{E}\Diamond\Box\neg q$ which is an ECTL formula if we define $\vee$ via $\wedge$.

BTL Logic	$\mathsf{E}\Box\Diamond q$	$\mathsf{E}(\Box\Diamond q \wedge \Diamond\Box\neg q)$	$\mathsf{A}((p\mathcal{U}\Box q) \vee (s\mathcal{U}\Box\neg r))$	$\mathsf{A}\Diamond(\circ p \wedge \mathsf{E}\circ\neg p)$	Dual T & SC
$\mathcal{B}(\mathcal{U},\circ)$ (CTL)	X	X	X	X	This thesis
$\mathcal{B}(\mathcal{U},\circ,\Box\Diamond)$ (ECTL)	$\surd$	X	X	X	This thesis
$\mathcal{B}^+(\mathcal{U},\circ,\Box\Diamond)$ (ECTL^+)	$\surd$	$\surd$	X	X	$\surd$
$\mathcal{B}^+(\mathcal{U},\circ,\mathcal{U}\Box)$ ($\mathsf{ECTL}^\#$)	$\surd$	$\surd$	$\surd$	X	$\surd$
$\mathcal{B}^\star(\mathcal{U},\circ)$ ($\mathsf{CTL}^\star$)	$\surd$	$\surd$	$\surd$	$\surd$	X

Tab. 4.1: Classification of context-based tableaux systems for CTL-type logics and relevant difficult cases of concatenations of temporal operators and path quantifiers.

In Table 4.1, following the notation initially proposed in [32] and further tuned in [69], we represent BTL logics (listed in the first column) classified by their expressiveness using '$\mathcal{B}$' for 'Branching', followed by the set of only allowed modalities as parameters; $\mathcal{B}^+$ indicates admissible Boolean combinations of the modalities and $\mathcal{B}^\star$ reflects 'no restrictions' in either concatenations of the modalities or Boolean combinations between them. Columns 2-5 of Table 4.1 illustrate the indicative formulae for the logics under considerations as follows:

- for ECTL, column 2: $\mathsf{E}\Box\Diamond q$, formula (4.4)
- for ECTL^+, column 3: $\mathsf{E}(\Box\Diamond q \wedge \Diamond\Box\neg q)$, formula (4.3)
- for $\mathsf{ECTL}^\#$, column 4: $\mathsf{A}((p\mathcal{U}\Box q) \wedge (s\mathcal{U}\Box\neg q))$, formula (4.2)
- for $\mathsf{CTL}^\star$, column 5: $\mathsf{A}\Diamond(\circ p \wedge \mathsf{E}\circ\neg p)$, formula (4.1)

Writing the '$\surd$' in Table 4.1 against the listed logics we indicate if a logic meets these grammar rules. For example, column 2 now illustrates which of the logics can express the property $\mathsf{E}\Box\Diamond q$: while this property is not expressible in CTL, it becomes expressible in ECTL and any of its extensions. In this respect, ECTL has those minimal grammar requirements enabling to express the property, hence, we can treat $\mathsf{E}\Box\Diamond q$ as ECTL indicative formula.

The last column in this table reflects the development of the dual system of context-based tableaux (T) and sequent calculus (SC) for CTL-type logics. The method has been developed for $\mathsf{ECTL}^\#$ [16] where the motivation was to cope with more complex cases of fairness. We note that this method developed for $\mathsf{ECTL}^\#$ is obviously applicable to all weaker logics. However, it only tackles one weaker logic - ECTL^+ - efficiently, introducing unnecessary complications for other $\mathsf{ECTL}^\#$ sublogics. This is based upon the fact that ECTL^+ and $\mathsf{ECTL}^\#$ have similar cases of the Boolean combinations of eventualities in the scope of A and E: disjunctions of the eventualities in the scope of the A quantifier

and conjunctions of eventualities in the scope of the E quantifier, see [16] for details. Thus, Table 4.1 also reflects syntactical cases of concatenations of temporal operators and path quantifiers that are difficult for context-based tableaux.

To manage these cases, in addition to α- and β-rules, that are standard to the tableaux, we adapted the β^+-rules presented for PLTL (Figure 2.9) which use the context to force the eventualities to be fulfilled as soon as possible. As ECTL$^\#$ is more expressive than ECTL$^+$ in allowing new type of fairness constraints that use the $\mathcal{U}$ operator, the relevant rules introduced in [16] cover all difficult concatenations of operators in ECTL$^+$. However, simply treating the case of context-based tableaux for the other two sublogics of ECTL$^\#$ –ECTL and CTL– as solved by the relevant development for a richer logic ECTL$^\#$, would introduce extra unnecessary complexity in the construction of dual system of relevant tableaux and sequent calculi. Hence, for both ECTL and CTL, simpler context-based tableaux methods are required. We concentrate on bridging this gap in our roadmap in supplying BTL logics by this technique, and develop the method for CTL and ECTL.

4.2 CTL *and* ECTL *Logics*

Subsequently, we proceed by defining the CTL and ECTL semantic conditions in Definition 20. These are derived from the CTL* semantic evaluation rules given in Definition 17.

Definition 20 (CTL and ECTL Semantics)**.**

- *The semantics for logic* CTL *is obtained from the* CTL* *semantics given in Definition 17 by preserving the evaluation conditions* (s_T) - (s_A) *and* $(p_\bigcirc)$ - $(p_\mathcal{R})$ *and setting* π, π_1 *and* π_2 *to be a state formula.*
- *The semantics for logic* ECTL *is obtained from the above* CTL* *semantics by deleting the evaluation conditions* $(p_\vee)$ *and* $(p_\wedge)$ *and also restricting* π *in the following rules as follows:*
  - *in the* $(p_\bigcirc)$ *rule* π *is a state formula,*
  - *in the* $(p_\Diamond)$ *rule* π *is a state formula or a formula* $\Box\sigma$*,*
  - *in the* $(p_\Box)$ *rule* π *is a state formula or a formula* $\Diamond\sigma$*, and*
  - *in the* $(p_\mathcal{U})$ *and* $(p_\mathcal{R})$ *rules* π_1, π_2 *are state formulae.*

For technical convenience, we will use the fact that cyclic Kripke structures have the ability to characterise satisfiability in branching temporal logics.

Definition 21 (Cyclic Sequence, Path and Kripke Structure)**.** *Let* $\mathcal{K} = (S, R, I, L)$ *be a given Kripke structure. Let* z *be a finite sequence of states* $z = s_0, s_1, \ldots, s_j$ *in* S *such that* $(s_k, s_{k+1}) \in R$ *for every* $0 \leq k < j$*. Then*

- z *is a cyclic sequence if and only if there exists* s_i*,* $0 \leq i \leq j$ *such that* $(s_j, s_i) \in R$*. In this case, the subsequence* $s_i, \ldots, s_j$ *of* z *is called a loop denoted as* $\langle s_i, \ldots, s_j \rangle^\omega$*.*

- *If z is a cyclic sequence, then the path*

$$s_0, s_1, \ldots, s_{i-1}\langle s_i, s_{i+1}, \ldots, s_j\rangle^\omega$$

is denoted by $\mathsf{path}(z)$ *and is called cyclic.*

A Kripke structure $\mathcal{K}$ is cyclic if every fullpath is a cyclic path over a cyclic sequence of states. ∎

The fact that branching-time satisfiability can be reduced to the interpretation over cyclic models only, is derived from the existence of the finite model property [34], see also [53]. In particular, for any CTL (ECTL) formula φ, such that $\mathsf{Mod}(\varphi) \neq \emptyset$, there always exists a model $\mathcal{K} \in \mathsf{Mod}(\varphi)$ such that $\mathcal{K}$ is cyclic. Therefore, when speaking about the satisfiability in CTL (hence ECTL) we can consider *cyclic* Kripke structures. Cyclic paths are also known as *ultimately periodic* paths.

In this section, we introduce the grammars for CTL and ECTL formulae in NNF. From now on, we abbreviate by Q either of the path quantifiers A or E.

Definition 22 (Syntax of CTL and ECTL in NNF)**.** *Let* Prop *be a fixed set of propositions, then the sets of* CTL *formulae and* ECTL *formulae in* NNF *over* Prop *are given by the grammar (where the elements of Lit are called literals):*

$$\begin{array}{lcl} \mathit{Lit} & := & \mathbf{F} \mid \mathbf{T} \mid p \mid \neg p \text{ where } p \in \mathsf{Prop} \\ \sigma_{\mathsf{CTL}} & := & \mathit{Lit} \mid \sigma_1 \wedge \sigma_2 \mid \sigma_1 \vee \sigma_2 \mid \mathsf{Q}\bigcirc\sigma \mid \mathsf{Q}\Diamond\sigma \mid \mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2) \mid \mathsf{Q}\Box\sigma \mid \mathsf{Q}(\sigma_1 \mathcal{R} \sigma_2) \\ \sigma_{\mathsf{ECTL}} & := & \sigma_{\mathsf{CTL}} \mid \mathsf{Q}\Box\Diamond\sigma \mid \mathsf{Q}\Diamond\Box\sigma \end{array}$$

The following result (e.g. [68]) can be easily established.

Proposition 3. *For any* CTL *formula φ there exists a* CTL *formula,* $\mathsf{NNF}(\varphi)$*, which is in* NNF *such that* $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\mathsf{NNF}(\varphi))$*. For any* ECTL *formula φ there exists an* ECTL *formula,* $\mathsf{NNF}(\varphi)$*, which is in* NNF *such that* $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\mathsf{NNF}(\varphi))$*.*

Proof. By structural induction on the formulae, using the following well-known equivalences (e.g. [32]):

$\neg\mathbf{T} \equiv \mathbf{F}$ $\quad$ $\neg\mathbf{F} \equiv \mathbf{T}$ $\quad$ $\neg\neg\varphi \equiv \varphi$ $\quad$ $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$ $\quad$ $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$ $\quad$ $\neg\mathsf{A}\bigcirc\varphi \equiv \mathsf{E}\bigcirc\neg\varphi$ $\quad$ $\neg\mathsf{E}\bigcirc\varphi \equiv \mathsf{A}\bigcirc\neg\varphi$

$\neg\mathsf{A}\Box\varphi \equiv \mathsf{E}\Diamond\neg\varphi$ $\quad$ $\neg\mathsf{E}\Box\varphi \equiv \mathsf{A}\Diamond\neg\varphi$ $\quad$ $\neg\mathsf{A}\Diamond\varphi \equiv \mathsf{E}\Box\neg\varphi$ $\quad$ $\neg\mathsf{E}\Diamond\varphi \equiv \mathsf{A}\Box\neg\varphi$ $\quad$ $\neg\mathsf{A}(\varphi\mathcal{U}\psi) \equiv \mathsf{E}(\neg\varphi\mathcal{R}\neg\psi)$ $\quad$ $\neg\mathsf{E}(\varphi\mathcal{U}\psi) \equiv \mathsf{A}(\neg\varphi\mathcal{R}\neg\psi)$

$\neg\mathsf{A}(\varphi\mathcal{R}\psi) \equiv \mathsf{E}(\neg\varphi\mathcal{U}\neg\psi)$ $\quad$ $\neg\mathsf{E}(\varphi\mathcal{R}\psi) \equiv \mathsf{A}(\neg\varphi\mathcal{U}\neg\psi)$ $\quad$ $\neg\mathsf{A}\Box\Diamond\sigma \equiv \mathsf{E}\Diamond\Box\neg\sigma$ $\quad$ $\neg\mathsf{E}\Box\Diamond\sigma \equiv \mathsf{A}\Diamond\Box\neg\sigma$ $\quad$ $\neg\mathsf{A}\Diamond\Box\sigma \equiv \mathsf{E}\Box\Diamond\neg\sigma$ $\quad$ $\neg\mathsf{E}\Diamond\Box\sigma \equiv \mathsf{A}\Box\Diamond\neg\sigma$

□

Using the same notation introduced in Definition 8, we will write $\sim\varphi$ instead of $\mathsf{NNF}(\neg\varphi)$. Also, for a finite set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, we let $\sim\Phi = \mathsf{NNF}(\neg\bigwedge_{i=1}^n \varphi_i)$. Note that the sets of CTL and ECTL formulae in NNF are closed under the operation $\sim$.

For the formulation of our tableaux technique we will need a concept of a consistent (inconsistent) set of formulae, which is introduced in the following definition.

Definition 23 (Syntactically Consistent and Inconsistent Sets of Formulae)**.** *A set* Σ *of state formulae of* CTL *and* ECTL *in* NNF *is (syntactically) inconsistent (denoted* $\mathsf{Incons}(\Sigma)$*) if and only if* $\mathbf{F} \in \Sigma$ *or* $\{\sigma, \sim\sigma\} \subseteq \Sigma$ *for some* σ*. Otherwise,* Σ *is said to be consistent.*

Next, we introduce a concept of a *basic modality* which reflects the restrictions on forming the basic admissible combinations of temporal operators in the scope of a path quantifier. We consider a basic modality of CTL or ECTL logic to be of the form QT, where T is a temporal operator. The structure QT is generated by the grammar rules for these logics in Def. 19. We can identify all basic modalities in a given formula φ by finding its most embedded modality(es), say M_1, then looking at the next basic modality in which M_1 is embedded, etc. For example, a basic modality for CTL is any admissible combination of Q and a temporal operator, i.e. Q$\circ$, Q$\Diamond$, Q$\mathcal{U}$, Q$\Box$, and Q$\mathcal{R}$, while ECTL basic modalities are those identified above for CTL and, additionally, the modalities that appear due to new admissible combinations $\Diamond\Box$ and $\Box\Diamond$ in the scope of a path quantifier –Q$\Diamond\Box$ and Q$\Box\Diamond$. If we analyse a CTL formula E$\circ$A$\circ p$ then the most embedded basic modality, M_1, would be A$\circ p$, which is embedded as E$\circ M_1$. These are generalised in Definition 24.

Definition 24 (ECTL and CTL Basic Modalities)**.**

$$\begin{array}{lcl} \mathsf{M_{CTL}} & ::= & c \mid \mathsf{Q}\circ\mathsf{M} \mid \mathsf{Q}\Diamond\mathsf{M} \mid \mathsf{Q}(\mathsf{M}\mathcal{U}\mathsf{M}) \mid \mathsf{Q}\Box\mathsf{M} \mid \mathsf{Q}(\mathsf{M}\mathcal{R}\mathsf{M}). \\ \mathsf{M_{ECTL}} & ::= & \mathsf{M_{CTL}} \mid \mathsf{Q}\Box\Diamond\mathsf{M} \mid \mathsf{Q}\Diamond\Box\mathsf{M}. \end{array}$$

where c stands for a purely classical formula (we can consider a purely classical formula as a zero-degree basic modality) and M *stands for any basic modality of* CTL *in the definition of* $\mathsf{M_{CTL}}$ *and of* ECTL *in the definition of* $\mathsf{M_{ECTL}}$*.*

In what follows, every CTL modality Q$\mathcal{U}$ or Q$\Diamond$ is called *eventuality* and $(\mathsf{Q}\circ)^i$ stands for i consecutive occurrences of a basic modality Q$\circ$.

CTL tableau rules are based on fixpoint characterisation of its basic modalities: (in the equations below μ and ν stand for 'minimal fixpoint' and 'maximal fixpoint' operators, respectively)

$$\begin{array}{ll} \mathsf{E}\Box\varphi = \nu\rho(\varphi \wedge \mathsf{E}\circ\rho) & \mathsf{E}(\varphi\mathcal{R}\psi) = \nu\rho(\psi \wedge (\varphi \vee \mathsf{E}\circ\rho)) \\ \mathsf{A}\Box\varphi = \nu\rho(\varphi \wedge \mathsf{A}\circ\rho) & \mathsf{A}(\varphi\mathcal{R}\psi) = \nu\rho(\psi \wedge (\varphi \vee \mathsf{A}\circ\rho)) \\ \mathsf{E}\Diamond\varphi = \mu\rho(\varphi \vee \mathsf{E}\circ\rho) & \mathsf{E}(\varphi\mathcal{U}\psi) = \mu\rho(\psi \vee (\varphi \wedge \mathsf{E}\circ\rho)) \\ \mathsf{A}\Diamond\varphi = \mu\rho(\varphi \vee \mathsf{A}\circ\rho) & \mathsf{A}(\varphi\mathcal{U}\psi) = \mu\rho(\psi \vee (\varphi \wedge \mathsf{A}\circ\rho)) \end{array} \tag{4.5}$$

This fixpoint characterisation of basic CTL and ECTL modalities as maximal or minimal fixpoints give rise to their analytical classification as α- or β-formulae which are associated, in the tableau, with α- and β-rules: Q$\Box$, and Q$\mathcal{R}$ as maximal fixpoints are classified as α-formulae while Q$\Diamond$ and Q$\mathcal{U}$ as minimal fixpoints are β-formulae. This is also reflected in the known equivalences:

$$\begin{array}{ll} \mathsf{E}\Box\varphi = \varphi \wedge \mathsf{E}\circ\mathsf{E}\Box\varphi & \mathsf{E}(\varphi\mathcal{R}\psi) = \psi \wedge (\varphi \vee \mathsf{E}\circ\mathsf{E}(\varphi\mathcal{R}\psi)) \\ \mathsf{A}\Box\varphi = \varphi \wedge \mathsf{A}\circ A\Box\varphi & \mathsf{A}(\varphi\mathcal{R}\psi) = \psi \wedge (\varphi \vee \mathsf{A}\circ\mathsf{A}(\varphi\mathcal{R}\psi)) \\ \mathsf{E}\Diamond\varphi = \varphi \vee \mathsf{E}\circ\mathsf{E}\Diamond\varphi & \mathsf{E}(\varphi\mathcal{U}\psi) = \psi \vee (\varphi \wedge \mathsf{E}\circ\mathsf{E}(\varphi\mathcal{U}\psi)) \\ \mathsf{A}\Diamond\varphi = \varphi \vee \mathsf{A}\circ\mathsf{A}\Diamond\varphi & \mathsf{A}(\varphi\mathcal{U}\psi) = \psi \vee (\varphi \wedge \mathsf{A}\circ\mathsf{A}(\varphi\mathcal{U}\psi)) \end{array} \tag{4.6}$$

4.3 Dual Methods for CTL: Tableaux and Sequent Calculus

The tableau method determines whether a given set of CTL state formulae, Σ, is satisfiable or not. In addition, in the affirmative case a model (Kripke structure) of Σ can be generated from the tableau, whereas in the negative case we can generate a proof in the dual sequent calculus. In this section, we introduce the tableau method: we define the set of tableau rules and a general view and intuitions on how tableaux are constructed. In Section 4.3.1 we provide a precise algorithm for constructing tableaux in a systematic way. The soundness and completeness proofs are presented in Section 4.3.2. The Sequent Calculus for CTL are explained in Section 4.3.3.

We precede the formal introduction of the technique by its informal overview. The main concepts are informally introduced here and technically defined later in this section or in Section 4.3.1. A tableau for a set of CTL formulae is a graph, namely an AND-OR-Tree (for the account on AND-OR-Trees we refer an interested reader to [55]), where nodes are labelled by sets of CTL formulae. The initial node (or root) of the tableau is labelled by some given set of CTL formulae whose satisfiability we want to check. Non-terminal nodes are further expanded by applications of the tableau rules to their labelling sets. There are two kinds of *terminal nodes*. First, any node labelled by an inconsistent set of formulae (see Definition 23) is terminal. The second type are the so-called *loop-nodes*. Intuitively, we say that a branch (i.e. a path from the root to some node n) has a loop when the label of n (or some superset of it) has already appeared in this branch. In this case, n is called a loop-node. Loop-nodes are terminal whenever some precise eventuality fulfilment conditions hold in the branch. Intuitively, such conditions ensure that the (systematic) tableau for any satisfiable set of formulae represents a model of this set.

For the node expansion, we have the following types of tableau rules: α- and β-rules, the 'next-state' rule, which reflects a 'jump' from a 'state' to a 'pre-state', and, finally, characteristic to our approach, β^+-rules, where the use of the context (of an eventuality) is essential. In our procedure, the application of β^+-rules to eventualities is essential to detect 'bad' loops. Only if β^+-rules have already been applied to every eventuality in the branch and there is no inconsistent node in this branch, then we can establish if all the eventualities have been fulfilled. When this check for fulfilment of eventualities is positive we have a 'good loop' and this branch would be part of a model of the input formula. Otherwise, when there is at least one unfulfilled eventuality, we choose one to which a corresponding β^+-rule has not been applied.

Next we proceed with formally defining the construction of the tableaux and introducing necessary concepts and tableau rules.

Definition 25 (Tableau, Consistent and Incons. Node, Closed and Open Branch)**.** *A tableau for a set of* CTL *state formulae Σ is a labelled tree $\langle T, \tau, \Sigma\rangle$, where T is a tree, and τ is a mapping of the nodes of T to sets of state formulae, such that the following two conditions hold:*

- *The root is labelled by the set Σ.*
- *For any other node $m \in T$, its label $\tau(m)$ is a set of state formulae obtained as the result of the application of one of the rules in Figures 4.1, 4.2 and 4.3 to its parent node n. When the applied rule is R, we term m an R-successor of n.*

A node n of a tree T is consistent if its label, $\tau(n)$, is a (syntactically) consistent set of formulae (see Definition 23), else n is inconsistent. If a branch b of T, contains an inconsistent node, then b is closed else b is open.

$$(\wedge)\ \frac{\Sigma, \sigma_1 \wedge \sigma_2}{\Sigma, \sigma_1, \sigma_2} \qquad (\mathsf{Q}\Box)\ \frac{\Sigma, \mathsf{Q}\Box\sigma}{\Sigma, \sigma, \mathsf{Q}\circ\mathsf{Q}\Box\sigma}$$

$$(\vee)\ \frac{\Sigma, \sigma_1 \vee \sigma_2}{\Sigma, \sigma_1 \ \mid\ \Sigma, \sigma_2} \qquad (\mathsf{Q}\mathcal{U})\ \frac{\Sigma, \mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)}{\Sigma, \sigma_2 \ \mid\ \Sigma, \sigma_1, \mathsf{Q}\circ\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)}$$

$$(\mathsf{Q}\mathcal{R})\ \frac{\Sigma, \mathsf{Q}(\sigma_1 \mathcal{R} \sigma_2)}{\Sigma, \sigma_2, \sigma_1 \vee \mathsf{Q}\circ\mathsf{Q}(\sigma_1 \mathcal{R} \sigma_2)} \qquad (\mathsf{Q}\Diamond)\ \frac{\Sigma, \mathsf{Q}\Diamond\sigma}{\Sigma, \sigma \ \mid\ \Sigma, \mathsf{Q}\circ\mathsf{Q}\Diamond\sigma}$$

Figure 4.1: α- and β-rules.

Figure 4.1 follows the standard for the tableaux classification of rules into α-rules and β-rules - this is based on the analytic classification of CTL modalities and reflects their interpretation as fixpoints (see equations 4.6) of the formula (in the node label) that is 'designated' for the rule application. In the systematic tableau construction, at each node, the designated formula is chosen following some strategy that we specify later (in Section 4.3.1).

An α- or β-rule is applied to a node labelled by a set of formulae Σ, φ, where φ is a designated formula that determines the rule (to be applied), and Σ is a possibly empty set of formulae that accompany φ in the label of the node. Then, if φ is an α-formula - $\wedge$, $\mathsf{Q}\Box$, or $\mathsf{Q}\mathcal{R}$ - then a corresponding α-rule applies, while if φ is a β-formula - $\vee$, $\mathsf{Q}\mathcal{U}$, or $\mathsf{Q}\Diamond$- then a corresponding β-rule applies. These applications of α- and β-rules generate the set of formulae in the conclusion of the rule as label(s) for the successor node(s): one successor in case of an α-rule, or two successors in case of a β-rule. In β-rules we use $\mid$ to emphasize that successors are OR-siblings.

When a node n is labelled by an *elementary set of formulae* – i.e. a set which is exclusively formed by literals and formulae of the form $Q\circ\sigma$ – then this structure is analogous to a 'state' in the terminology of [83]; it enables us to construct the successors of n corresponding to 'pre-states' [83].

Let $\Sigma, \mathcal{A}, \mathcal{E}$ be an elementary set of formulae where

- Σ is a set of literals,
- $\mathcal{A}$ is a possibly empty set of $\mathsf{A}\circ$ formulae $\{\mathsf{A}\circ\sigma_1, \ldots, \mathsf{A}\circ\sigma_\ell\}$, and
- $\mathcal{E}$ is a non-empty set of $\mathsf{E}\circ$ formulae $\{\mathsf{E}\circ\sigma'_1, \ldots, \mathsf{E}\circ\sigma'_k\}$.

$$(\circ\mathsf{E})\ \frac{\Sigma, \mathcal{A}, \mathcal{E}}{\mathcal{A}^{\downarrow}, \sigma'_1\ \&\ \ldots\ \&\ \mathcal{A}^{\downarrow}, \sigma'_k} \qquad (\circ\mathsf{A})\ \frac{\Sigma, \mathcal{A}}{\mathcal{A}^{\downarrow}}$$

where $\mathcal{A}^{\downarrow} = \{\sigma_1, \ldots, \sigma_\ell\}$. Hence, $\mathcal{A}^{\downarrow}$ is empty if and only if $\mathcal{A}$ is empty.

Figure 4.2: Next-state rules ('&' joins AND-successors in the conclusion of $(\circ\mathsf{E})$).

According to the next proposition we are guaranteed to reach such a tree structure, where the last node of every branch, at this stage of the construction, is a state.

Proposition 4. *Any set of* CTL *state formulae has a tableau T such that the last node of every branch is labelled by an elementary set of state formulae.*

Proof. Repeatedly apply to every expandable node any applicable α- or β-rule until all expandable nodes are labelled by elementary sets of formulae. Then, the appropriate next-state rule of Figure 4.2 must be applied to every expandable node depending on the number of formulae starting by $\mathsf{E}\circ$ appearing at the node. □

Proposition 4 enables the application of the so-called 'next-state rules' depicted in Figure 4.2. Applying rule ($\circ\mathsf{E}$) we split the current branch at node n where the set

$$\Sigma, \mathsf{A}\circ\sigma_1, \ldots, \mathsf{A}\circ\sigma_\ell, \mathsf{E}\circ\sigma'_1, \ldots, \mathsf{E}\circ\sigma'_k \tag{4.7}$$

is satisfied, $0 \leq l$ and $k \geq 1$, into k branches: the number of branches is equal to the number of $\mathsf{E}\circ$ constraints, where the successors of n along these branches, say $m_1, \ldots, m_k$ are AND-successors of n. We label each AND-successor m_j ($1 \leq j \leq k$) in the following way. As any constraint $\mathsf{A}\circ\sigma_i$ ($1 \leq i \leq l$) in the premise of the rule would propagate σ_i to any successor node, then each of these successor nodes, m_j should have the set $\sigma_1, \ldots, \sigma_l$ as part of its label. The next part of the label of m_j is coming from the corresponding $\mathsf{E}\circ\sigma'_s$ ($1 \leq s \leq k$) as this existential constraint only determines the label for one of the AND-successors. Thus, each AND-successor m_j of n has its label of the form $\sigma_1, \ldots, \sigma_l, \sigma'_s$. The rule ($\circ\mathsf{E}$) splits branches in a 'conjunctive' way, and we use the symbol & to represent the generation of AND-successors of node n. Thus, the graphs generated with the application of the 'next-state' rule ($\circ\mathsf{E}$) are indeed AND-OR trees. When $k = 0$ in the elementary set of formulae that labels node n (4.7), the rule ($\circ\mathsf{A}$) is applied. The application of both rules ($\circ\mathsf{E}$) and ($\circ\mathsf{A}$) represents a 'jump' to the next state, hence the set of literals Σ in (4.7) disappears in every child produced by these rules.

Next we extend our set of tableau rules with the new two rules named as β^+-rules (Figure 4.3). Note that the $(\mathsf{Q}\Diamond)^+$ rule can be derived from the application of the $(\mathsf{Q}\mathcal{U})^+$ to the CTL formula $\mathsf{T}\mathcal{U}\sigma$. These rules, similarly to β-rules, also split a branch into two branches. Moreover, whenever a β^+-rule -$(Q\mathcal{U})^+$ or $(Q\Diamond)^+$- is applicable, so is the respective β-rule -$(Q\mathcal{U})$ or $(Q\Diamond)$. Which rules, β-rules or β^+-rules are applied at each step of the tableau construction, does not affect the correctness. However, for completeness, some strategy is required. It is easy to see that the extension with β^+-rules preserves the property given in Proposition 4.

$$(\mathsf{Q}\mathcal{U})^+ \ \frac{\Sigma, \mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)}{\Sigma, \sigma_2 \ \mid \ \Sigma, \sigma_1, \mathsf{Q}\circ\mathsf{Q}((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)} \qquad (\mathsf{Q}\Diamond)^+ \ \frac{\Sigma, \mathsf{Q}\Diamond\sigma}{\Sigma, \sigma \ \mid \ \Sigma, \mathsf{Q}\circ\mathsf{Q}((\sim\Sigma')\mathcal{U}\sigma)}$$

where $\Sigma' = \Sigma \setminus \{(\mathsf{A}\circ)^i \mathsf{A}\Box\sigma \in \Sigma \mid i \geq 0\}$

Figure 4.3: β^+-Rules

These β^+-rules are the only rules in our system that make use of the context -their application forces the eventualities to be satisfied as soon as possible (from the point

of the tableau construction where an eventuality is selected to be expanded with a β^+-rule). The strategy of selecting eventualities is identical to the one used in PLTL. The context is given by the possibly empty set Σ that accompanies the designated formula, which in this case is an eventuality. Inside one of formulae of the right-hand child of each β^+-rule we add a conjunct $\sim\Sigma'$ that is calculated by deleting some specific formulae from Σ (the context). In particular, if Σ is empty, then so is Σ' and the formula $\sim\Sigma'$ is the constant $\mathbf{F}$.

In what follows, any formula of the form $\mathsf{Q}((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)$ (respectively, $\mathsf{Q}((\sim\Sigma')\mathcal{U}\sigma)$) is called *the contextualised variant* of $\mathsf{Q}(\sigma_1\mathcal{U}\sigma_2)$ (resp. $\mathsf{Q}\Diamond\sigma$) provided that $\mathsf{Q}\circ\mathsf{Q}((\sigma_1\wedge\sim\Sigma')\mathcal{U}\sigma_2)$ (respectively, $\mathsf{Q}\circ\mathsf{Q}((\sim\Sigma')\mathcal{U}\sigma)$) has been obtained by the application of the corresponding β^+-rule to a formula $\mathsf{Q}(\sigma_1\mathcal{U}\sigma_2)$ (resp. $\mathsf{Q}\Diamond\sigma$) with a context Σ. Note that the contextualised variants will appear after one application of the rule ($\circ\mathsf{E}$) or ($\circ\mathsf{A}$), which removes all $\mathsf{Q}\circ$ prefixes.

Recall that $\sim\Sigma'$ is the NNF of the negation of the conjunction of all formulae in Σ' that are left from Σ after performing the set-theoretical difference constraint indicated in the formulation of the rule. The idea now is that $\sim\Sigma'$ should also be satisfied until σ_2 becomes satisfied. This prevents the repetition of the context while σ_2 is 'delayed'. Note that Σ' does not include the $\mathsf{A}\Box$ constraint (prefixed by any sequence of the $\mathsf{A}\circ$ constraints) because these formulae would be necessarily repeated along any branch - indeed, if we use $\sim\Sigma$ instead of $\sim\Sigma'$ we will generate a branch for each $\mathsf{A}\Box$ that will be immediately closed.

Example 10. *Consider $\Sigma = \{q, \mathsf{E}\circ\neg p, \mathsf{A}\Box a\}$ to be the context of the formula $\mathsf{A}\Diamond p$. Then, $\Sigma' = \{q, \mathsf{E}\circ\neg p\}$ and $\sim\Sigma'$ is the formula $\neg q \vee \mathsf{A}\circ p$. Consequently, the contextualised variant $\mathsf{A}((\neg q \vee \mathsf{A}\circ p)\mathcal{U}p)$ of $\mathsf{A}\Diamond p$, preceded by $\mathsf{A}\circ$, is introduced in the branch where $\mathsf{A}\Diamond p$ is delayed. This variant says that p should be fulfilled (in all branches) after the next state, but while p is not fulfilled some formula in the context $\{q, \mathsf{E}\circ\neg p, \mathsf{A}\Box a\}$ should be 'violated'. Since $\mathsf{A}\Box a$ is a formula that cannot be violated in any branch (it is in the initial node), then either $\neg q$ or $\sim\mathsf{E}\circ\neg p$ (in NNF, $\mathsf{A}\circ p$) should be satisfied in all path until p is satisfied. If the context of eventuality $\mathsf{A}\Diamond p$ was empty or, for example, $\{\mathsf{A}\Box a\}$, then the contextualised variant of $\mathsf{A}\Diamond p$ would be $\mathsf{A}(\mathbf{F}\mathcal{U}p)$.*

Next, we illustrate in Example 11 the application of the β^+-rule $(\mathsf{E}\mathcal{U})^+$ to a node labelled by $\{\mathsf{E}(p\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$. A tableau for $\{\mathsf{E}(p\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$ is also exhibited in [1, 47]. In Section 4.3.1 we will use this tableau as a running example, complete its construction, and compare it with the tableau presented in [1, 47]. As we have done in previous chapters, we highlight with the gray color the formula (or subformula) to which the β^+-rule is (or will be) applied.

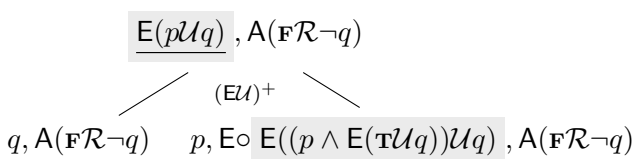


Figure 4.4: Application of rule $(\mathsf{E}\mathcal{U})^+$

Example 11. *In Figure 4.4, the context of the eventuality* $\mathsf{E}(p\mathcal{U}q)$ *in the root is the formula* $\mathsf{A}(\mathbf{F}\mathcal{R}\neg q)$*. The rule* $(\mathsf{E}\mathcal{U})^+$ *splits the tableau into two branches. In the right-hand successor the middle formula* $\mathsf{E}\circ\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathcal{U}q))\mathcal{U}q)$ *contains the contextualised variant of the eventuality* $\mathsf{E}(p\mathcal{U}q)$ *that is constructed by the conjunction of* p *and* $\sim\mathsf{A}(\mathbf{F}\mathcal{R}\neg q) = \mathsf{E}(\mathbf{T}\mathcal{U}q)$ *as new left-hand component of the until formula. The effect is that along the future states (from the next one), while* q *is not satisfied, not only* p *should be satisfied, but also the negation of the current context (i.e.* $\mathsf{E}(\mathbf{T}\mathcal{U}q)$*) should be satisfied.*

4.3.1 Systematic Tableau Construction

In this section we define a recursive algorithm, $\mathcal{A}^{sys}$, that constructs a *fully expanded systematic tableau* for a given set of formulae Σ. Intuitively, 'fully expanded' means that we 'complete' the formation of the tableau in the sense that every expandable node has been already expanded. In Chapter 5 we explain how the implementation of this algorithm, with more parameters and results, returns a model when the resulting tableau is open or, otherwise, a proof in the dual sequent calculus that corresponds with the closed tableau. We first define all the main concepts involved in the algorithm.

A branch is a finite linear structure – formed by the successive nodes, from the root to a leaf– inside the tree-shaped structure of the tableau. When the leaf of a branch has occurred previously in the branch, it is called a loop-node, and the sequence of nodes finitely represents an infinite loop branch. We will load the current branch by 'stages' instead of 'node-by-node' mode.

Definition 26 (Stage)**.** *Given a branch,* b *of a tableau* T*, a stage in* b *is every maximal subsequence of successive nodes* $n_i, n_{i+1}, \ldots, n_j$ *in* b *such that* $\tau(n_k)$ *is not a* $(\circ\mathsf{E})$*-child or* $(\circ\mathsf{A})$*-child of* $\tau(n_{k-1})$*, for all* k *such that* $i < k \leq j$*. We denote by* $\mathsf{stages}(b)$ *the sequence of all stages of* b*. The successor relation on* $\mathsf{stages}(b)$ *is induced by the successor relation on* b*. The labelling function* τ *is extended to stages as the union of the original* τ *applied to every node in a stage.*

When the input is a satisfiable set of formulae, the systematic tableau aims to obtain one loop-node since it represents a cycle in a cyclic Kripke structure that could be part of a model of the input set of formulae.

Definition 27 (Loop-node)**.** *Let* $b = n_0, n_1, \ldots, n_i$ *(where* $i > 0$*) be a tableau branch and* $\mathsf{stages}(b) = s_0, s_1, \ldots, s_m$ *(where* $m > 0$ *and* $n_i \in s_m$*). Then,* n_i *is a loop-node if there exists some* $0 \leq j < m$ *such that* $\tau(n_i) \subseteq \tau(s_j)$*. We say that* s_j *is the companion stage of the node* n_i *and* $s_j, \ldots, s_m$ *are the stages in the loop.*

Example 12. *To illustrate the notions of stage and loop-node, let us consider the tree (tableau) in Figure 4.5, which contains our running example as a sub-tree. Note that the application of rule* $(\circ\mathsf{E})$*, at step 2, generates two AND-successors (AND-edges are denoted with a big circle). The left successor is our running example and will be fully expanded later (Figure 4.7). The right-most branch is formed by five nodes. This branch has three stages, the first one is formed by the first two nodes, hence its label is the union of their labels, i.e. the set* $\{\mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q), \mathsf{E}\circ\mathsf{E}(p\mathcal{U}q) \wedge \mathsf{E}\circ\neg q, \mathsf{E}\circ\mathsf{E}(p\mathcal{U}q), \mathsf{E}\circ\neg q\}$*. The second stage in that branch is labeled by* $\{\mathsf{A}(\mathbf{F}\mathcal{R}\neg q), \neg q, \mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$*. The last node* $\mathsf{A}(\mathbf{F}\mathcal{R}\neg q)$ *is a loop-node because it is included in the second stage, which is its companion stage.*

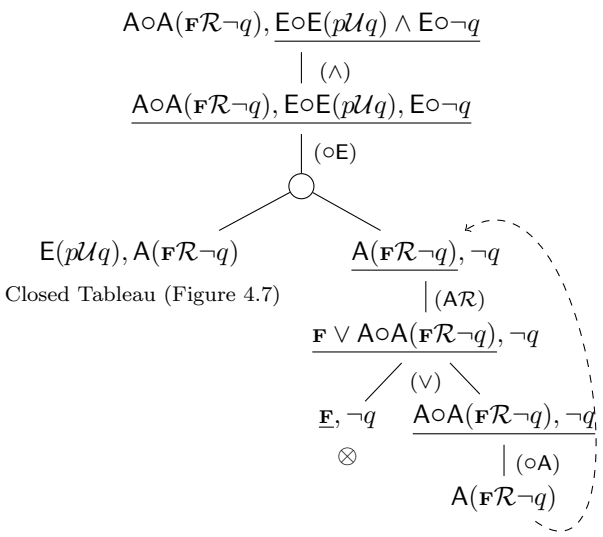


Figure 4.5: A closed tableau for $\{\mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q), \mathsf{E}\circ\mathsf{E}(p\mathcal{U}q) \wedge \mathsf{E}\circ\neg q\}$.

When a loop-node is found, the fulfilment of eventualities along the branch must be ensured. When this fulfilment condition holds we say that the branch is eventuality-covered. It is obvious that universal eventualities ($\mathsf{A}\mathcal{U}$ or $\mathsf{A}\Diamond$) should be fulfilled in all branches that go across the node where these eventualities appear, as reflected by rules ($\circ\mathsf{Q}$). However, as a consequence of the distribution of existential formulae in different branches performed by the rule ($\circ\mathsf{E}$), an existential eventuality ($\mathsf{E}\mathcal{U}$ or $\mathsf{E}\Diamond$) should be fulfilled only along the branch it belongs to after the ($\circ\mathsf{E}$) splitting, but not in the other branches. Indeed, more than one existential eventuality could appear in the stages of a tableau branch b, however not all of them should be fulfilled in b but, on the contrary, some of them could be 'delayed' and then, by an application of rule ($\circ\mathsf{E}$), could be 'sent' to a different tableau branch.

Example 13. *For example, in Figure 4.5 after the splitting by* ($\circ\mathsf{E}$) *in two AND-successors, the existential eventuality* $\mathsf{E}(p\mathcal{U}q)$ *goes to the left subtree, hence it could not be satisfied in the right subtree, where* $\mathsf{E}\circ\neg q$ *forces to satisfy* $\neg q$.

Hence, the notion of eventuality coverage (in branches) differentiates existential eventualities from universal ones es explained above.

Definition 28 (Eventuality Fulfilment and Eventuality-covered Branch)**.** *Let b be a tableau branch such that* $\mathsf{stages}(b) = s_0, \ldots, s_n$. *An eventuality* $\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)$ *(resp.* $\mathsf{Q}(\Diamond\sigma)$*) is fulfilled in the branch b if and only if* $\sigma_2 \in \tau(s_k)$ *(resp.* $\sigma \in \tau(s_k)$*) for some* $0 \leq k \leq n$. *The branch b is eventuality-covered if and only if the following two conditions hold:*

- *Every eventuality* $\mathsf{A}\mathcal{U}$ *or* $\mathsf{A}\Diamond$ *that occurs at some stage of b is fulfilled in b.*
- *For every eventuality* $\varphi = \mathsf{E}(\sigma_1 \mathcal{U} \sigma_2)$ *(resp.* $\varphi = \mathsf{E}\Diamond\sigma$*) that occurs in some stage of b either φ is fulfilled in b or* $\mathsf{E}\circ\varphi \notin s_k$ *for some* $0 \leq k \leq n$.

We only need to check the above two conditions for those eventualities that have not been selected in the branch, because loops cannot contain an unfilled eventuality that has been selected at some point.

In the next example we have a branch with two existential eventualities.

Example 14. *Consider a branch of five nodes labelled by the following sets:*

1. $\neg p, \neg q, \mathsf{E}\Diamond p, \mathsf{E}\Diamond q$
2. $\neg p, \neg q, \mathsf{E}\circ\, \mathsf{E}((p \vee q \vee \mathsf{A}\Box\neg q)\mathcal{U}p), \mathsf{E}\Diamond q$
3. $\neg p, \neg q, \mathsf{E}\circ\, \mathsf{E}((p \vee q \vee \mathsf{A}\Box\neg q)\mathcal{U}p), \mathsf{E}\circ\mathsf{E}\Diamond q$
4. $\mathsf{E}((p \vee q \vee \mathsf{A}\Box\neg q)\mathcal{U}p)$
5. p

This branch would be generated by successively applying the rules $(\mathsf{E}\Diamond)^+$ *to the selected eventuality* $\mathsf{E}\Diamond p$ *in 1,* $(\mathsf{E}\Diamond)$ *to eventuality* $\mathsf{E}\Diamond q$ *in 2,* $(\circ\mathsf{E})$ *in 3, and* $(\mathsf{E}\Diamond)^+$ *in 4, and taking only one of the children at each step. This branch is eventuality covered, though* $\mathsf{E}\Diamond q$ *is not fulfilled in the branch. Indeed the application of the rule* $(\circ\mathsf{E})$ *at item 3 generates two AND-branches and the other branch starts with node* $\mathsf{E}\Diamond q$*.*

Since $(\circ\mathsf{E})$-children are AND-siblings, whenever one of them does not have a possible model, the parent node is unsatisfiable. On the contrary, to ensure the satisfiability of a node where $(\circ\mathsf{E})$ is applied it should have a collection of satisfiable branches that includes all the $(\circ\mathsf{E})$-successors of any node labelled by an elementary set of formulae. These collections of branches are called *bunches*. Next, we define the notion of a bunch and how bunches determine whether a tableau is open, closed and fully expanded.

Definition 29 (Bunch, Closed Bunch and Closed Tableau)**.** *A bunch H is a collection of branches which is maximal with respect to* $(\circ\mathsf{Q})$*-successors, i.e. every* $(\circ\mathsf{A})$*-successor and every* $(\circ\mathsf{E})$*-successor of any node in H is also in H. A bunch H is closed if and only if at least one of its branches is closed, otherwise it is open. A tableau is closed if, and only if, all its bunches are closed.*

Definition 30 (Fully Expanded Bunch and Tableau)**.** *A branch b is fully expanded if and only if either b is closed (see Definition 25) or the last node in b is a loop-node and b is eventuality-covered. A bunch is fully expanded if all its branches are fully expanded. A tableau is fully expanded if all its bunches are fully expanded.*

Example 15. *The tableau in Figure 4.5 is closed in spite of the open sub-tableau at the right* $(\circ\mathsf{E})$*-successor of the second node. Any bunch in this tableau should include at least one branch across the left node* $\{\mathsf{E}(p\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$ *and at least one branch across the right node* $\{\mathsf{A}(\mathbf{F}\mathcal{R}\neg q), \neg q\}$*. Independently of the branch chosen in the right-hand tree (the closed or the open one), any branch in the left-hand subtree is closed (the details of the closed tableau for* $\{\mathsf{E}(p\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$ *are given later). Therefore, any bunch in the tableau of Figure 4.5 is closed.*

Any open tableau has at least one open bunch, formed by one or more open branches. Open branches are ended in a loop-node. Open bunches represent models, specifically *cyclic models* as defined in Definition 21.

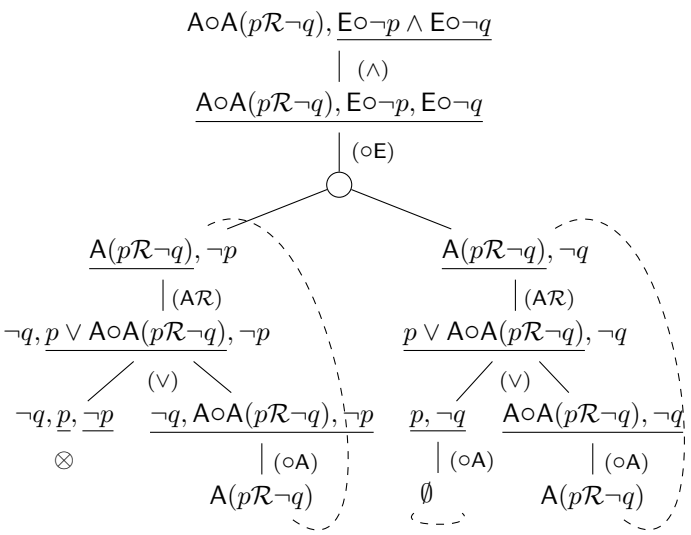


Figure 4.6: An open tableau for $\{\mathsf{A}\circ\mathsf{A}(p\mathcal{R}\neg q), \mathsf{E}\circ\neg p \wedge \mathsf{E}\circ\neg q\}$.

Example 16. *In Figure 4.6 we depict an open tableau that is a slight modification of the closed tableau in Example 12 (Figure 4.5). This tableau has three open branches. One open branch is crossing the left-hand* $(\circ\mathsf{E})$*-child of the second node. Two open branches cross the right-hand child of that node. Hence, there are two possible open bunches, depending on which of the latter two branches we choose.*

Next, we introduce the recursive Algorithm 7, called $\mathcal{A}^{sys}$, that constructs a *fully expanded systematic tableau* for any input Σ and returns a Boolean value saying if such tableau is closed. The current branch of the tableau construction is passed through the recursive calls. We can see the branches as lists of stages, which in turn are lists of formulae. Hence, in the first call the branch that receives $\mathcal{A}^{sys}$ as input is $[[\Sigma]]$. We illustrate the steps of the algorithm with details of the construction of the tableaux in Figures 4.5, 4.6 and 4.7.

Lines 1-6 gives the three non-recursive (or simple) cases of $\mathcal{A}^{sys}$. Lines 1-2 deal with the case of the empty input which is trivially satisfiable. Note that by application of the rule $(\circ\mathsf{A})$ when $\mathcal{A}$ is empty we could get this case. Indeed, this is a special case of loop-node labelled by the empty set, whose companion is itself. In Figure 4.6 there is a node $\{p, \neg q\}$ where $(\circ\mathsf{A})$ is applied and the loop on the empty set is produced. Lines 3-4 deal with the terminal nodes by inconsistency. For example, the four terminal nodes in Figure 4.7 correspond to this case. Three of their labels contain $q, \neg q$ or $\mathbf{F}$ or both. The fourth (right-most) terminal node contains the inconsistent set $\{\mathsf{E}(\mathbf{T}\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$.

In line 5, Algorithm 7 detects a 'good loop'. An example of this is the last node in the right-most branch in Figure 4.5.

Otherwise, there is either a 'bad loop' (i.e. a loop in a non-eventuality-covered branch) or not a loop, and the algorithm tries to apply (if possible) a β^+ rule to some selected eventuality. This is the goal of lines 7-9.

Line 7 stands for the first checked case whenever some tableau rule must be applied to Σ. The rules α, β could really be applied in any order. However, next-state rules can only be applied to elementary sets of formulae. For generating simplest contextualised

Algorithm 7: $\mathcal{A}^{sys}$
Input = Σ: set of formulae, b: Branch.
Output = *is_closed*: boolean, b': Branch.

```
1  if Σ = ∅ then
2  |  is_closed, b' := false, b;
3  else if Incons(Σ) then
4  |  is_closed, b' := true, b;
5  else if Σ ⊆ τ(s) for some stage s in b & Ev_Covered(b) then
6  |  is_closed, b' := false, b;
7  if β+_is_applicable(Σ) then
8  |  select_eventuality(Σ) ;
9  |  is_closed, b' := apply_β+_rule(Σ, b);
10 else if α_β_is_applicable(Σ) then
11 |  is_closed, b' := apply_α_β_rule(Σ, b);
12 else // is_elementary(Σ)
13 |  Let Σ = Φ, A∘(Ψ), E∘σ_1, ..., E∘σ_k, Φ is a set of literals and k ≥ 0.
14 |  if k ≥ 1 then
15 |  |  Let Σ_i = Ψ, σ_i for all 1 ≤ i ≤ k;
16 |  |  n := k;
17 |  else
18 |  |  Σ_1, n := Ψ, 1
19 |  end
20 |  i, is_closed :=0, false ;
21 |  while is_closed = false & i < n do
22 |  |  is_closed, b' := A^sys(Σ_i, b + [[Σ_i]]);
23 |  |  i := i + 1 ;
24 |  end
25 end
```

variants, we just prioritised the application of the β^+-rules. Note that at each stage at most one eventuality can be selected, and at most one β^+ rule is applied. More precisely, the call *select_eventuality*(Σ) selects an unfulfilled eventuality to force its fulfilment by application of the corresponding β^+-rule. Since β^+-rules keep the contextualised variant as selected, the predicate call β^+*_is_applicable*(Σ) gives true if and only if either there is already a selected eventuality $\mathsf{Q}\mathcal{U} \in \Sigma$ or else, both of the following hold: there exists a not selected $\mathsf{Q}\circ\mathsf{Q}\mathcal{U} \in \Sigma$ and there exists a (non-fulfilled) eventuality to be selected. In the former case, *select_eventuality*(Σ) keeps the selection. In the latter case, it does perform a selection.

Example 17. *For example, in Figure 4.7 the eventuality* $\mathsf{E}(p\mathcal{U}q)$ *is selected in the root, and its contextualised variant* $\mathsf{E}((p \wedge \mathsf{E}(\mathtt{T}\mathcal{U}q))\mathcal{U}q)$ *is kept selected along the right-most branch.*

The application of an specific β or a β^+ rule R, inside the calls of the procedures *apply_*β^+*_rule*(Σ, b) or *apply_*α*_*β*_rule*(Σ, b), produces two R-children, namely Σ_1

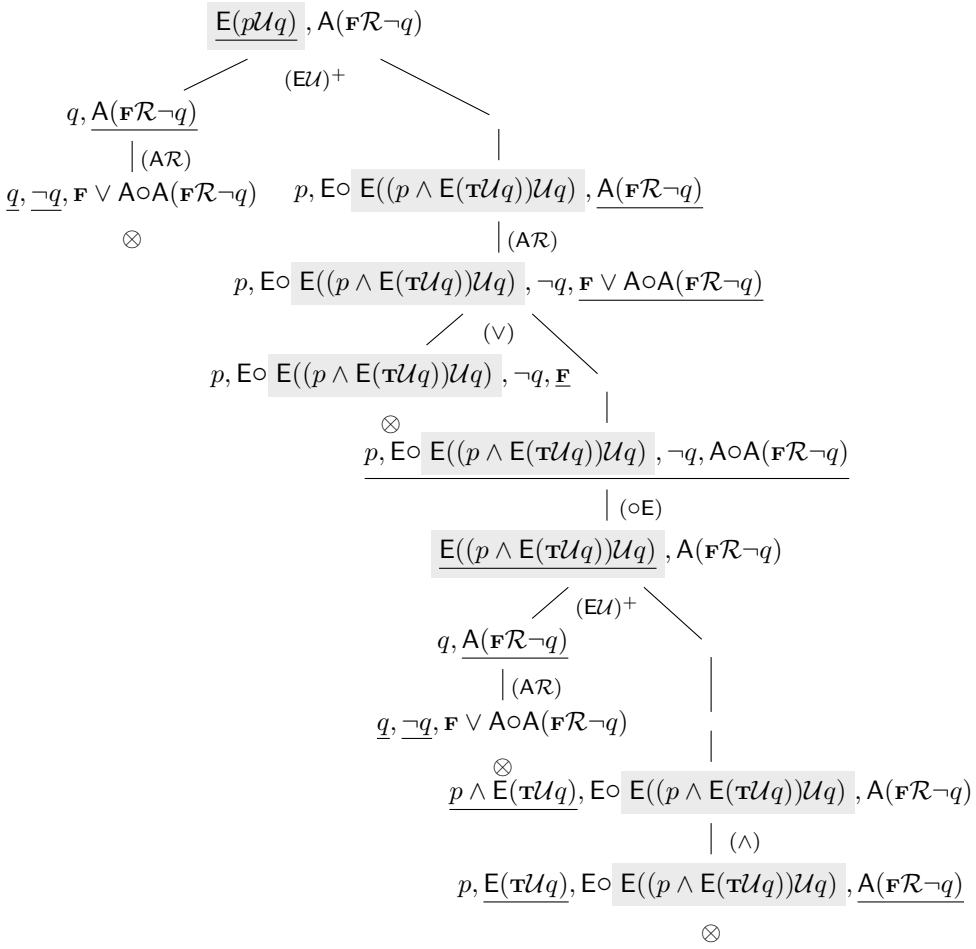


Figure 4.7: A closed tableau for $\{\mathsf{E}(p\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$

and Σ_2, that are OR-siblings. Hence, in these cases, lines 9 and 11, first do the recursive call $is\_closed, b' := \mathcal{A}^{sys}(\Sigma_1, b_1)$. Then, only if is_closed is true, do the call $is\_closed, b' := \mathcal{A}^{sys}(\Sigma_2, b_2)$. The above branches b_1 and b_2 stand for the adequate actualisation according to Σ_1 and Σ_2. As an example, in Algorithm 8, we provide the details for applying the rule $(\mathsf{E}\mathcal{U})^+$ to input $\Sigma, \mathsf{E}(\sigma_1\mathcal{U}\sigma_2)$. Note that branches are conveniently updated in recursive calls. In fact, function $update(b, \Sigma)$ adds to the last stage of b the formulae in Σ that were not previously in that stage.

Algorithm 8: Apply $(\mathsf{E}\mathcal{U})^+$ to $\Phi = \Sigma, \mathsf{E}(\sigma_1\mathcal{U}\sigma_2)$.

```
1 Σ := Φ \ {E(σ1 U σ2)};
2 is_closed, b' := A^sys(Σ1, b1) where Σ1 = Σ ∪ {σ2} and b1 = update(b, Σ1);
3 if is_closed = true then
4 |   is_closed, b' := A^sys(Σ2, b2) where Σ2 = Σ ∪ {σ1, E((σ1 ∧ ∼Σ') U σ2)}
5 |                                          and b2 = update(b, Σ2);
6 end
```

Example 18. *In the tableau of Figure 4.7, the first application of* $(\mathsf{E}\mathcal{U})^+$ *to the root*

node, calls $\mathcal{A}^{sys}(\Sigma_1, b)$ *for* $\Sigma_1 = \{q, \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$ *which returns true in is_closed after the construction of the left-most sub-tree. Then, the call* $\mathcal{A}^{sys}(\Sigma_2, b)$ *where* Σ_2 *contains the contextualised variant, constructs the right sub-tree to return also true. Hence, the whole tableau is closed. Note that, in Figure 4.7, there is a second call to apply* $(\mathsf{E}\mathcal{U})^+$ *that works in a similar way. At this second application, the selected eventuality is* $\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathcal{U}q))\mathcal{U}q)$*. Its contextualised variant is* $\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathcal{U}q) \wedge \mathbf{F})\mathcal{U}q)$*, which is kept selected.*

For each rule that splits branches (i.e. β or β^+) there is an application algorithm similar to Algorithm 8. For the α-rules, that enlarge the branch with one node, a recursive call to $\mathcal{A}^{sys}$ on the only one child suffices.

In line 10, the algorithm checks and applies applicable α- or β-rules. When no α- or β-rules are applicable, it means that Σ is an elementary set of formulae (line 12). Therefore, in lines 13-23, the rule $(\circ\mathsf{E})$ or $(\circ\mathsf{A})$ is applied, by iterating recursive calls to $\mathcal{A}^{sys}(\Sigma_i, b + [[\Sigma_i]])$ for each child labelled by Σ_i. Note that, each application of a tableau rule $(\circ\mathsf{Q})$ produces a recursive call for a $(\circ\mathsf{Q})$-child Σ_i, where the current branch b is actualised to include a new stage containing just Σ_i (this is expressed in the algorithm by $b + [[\Sigma_i]]$). Let us also observe that $(\circ\mathsf{E})$-children are AND-siblings, hence the iteration on the $(\circ\mathsf{E})$-children terminates as soon as one of them is closed. On the contrary, if every $(\circ\mathsf{E})$-child is open the tableau should have a collection of open branches including all the $(\circ\mathsf{E})$-successors of any node labelled by an elementary set of formulae, i.e. and open bunch. Any open bunch of the systematic tableau, constructed by the algorithm $\mathcal{A}^{sys}$ introduced in this section, enables the construction of a model for the initial set of formulae.

Example 19. *For example, in Figure 4.5, if the left-most sub-tableau (i.e. the closed tableau in the figure) is firstly constructed, then its right-hand child would not be constructed, since any bunch would be necessarily closed (see Example 12). In Example 16 we show an open tableau that contains two open bunches.*

Example 20. *The call* $\mathcal{A}^{sys}$ *with input* $\Sigma = \{\mathsf{E}(p\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$ *constructs the closed tableau in Figure 4.7 as explained along this section. A tableau for the same input* Σ *is also exhibited in [1, 47]. Note the direct correspondence between our context-based tableau (Figure 4.7) and the one in [1, 47] – they have exactly the same nodes. The right-most branch, in our case, closes by (syntactical) inconsistency, likewise all the other branches. The difference is that, in this branch, the inconsistency comes from the use of the context in the selected eventuality. The corresponding branch in the tableau in [1, 47] is closed by the detection of a 'bad loop' using information loaded during the construction of the previously constructed branches.*

Let us recall that in [16] some (subsumption-like) simplification rules were introduced to ensure the termination of the tableau method for the logic $\mathsf{ECTL}^{\#}$. The method for CTL (and also for ECTL) requires the following simplification rule:

$$(\sqsubset \mathsf{Q}\mathcal{U})\ \{\mathsf{Q}((\sigma_1 \wedge \chi)\mathcal{U}\sigma_2), \mathsf{Q}(\sigma_1\mathcal{U}\sigma_2)\} \longrightarrow \{\mathsf{Q}((\sigma_1 \wedge \chi)\mathcal{U}\sigma_2)\} \tag{4.8}$$

By means of this rule, any contextualised variant of an eventuality φ subsumes the original eventuality φ that could repeatedly appear otherwise. Our algorithm systematically performs these simplifications in nodes.

To complete this section, we provide an example of systematic tableau construction involving a universal eventuality $\mathsf{A}\Diamond$ where bunches should cross along the two applications of $(\circ\mathsf{E})$.

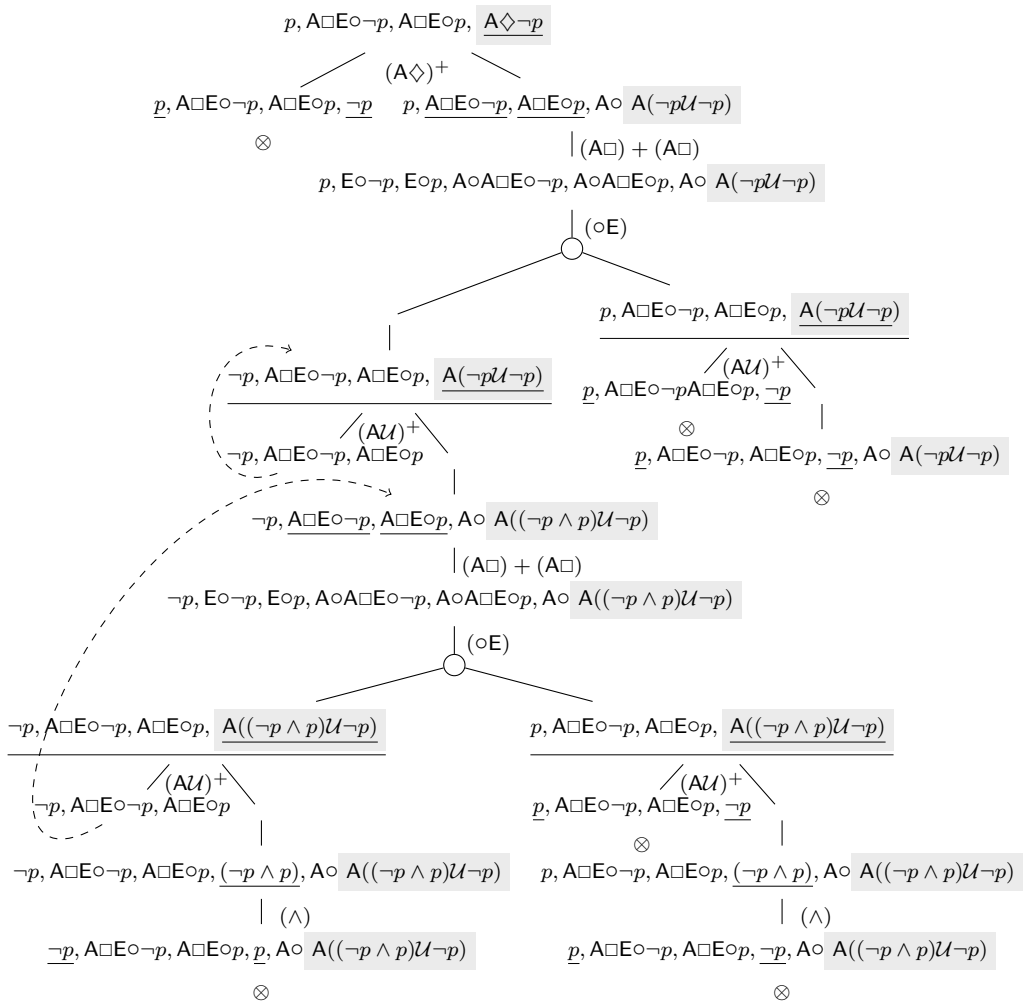


Figure 4.8: A closed tableau for $\{p, \mathsf{A}\Box\mathsf{E}\circ p, \mathsf{A}\Box\mathsf{E}\circ\neg p, \mathsf{A}\Diamond\neg p\}$.

Example 21. *Figure 4.8 presents a closed fully-expanded tableau for the unsatisfiable set of formulae* $\{p, \mathsf{A}\Box\mathsf{E}\circ p, \mathsf{A}\Box\mathsf{E}\circ\neg p, \mathsf{A}\Diamond\neg p\}$*. The systematic construction starts by selecting the unique eventuality and applying the rule* $(\mathsf{A}\Diamond)^+$*. The successive contextualised variants of this eventuality are kept selected and the rule* $(\mathsf{A}\mathcal{U})^+$ *is applied to them after each application of the next-state rule* $(\circ\mathsf{E})$*. The rules* $(\mathsf{A}\Box)$ *and* $(\wedge)$ *are applied to reach the elementary sets where* $(\circ\mathsf{E})$ *is applied. For saving space, we sometimes represent two applications of the rule* $(\mathsf{A}\Box)$ *in the same step.*

It is worth noting that, in spite of the two open branches (see the two loops in Figure 4.8), every bunch in the figure is closed. Indeed, any bunch should contain branches for crossing the four $(\circ\mathsf{E})$*-nodes (doubly underlined in Figure 4.8). It is easy to see that any bunch contains at least one closed branch. The 'bad loop detection approach'*

([1, 47]) would create a 'bad loop' branch similar to the largest branch of our tableau in which $\circ p$ is satisfied. For that, the information of the other branches (where p is satisfied) should be used.

4.3.2 Soundness and Completeness

In this section, we prove that the presented tableau method for CTL is sound and complete. We essentially adapt the soundness and completeness proofs developed in [16] to CTL. In Section 4.4 we extend both results to ECTL.

To prove the soundness of the tableau method for CTL (Theorem 1), we show that every tableau rule in Figures 4.1, 4.2 and 4.3 preserves satisfiability in the sense of the next Lemma 1.

Lemma 1 (Soundness of the Tableau Rules for CTL)**.** *Consider all the rules in Figures 4.1, and 4.2 and 4.3.*

1. *For any α-rule of the form $\dfrac{\Sigma}{\Sigma_1}$, we have $\mathsf{Sat}(\Sigma)$ if and only if $\mathsf{Sat}(\Sigma_1)$.*
2. *For any β-rule and any β^+-rule of the form $\dfrac{\Sigma}{\Sigma_1 \mid \Sigma_2}$, we have $\mathsf{Sat}(\Sigma)$ if and only if $\mathsf{Sat}(\Sigma_1)$ or $\mathsf{Sat}(\Sigma_2)$.*
3. *If Σ is a consistent set of literals, then*
   (a) $\mathsf{Sat}(\Sigma \cup \{\mathsf{A}\circ\sigma_1, \ldots, \mathsf{A}\circ\sigma_\ell, \mathsf{E}\circ\sigma'_1, \ldots, \mathsf{E}\circ\sigma'_k\})$ *if and only if* $\mathsf{Sat}(\{\sigma_1, \ldots, \sigma_\ell, \sigma'_i\})$ *for all* $1 \leq i \leq k$.
   (b) $\mathsf{Sat}(\Sigma \cup \{\mathsf{A}\circ\sigma_1, \ldots, \mathsf{A}\circ\sigma_\ell\})$ *if and only if* $\mathsf{Sat}(\{\sigma_1, \ldots, \sigma_\ell\})$.

Proof. All these statements follow very easily from the 'systematic' application of the semantic definitions of the temporal modalities, except the 'if' direction' for the β^+-rules. Next we prove the 'if' direction of the rules $(\mathsf{Q}\mathcal{U})^+$ for $\mathsf{Q} = \mathsf{E}$ and $\mathsf{Q} = \mathsf{A}$, because this proof entails the proof for the rules $(\mathsf{Q}\Diamond)^+$ as particular cases (using the abbreviation $\Diamond\sigma = \mathrm{T}\mathcal{U}\sigma$).

For the 'if' direction of the rule $(\mathsf{E}\mathcal{U})^+$, let $\mathcal{K} \models \Sigma, \mathsf{E}(\sigma_1 \mathcal{U} \sigma_2)$ and let x be the path in $\mathcal{K}$ such that $\mathcal{K}, x \models \Sigma, \sigma_1 \mathcal{U} \sigma_2$. Then, let j be the least $i \geq 0$ such that $\mathcal{K}, x, i \models \sigma_2$. If $j = 0$, then $\mathcal{K}, x, 0 \models \Sigma, \sigma_2$. Otherwise, if $j > 0$ then $\mathcal{K}, x, m \models \sigma_1$, for all $0 \leq m < j$. Consider k to be the greatest of those m such that $\mathcal{K}, x, m \models \Sigma$. Hence, $\mathcal{K}, x, h \models \sim\Sigma$, for all h such that $k + 1 \leq h < j$. In particular, by definition of Σ' (obtained from Σ) it is easy to see that $\mathcal{K}, x, h \models \sigma$ for every $\sigma \in (\Sigma \setminus \Sigma')$. Therefore, $\mathcal{K}, x, h \models \sim\Sigma'$, for all h such that $k + 1 \leq h < j$. Consequently,

$$\mathcal{K}, x, k \models \Sigma, \sigma_1, \mathsf{E}\circ\mathsf{E}((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2).$$

For the 'if' direction of rule $(\mathsf{A}\mathcal{U})^+$, let us suppose that

$$\mathsf{UnSat}(\Sigma \cup \{\sigma_2\}) \text{ and } \mathsf{UnSat}(\Sigma \cup \{\sigma_1, \mathsf{A}\circ\mathsf{A}((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)\}).$$

We will show that $\mathsf{UnSat}(\Sigma \cup \{\mathsf{A}(\sigma_1 \mathcal{U} \sigma_2)\})$. For that, let us consider any arbitrary $\mathcal{K}$ such that $\mathcal{K} \models \Sigma$ to show that $\mathcal{K} \not\models \mathsf{A}(\sigma_1 \mathcal{U} \sigma_2)$. By the above unsatisfiability hypothesis, if $\mathcal{K} \models \Sigma$, then both $\mathcal{K} \not\models \sigma_2$ and $\mathcal{K} \not\models \sigma_1 \wedge \mathsf{A}\circ\mathsf{A}((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)$. Then, there are two possible cases. First, if $\mathcal{K} \models \neg\sigma_1 \wedge \neg\sigma_2$, then it is obvious that $\mathcal{K} \not\models \mathsf{A}(\sigma_1 \mathcal{U} \sigma_2)$. Second, if $\mathcal{K} \models \neg\sigma_2 \wedge \neg\mathsf{A}\circ\mathsf{A}((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)$, then there exists $x_1 \in \mathsf{fullpaths}(\mathcal{K})$ and $i_1 > 0$ that satisfy both $\mathcal{K}, x_1, j \models \neg\sigma_2$ for all j such that $0 \leq j \leq i_1$, and $\mathcal{K}, x_1, i_1 \models \neg\sigma_1 \vee \Sigma'$. Since all the formulae in $\Sigma \setminus \Sigma'$ are satisfied in all states along all paths, indeed $\mathcal{K}, x_1, i_1 \models \neg\sigma_1 \vee \Sigma$. Therefore, if $\mathcal{K}, x_1, i_1 \models \neg\sigma_1$, then obviously $\mathcal{K} \not\models \mathsf{A}(\sigma_1 \mathcal{U} \sigma_2)$. Otherwise, if $\mathcal{K}, x_1, i_1 \models \Sigma$, applying the same reasoning for $\mathcal{K} \restriction x_1(i_1)$ as we did above for $\mathcal{K}$, we can conclude that there should be a path $x_2 \in \mathsf{fullpaths}(\mathcal{K} \restriction x_1(i_1))$ and some $i_2 > 0$ such that either $\mathcal{K} \restriction x_1(i_1), x_2, j \models \neg\sigma_2$ for all j such that $i_1 \leq j \leq i_2$ and $\mathcal{K} \restriction x_1(i_1), x_2, i_2 \models \neg\sigma_1 \vee \Sigma$. Hence, if $\mathcal{K} \restriction x_1(i_1), x_2, i_1 \models \neg\sigma_1$, then trivially $\mathcal{K} \not\models \mathsf{A}(\sigma_1 \mathcal{U} \sigma_2)$. Otherwise, $\mathcal{K} \restriction x_1(i_1), x_2, i_1 \models \Sigma$. Hence, there are two possible scenarios: 1.) After a finite number of iterations we get a path $y = x_1^{<i_1}, x_2^{<i_2}, \cdots x_k^{<i_k} \cdots$ such that $\mathcal{K}, y, j \models \neg\sigma_2$ for all j such that $0 \leq j \leq i_k$ and $\mathcal{K}, y, i_k \models \neg\sigma_1$. 2.) The infinite iteration of the second case yields a path $y = x_1^{<i_1}, x_2^{<i_2}, \cdots, x_k^{<i_k}, \cdots$ (that exists by the limit closure property) such that $\mathcal{K}, y, i \models \neg\sigma_2$ for all $i \geq 0$. In both scenarios we have $\mathcal{K} \not\models \mathsf{A}(\sigma_1 \mathcal{U} \sigma_2)$ holds for any arbitrary $\mathcal{K}$ that satisfies Σ. Thus, $\mathsf{UnSat}(\Sigma \cup \{\mathsf{A}(\sigma_1 \mathcal{U} \sigma_2)\})$. □

According to Lemma 1 and the definition of the closed tableau, we prove the following result.

Theorem 1 (Soundness of the Tableau Method for CTL)**.** *Given any set of state formulae Σ, if there exists a closed tableau for Σ then* $\mathsf{UnSat}(\Sigma)$.

Proof. In a closed tableau for Σ, at least one leaf in each bunch must have an inconsistent set of formulae that labels it. Therefore, this set is unsatisfiable. Then, by (the converse of) Lemma 1, the label of the root node, Σ, is unsatisfiable. □

Next, we prove the refutational completeness of the tableau method for CTL (Theorem 2). For that, we firstly define the notion of a saturated stage and prove some auxiliary properties of the stages and bunches of the systematic tableau. These properties are necessary to prove that every open bunch in the systematic tableau represents a model of the initial set of formulae Σ (Lemma 2).

Definition 31 ($\alpha\beta^+$-saturated Stage)**.** *We say that a stage $s = n_i, \ldots, n_j$ in the tableau $\mathcal{A}^{sys}$ for Σ is $\alpha\beta^+$-saturated if and only if it satisfies the following conditions:*

1. *For all $\sigma_1 \wedge \sigma_2 \in \tau(s)$: $\{\sigma_1, \sigma_2\} \subseteq \tau(s)$.*
2. *For all $\mathsf{Q}\Box\sigma \in \tau(s)$: $\{\sigma, \mathsf{Q}\circ\mathsf{Q}\Box\sigma\} \subseteq \tau(s)$.*
3. *For all $\sigma_1 \vee \sigma_2 \in \tau(s)$: $\sigma_1 \in \tau(s)$ or $\sigma_2 \in \tau(s)$.*
4. *For all $\mathsf{Q}(\sigma_1 \mathcal{R} \sigma_2) \in \tau(s)$: $\{\sigma_2, \sigma_1 \vee \mathsf{Q}\circ\mathsf{Q}(\sigma_1 \mathcal{R} \sigma_2)\} \subseteq \tau(s)$.*
5. *For all $\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2) \in \tau(s)$: $\sigma_2 \in \tau(s)$ or $\{\sigma_1, \mathsf{Q}\circ\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)\} \subseteq \tau(s)$ or $\{\sigma_1, \mathsf{Q}\circ\mathsf{Q}((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)\} \subseteq \tau(s)$, where $\Sigma' = (\tau(n_i) \setminus \{\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)\}) \setminus \{(\mathsf{A}\circ)^i \mathsf{A}\Box\varphi \mid i \geq 0$ and $(\mathsf{A}\circ)^i \mathsf{A}\Box\varphi \in \tau(n_i)\}$.*

6. *For all* $\mathsf{Q}(\Diamond\sigma) \in \tau(s)$ *:* $\sigma \in \tau(s)$ *or* $\{\mathsf{Q}\circ\mathsf{Q}(\Diamond\sigma)\} \subseteq \tau(s)$ *or* $\{\mathsf{Q}\circ\mathsf{Q}((\sim\Sigma')\mathcal{U}\sigma)\} \subseteq \tau(s)$*, where* $\Sigma' = (\tau(n_i) \setminus \{\mathsf{Q}\Diamond\sigma\}) \setminus \{(\mathsf{A}\circ)^i\mathsf{A}\Box\varphi \mid i \geq 0$ *and* $(\mathsf{A}\circ)^i\mathsf{A}\Box\varphi \in \tau(n_i)\}$.

The construction of the systematic tableau applies exactly one β^+-rule to exactly one selected eventuality (if any) at the first node of the stage, and then applies exhaustively all the applicable α- and β-rules to the formulae in the stage, until the branch closes, or its leaf is labelled by an elementary set, or it contains a loop-node. Consequently, the following result can be trivially proved by construction.

Proposition 5. *Given any set of state formulae* Σ*, the systematic tableau* $\mathcal{A}^{sys}$ *for* Σ *is fully expanded.*

Proof. It is trivial, by construction, that every stage in $\mathcal{A}^{sys}$ is $\alpha\beta^+$-saturated. □

Next we prove a crucial property of the systematic tableau management of eventualities by means of the selection policy.

Proposition 6. *Let* b *be an open branch of the tableau* $\mathcal{A}^{sys}$ *for* Σ*.*

1. *If a formula* $\mathsf{Q}(\sigma_1\mathcal{U}\sigma_2)$ *is selected at some stage* $s_i \in \mathsf{stages}(b)$*, then there exists some stage* $s_k \in \mathsf{stages}(b)$ *(for some* $k \geq i$*) such that* $\sigma_2 \in \tau(s_k)$ *and* $\sigma_1 \in \tau(s_j)$ *for all* $j \in \{i, \ldots, k-1\}$*.*

2. *If a formula* $\mathsf{Q}(\Diamond\sigma)$ *is selected at some stage* $s_i \in \mathsf{stages}(b)$*, then there exists some stage* $s_k \in \mathsf{stages}(b)$ *(for some* $k \geq i$*) such that* $\sigma \in \tau(s_k)$*.*

Proof. We will prove item 1. Item 2 is the particular case where $\sigma_1 = \mathbf{T}$ and $\sigma_2 = \sigma$. If $\mathsf{Q}(\sigma_1\mathcal{U}\sigma_2)$ is the selected formula at stage $s_i \in \mathsf{stages}(b)$, by algorithm 7, the set labelling the first node at each stage s_j $(j \geq i)$ of b has the form

$$\Sigma_{s_j}, \mathsf{Q}((\sigma_1 \wedge (\sim\Sigma_{s_i} \wedge \cdots \wedge \sim\Sigma_{s_{j-1}}))\mathcal{U}\sigma_2)$$

where each Σ_{s_j} is the context of the selected formula containing the contextualised variant of $\mathsf{Q}(\sigma_1\mathcal{U}\sigma_2)$ at the first node of each stage s_j. Since no other β^+-rule is applied each Σ_{s_j} is a subset of the finite set formed by all state formulae that are subformulae of some formula in Σ_{s_i} and their negations. Hence, there are a finite number of different Σ_{s_j}. Therefore, after finitely many applications of the β^+-rule, $\Sigma_{s_h} = \Sigma_{s_j}$, for some $h >= i$, for some $j \in \{i, \ldots, h-1\}$, and $\sigma_1 \wedge (\sim\Sigma_{s_i} \wedge \cdots \wedge \sim\Sigma_{s_{h-1}}) \in \tau(s_h)$. In particular, $\sim\Sigma_{s_h} \in \tau(s_h)$, hence, Σ_{s_h} must be inconsistent. Since b is open, this is a contradiction. This means that, for some $k \geq i$ the application of the corresponding β^+-rule should force that $\sigma_2 \in \tau(s_k)$. In addition, by Proposition 5 and Definition 31(5), $\sigma_1 \in \tau(s_j)$ for all $j \in \{i, \ldots, k-1\}$. □

Lemma 2 (Model Existence)**.** *Let* Σ *be any set of formulae. For any fully expanded bunch* H *of the tableau* $\mathcal{A}^{sys}$ *for* Σ*, there exists a Kripke structure* $\mathcal{K}_H$ *such that* $\mathcal{K}_H \models \Sigma$*.*

Proof. Let H be any fully expanded bunch of the tableau $\mathcal{A}^{sys}$ for Σ. We define $\mathcal{K}_H = (S, R, L)$ such that $S = \bigcup_{b\in H} \mathsf{stages}(b)$ and for any $s \in S$: $L(s) = \{p \mid p \in \tau(n) \cap \mathsf{Prop}$ for some node $n \in s\}$. R is the relation induced in $\mathsf{stages}(b)$ for each $b \in H$. Any

branch in $b \in H$ is open, hence b ends in a loop-node. Moreover, every eventuality has been selected in some stage of b. Therefore, there exists a (possibly empty) set Σ_ℓ such that for some $i \geq 0$: $b = s_0, s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_j, n_\ell$, where each s_h stands for a stage and n_ℓ is a non-expandable loop-node labelled by Σ_ℓ whose companion node is the first node at stage s_i. We are going to prove the following fact:

$$\mathcal{K}_H, s_a, 0 \models \sigma \text{ for any } a \in \{0, \ldots, j\} \text{ and any formula } \sigma \text{ in } L(s_a)$$

by structural induction on the formula σ.

The base of the induction, for $\sigma = p \in \mathsf{Prop}$, follows by definition of $\mathcal{K}_H$.

The cases where σ has one of the forms $\sigma_1 \wedge \sigma_2$, $\mathsf{Q}\Box\sigma$, $\sigma_1 \vee \sigma_2$ and $\mathsf{Q}(\sigma_1 \mathcal{R} \sigma_2)$ are trivial by Definition 31 and the induction hypothesis. Hence, to complete the inductive proof we will show that $\mathcal{K}_H, s_a, 0 \models \mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)$ for any $\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2) \in L(s_a)$.

The case for all $\mathsf{Q}\Diamond\sigma \in L(s_a)$ follows as a particular case by $\Diamond\sigma \equiv \mathbf{T}\mathcal{U}\sigma$. Consider any $\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2) \in L(s_a)$. Since b is eventuality-covered and n_ℓ is a loop-node, $\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)$ must be the selected eventuality at some node between the states s_a and s_j. Hence, by Proposition 6 and the definition of $\mathcal{K}_H$, there should be a state $s_k \in S$ (for some $a \leq k \leq j$) such that $\sigma_2 \in L(s_k)$ and $\sigma_1 \in L(s_z)$ for all $z \in \{a, \ldots, k-1\}$. Then, by induction hypothesis, $\mathcal{K}_H, s_k, 0 \models \sigma_2$ and $\mathcal{K}_H, s_z, 0 \models \sigma_1$ for all $z \in \{a, \ldots, k-1\}$. Therefore, $\mathcal{K}_H, s_a, 0 \models \mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2)$.

To complete the proof, we show that the successor relation between states in $\mathcal{K}_H$ is well-defined. For that, consider any tableau node in any stage s_a that is labelled by an elementary set

$$\{\Sigma, \mathsf{A}\circ\sigma_1, \ldots, \mathsf{A}\circ\sigma_n, \mathsf{E}\circ\sigma'_1, \ldots, \mathsf{E}\circ\sigma'_k\}$$

where Σ is a consistent set of literals, by rule ($\circ\mathsf{E}$), s_a has (in $\mathcal{K}_H$) a successor state s^i_{a+1}, for each $i \in \{1, \ldots, k\}$, such that $L(s^i_{a+1}) = \{\sigma_1, \ldots, \sigma_n, \sigma'_i\}$. We can assume (by the above proved fact) that $\mathcal{K}_H, s^i_{a+1}, 0 \models \{\sigma_1, \ldots, \sigma_n, \sigma'_i\}$ for all $i \in \{1, \ldots, k\}$. Therefore, we can infer that $\mathcal{K}_H, s_a, 0 \models \{\Sigma, \mathsf{A}\circ\sigma_1, \ldots, \mathsf{A}\circ\sigma_n, \mathsf{E}\circ\sigma'_1, \ldots, \mathsf{E}\circ\sigma'_k\}$. □

Next, we prove the refutational completeness of the tableau method.

Theorem 2 (Refutational Completeness for CTL)**.** *For any set of state formulae Σ, if* $\mathsf{UnSat}(\Sigma)$ *then there exists a closed tableau for Σ.*

Proof. Suppose the contrary, i.e. there exists no closed tableau for Σ. Then the systematic tableau $\mathcal{A}^{sys}$ for Σ is open. Hence, there is at least one fully expanded bunch H in $\mathcal{A}^{sys}$. By Lemma 2, there exists a Kripke structure $\mathcal{K}_H$ such that $\mathcal{K}_H \models \Sigma$. Consequently, $\mathsf{Sat}(\Sigma)$. □

Finally, we prove the completeness of our tableau method for CTL, and the first step here is to show that the method terminates.

Theorem 3 (Termination of the Tableau Method for CTL)**.** *For any set of state formulae Σ, the construction of the fully expanded tableau $\mathcal{A}^{sys}$ for Σ terminates.*

Proof. Tableau rules produce a finite branching, hence König's Lemma, applies. Therefore, it suffices to prove that every branch is finite. By Proposition 6, the application of a β^+-rule to a selected formula stops after a finite number of steps. Since the number of selectable eventualities in any open branch is finite, any open branch is eventuality-covered

after a finite number of eventuality selections. Recall that we assume the eventuality selection strategy to be fair. □

Theorem 4 (Completeness of the Tableau Method for CTL)**.** *For any set of state formulae Σ, if Σ is satisfiable then there exists a (finite) open fully expanded tableau for Σ.*

Proof. By Theorems 1 and 3, and the fact that the fully expanded systematic tableau $\mathcal{A}^{sys}$ for Σ is finite and cannot be closed. □

4.3.3 The Dual Sequent Calculus for CTL

$$(\wedge)\ \frac{\Sigma, \sigma_1, \sigma_2 \vdash \mathbf{F}}{\Sigma, \sigma_1 \wedge \sigma_2 \vdash \mathbf{F}} \quad (\vee)\ \frac{\Sigma, \sigma_1 \vdash \mathbf{F} \mid \Sigma, \sigma_2 \vdash \mathbf{F}}{\Sigma, \sigma_1 \vee \sigma_2 \vdash \mathbf{F}} \quad (Ctd)\ \frac{}{\Sigma, \sigma, \sim\sigma \vdash \mathbf{F}} \quad (\mathbf{F})\ \frac{}{\Sigma, \mathbf{F} \vdash \mathbf{F}}$$

$$(\mathsf{Q}\mathcal{R})\ \frac{\Sigma, \sigma_2, \sigma_1 \vee \mathsf{Q}\circ\mathsf{Q}(\sigma_1 \mathcal{R} \sigma_2) \vdash \mathbf{F}}{\Sigma, \mathsf{Q}(\sigma_1 \mathcal{R} \sigma_2) \vdash \mathbf{F}} \qquad (\mathsf{Q}\Box)\ \frac{\Sigma, \sigma, \mathsf{Q}\circ\mathsf{Q}\Box\sigma \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Box\sigma \vdash \mathbf{F}}$$

$$(\mathsf{Q}\mathcal{U})\ \frac{\Sigma, \sigma_2 \vdash \mathbf{F} \quad \Sigma, \sigma_1, \mathsf{Q}\circ\mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2) \vdash \mathbf{F}}{\Sigma, \sigma_1 \mathcal{U} \sigma_2 \vdash \mathbf{F}} \qquad (\mathsf{Q}\Diamond)\ \frac{\Sigma, \sigma \vdash \mathbf{F} \quad \Sigma, \mathsf{Q}\circ\mathsf{Q}\Diamond\sigma \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Diamond\sigma \vdash \mathbf{F}}$$

$$(\mathsf{Q}\mathcal{U})^+\ \frac{\Sigma, \sigma_2 \vdash \mathbf{F} \quad \Sigma, \sigma_1, \mathsf{Q}\circ\mathsf{Q}((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2) \vdash \mathbf{F}}{\Sigma, \mathsf{Q}(\sigma_1 \mathcal{U} \sigma_2) \vdash \mathbf{F}}$$

$$(\mathsf{Q}\Diamond)^+\ \frac{\Sigma, \sigma \vdash \mathbf{F} \quad \Sigma, \mathsf{Q}\circ\mathsf{Q}((\sim\Sigma')\mathcal{U}\sigma) \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Diamond\sigma \vdash \mathbf{F}}$$

$$(\circ\mathsf{E})\ \frac{\Sigma, \sigma \vdash \mathbf{F}}{\Sigma_0, \mathsf{A}\circ\Sigma, \mathsf{E}\circ\sigma \vdash \mathbf{F}} \qquad (\circ\mathsf{A})\ \frac{\Sigma \vdash \mathbf{F}}{\Sigma_0, \mathsf{A}\circ\Sigma \vdash \mathbf{F}}$$

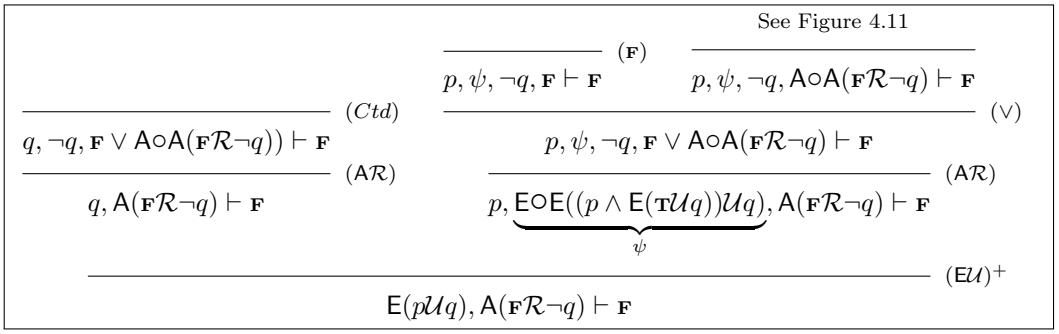
Figure 4.9: The Sequent Calculus for $\mathcal{C}_{\vdash}$.
In $(\mathsf{Q}\mathcal{U})^+$ and $(\mathsf{Q}\Diamond)^+$, $\Sigma' = \Sigma \setminus \{(\mathsf{A}\circ)^i \mathsf{A}\Box\sigma \in \Sigma \mid i \geq 0\}$. In $(\circ\mathsf{E})$ and $(\circ\mathsf{A})$, Σ_0 is a set of literals.

In this section we introduce a sequent calculus for CTL, called $\mathcal{C}_{\vdash}$, that is dual to the tableau method presented in previous sections. Indeed, the sequent calculus, given in Figure 4.9, is simply a reformulation of the tableau rules as a one-sided sequent calculus, the right-hand side of every sequent is the constant $\mathbf{F}$.

The calculus $\mathcal{C}_{\vdash}$ contains classical rules, such as $(\wedge)$ and $(\vee)$ for Booleans, and (Ctd) and $(\mathbf{F})$ for contradictions. The last two rules correspond to the branch closing conditions of the tableau method and they are premise-free, while all the other rules have at least one *premise*. In the rules $(\mathsf{Q}\mathcal{R}), (\mathsf{Q}\Box), (\mathsf{Q}\mathcal{U}), (\mathsf{Q}\Diamond), (\mathsf{Q}\mathcal{U})^+$, and $(\mathsf{Q}\Diamond)^+$, the symbol Q again denotes either of the path quantifiers, E or A. Therefore, each of these rules stands for a pair of analogous rules for each path quantifier, which correspond directly to the tableaux rules of the same name.

Note that the children of an elementary set with more than one formula starting by $\mathsf{E}\circ$ are AND-siblings, so that in the calculus it is enough to refute one of this AND-children. This corresponds to the tableau notion of a closed bunch (Definition 29).

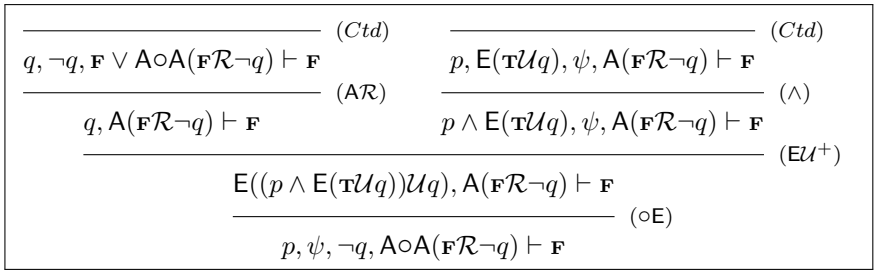
Example 22. *In Figure 4.10, we show the sequent proof that corresponds to the closed tableau in Figure 4.7.*



$$\dfrac{\dfrac{}{q, \neg q, \mathbf{F} \vee \mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q)) \vdash \mathbf{F}}\,(Ctd)}{q, \mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(\mathsf{A}\mathcal{R}) \qquad \dfrac{\dfrac{}{p, \psi, \neg q, \mathbf{F} \vdash \mathbf{F}}\,(\mathbf{F}) \qquad \dfrac{}{p, \psi, \neg q, \mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}}{p, \psi, \neg q, \mathbf{F} \vee \mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(\vee)}{p, \underbrace{\mathsf{E}\circ\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathcal{U}q))\mathcal{U}q)}_{\psi}, \mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(\mathsf{A}\mathcal{R})$$

$$\dfrac{\mathsf{E}(p\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}{\qquad}\,(\mathsf{E}\mathcal{U})^{+}$$

Figure 4.10: Sequent proof for $\{\mathsf{E}(p\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$.



$$\dfrac{\dfrac{\dfrac{\dfrac{}{q, \neg q, \mathbf{F} \vee \mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(Ctd)}{q, \mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(\mathsf{A}\mathcal{R}) \qquad \dfrac{}{p, \mathsf{E}(\mathbf{T}\mathcal{U}q), \psi, \mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(Ctd)}{p \wedge \mathsf{E}(\mathbf{T}\mathcal{U}q), \psi, \mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(\wedge)}{\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathcal{U}q))\mathcal{U}q), \mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(\mathsf{E}\mathcal{U}^{+})}{p, \psi, \neg q, \mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q) \vdash \mathbf{F}}\,(\circ\mathsf{E})$$

Figure 4.11: Sequent proof for $\{p, \psi, \neg q, \mathsf{A}\circ\mathsf{A}(\mathbf{F}\mathcal{R}\neg q)\}$ where $\psi = \mathsf{E}\circ\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathcal{U}q))\mathcal{U}q)$.

The soundness and completeness of $\mathcal{C}_{\vdash}$ easily follows from its duality with the tableau method for CTL presented in the previous sections.

Theorem 5 (Soundness)**.** *For any set of* CTL *formulae* Σ*, if there exists a sequent proof of* $\Sigma \vdash \mathbf{F}$*, then* $\mathsf{UnSat}(\Sigma)$*.*

Proof. By induction on the length of the sequent calculus proof, it suffices to prove that every sequent rule is correct in the sense that if the set of formulae in the left-hand-side of each premise is unsatisfiable then the set of formulae of the conclusion is unsatisfiable. For the rules (Ctd) and $(\mathbf{F})$ is trivial. For the rest of the sequent rules the proof is similar to the proofs of the analogous tableaux rules in Lemma 1. □

Theorem 6 (Completeness)**.** *For any set of* CTL *formulae* Σ*, if* $\mathsf{UnSat}(\Sigma)$*, then there exists a sequent proof of* $\Sigma \vdash \mathbf{F}$*.*

Proof. If $\mathsf{UnSat}(\Sigma)$, then the tableau $\mathcal{A}^{sys}$ for Σ is fully expanded and closed. Each leaf of $\mathcal{A}^{sys}$ corresponds to an axiom obtained by application of the sequent rule (Ctd) or $(\mathbf{F})$. A sequent proof of $\Sigma \vdash \mathbf{F}$ is obtained from axioms by an application of the sequent rule corresponding to a tableau rule in the construction of $\mathcal{A}^{sys}$. □

4.4 Extending the Dual Method to ECTL

In this section we explain a (relatively easy) way to extend the CTL tableau method and its dual sequent calculus to the more expressive logic ECTL. This is achieved by adding the new tableau rules given in Figure 4.12 and their dual sequents rules in Figure 4.15. These rules correspond to the following logical equivalences for the basic modalities that extend CTL to ECTL:

$$\begin{array}{ll} \mathsf{E}\Box\Diamond\sigma \equiv \mathsf{E}\Diamond\sigma \wedge \mathsf{E}\circ\mathsf{E}\Box\Diamond\sigma & \mathsf{E}\Diamond\Box\sigma \equiv \mathsf{E}\Box\sigma \vee (\mathsf{E}\Diamond\sigma \wedge \mathsf{E}\circ\mathsf{E}\Diamond\Box\sigma) \\ \mathsf{A}\Box\Diamond\sigma \equiv \mathsf{A}\Diamond\sigma \wedge \mathsf{A}\circ\mathsf{A}\Box\Diamond\sigma & \mathsf{A}\Diamond\Box\sigma \equiv \mathsf{A}\Box\sigma \vee (\mathsf{A}\Diamond\sigma \wedge \mathsf{A}\circ\mathsf{A}\Diamond\Box\sigma) \end{array} \tag{4.9}$$

$$(\mathsf{Q}\Box\Diamond)\ \frac{\Sigma, \mathsf{Q}\Box\Diamond\sigma}{\Sigma, \mathsf{Q}\Diamond\sigma, \mathsf{Q}\circ\mathsf{Q}\Box\Diamond\sigma} \qquad (\mathsf{Q}\Diamond\Box)\ \frac{\Sigma, \mathsf{Q}\Diamond\Box\sigma}{\Sigma, \mathsf{Q}\Box\sigma \ \mid\ \Sigma, \mathsf{Q}\Diamond\sigma, \mathsf{Q}\circ\mathsf{Q}\Diamond\Box\sigma}$$

Figure 4.12: Additional tableau rules for ECTL

The rule $(\mathsf{Q}\Box\Diamond)$ is added to the set of α-rules and the rule $(\mathsf{Q}\Diamond\Box)$ is added to the set of β-rules. There are no additional β^+-rules. Indeed the eventualities $\mathsf{Q}\Diamond\sigma$ introduced by the rules in Figure 4.12 are CTL-modalities that are handled by the β^+-rules of the method for CTL.

We extend soundness and completeness results (for CTL) to ECTL. Firstly, we extend Lemma 1 to the rules in Figure 4.12.

Lemma 3. *For any* ECTL *set of state formulae* Σ *and any state formula* σ*:*

1. $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Box\Diamond\sigma\})$ *if and only if* $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Diamond\sigma, \mathsf{Q}\circ\mathsf{Q}\Box\Diamond\sigma\})$.
2. $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Diamond\Box\sigma)$ *if and only if* $\mathsf{Sat}(\Sigma, \mathsf{Q}\Box\sigma\})$ *or* $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Diamond\sigma, \mathsf{Q}\circ\mathsf{Q}\Diamond\Box\sigma\})$.

Proof. It follows by 'systematic' application of the semantic definitions of the modalities $\mathsf{Q}\Box\Diamond$ and $\mathsf{Q}\Diamond\Box$ given by the equivalences (4.9). □

As a consequence of Lemma 3, the soundness Theorem 1 easily extends to ECTL. For refutational completeness of ECTL, we firstly extend Definition 31 with the following additional conditions for a stage to be $\alpha\beta^+$-saturated:

Definition 32. *We say that a stage* $s = n_i, \ldots, n_j$ *in the* ECTL *systematic tableau for* Σ *is* $\alpha\beta^+$*-saturated if and only if it satisfies the conditions in Definition 31 and the following two additional conditions:*

7. *For all* $\mathsf{Q}\Box\Diamond\sigma \in \tau(s)$*:* $\{\mathsf{Q}\Diamond\sigma, \mathsf{Q}\circ\mathsf{Q}\Box\Diamond\sigma\} \subseteq \tau(s)$.
8. *For all* $\mathsf{Q}\Diamond\Box\sigma \in \tau(s)$*:* $\{\mathsf{Q}\Box\sigma\} \subseteq \tau(s)$ *or* $\{\mathsf{Q}\Diamond\sigma, \mathsf{Q}\circ\mathsf{Q}\Diamond\Box\sigma\} \subseteq \tau(s)$.

It is obvious that the conditions 7. and 8. are satisfied in any stage of the systematic tableau by construction. Using these conditions, it is routine to prove that $\mathcal{K}_H$, as defined in Lemma 2, satisfies the fact that: $\mathcal{K}_H, s_a, 0 \models \sigma$ for any $a \in \{0, \ldots, j\}$ and any formula of the form $\mathsf{Q}\Box\Diamond\sigma, \mathsf{Q}\Diamond\Box\sigma$ that belongs to $L(s_a)$. Therefore, refutational completeness (i.e. Theorem 2) extends to ECTL. Finally, for the termination result (see the proof in

Theorem 3), it suffices to ensure that the rules in Figure 4.12 do not affect the behaviour of the β^+-rules on the selected eventualities in the sense that Proposition 6 is preserved. Furthermore, although each application of a rule in Figure 4.12 introduces a new $\mathsf{Q}\Diamond\sigma$, the simplification rule $(\sqsubset \mathsf{Q}\mathcal{U})$ (see (4.8) in Section 4.4) ensures that any occurrence of an eventuality is subsumed by its contextualised variant. Therefore, Proposition 6 holds and hence Theorem 3 extends to ECTL.

Next, we present an example of systematic tableau for an ECTL input where additional rules for ECTL are applied, along with CTL rules.

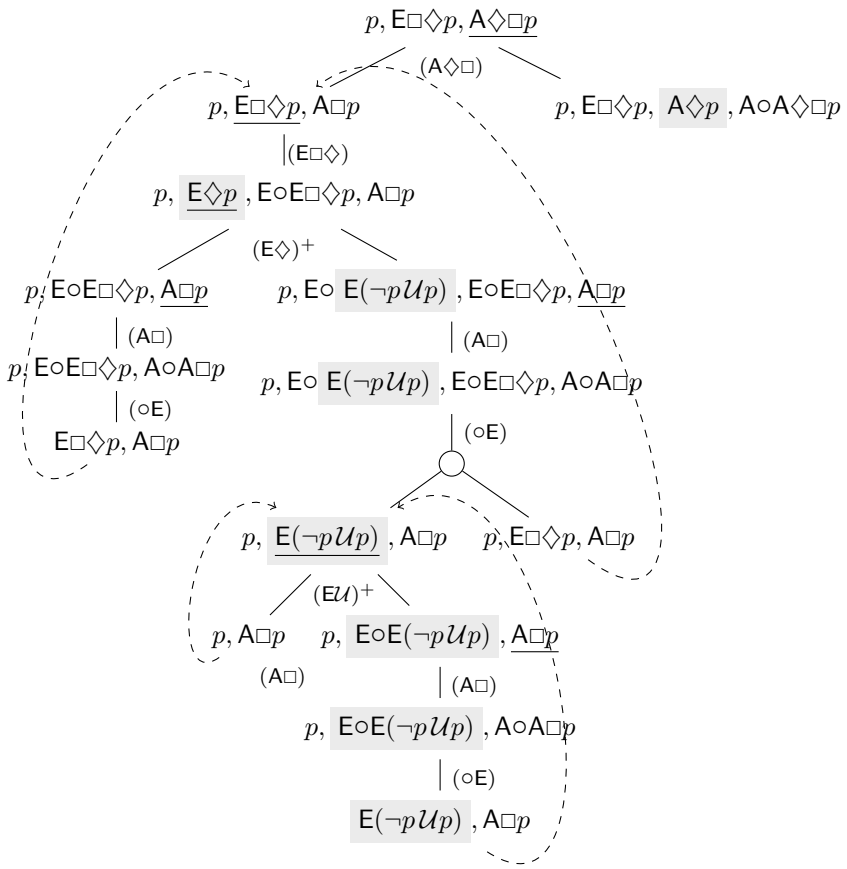


Figure 4.13: ECTL systematic tableau for $\{p, \mathsf{E}\Box\Diamond p, \mathsf{A}\Diamond\Box p\}$.

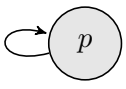


Figure 4.14: Graphic representation of the model for $\{p, \mathsf{E}\Box\Diamond p, \mathsf{A}\Diamond\Box p\}$.

Example 23. *Figure 4.13 partially shows an open tableau with the application of two of the rules added to extend* CTL *to* ECTL. *We just outline some branches to illustrate*

how the ECTL*-rules* (A$\Diamond\Box$)*, and* (E$\Box\Diamond$) *(in Figure 4.12) are applied. The tableau for the right-most node, which is also open, is not depicted for space reasons. Note that our algorithm constructs just the left-most branch to decide that the label of the root is satisfiable returns a very simple model depicted in Figure 4.14, which is a cyclic Kripke structure with the single fullpath* $\{p\}^w$*.* ∎

The calculus $\mathcal{C}_{\vdash}$ is also extended by adding the sequent rules that are dual to the tableau rules in Figure 4.12. These additional sequent rules are given in Figure 4.15 and the duality of the extended tableau and sequent calculus for ECTL is trivial.

$$(\mathsf{Q}\Box\Diamond)\ \frac{\Sigma, \mathsf{Q}\Diamond\sigma, \mathsf{Q}\circ\mathsf{Q}\Box\Diamond\sigma \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Box\Diamond\sigma \vdash \mathbf{F}}$$

$$(\mathsf{Q}\Diamond\Box)\ \frac{\Sigma, \mathsf{Q}\Box\sigma \vdash \mathbf{F} \quad \Sigma, \mathsf{Q}\Diamond\sigma, \mathsf{Q}\circ\mathsf{Q}\Diamond\Box\sigma \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Diamond\Box\sigma \vdash \mathbf{F}}$$

Figure 4.15: Additional sequent rules for ECTL.

5. MomoCTL: IMPLEMENTATION AND EXPERIMENTATION

A useful comparison of five satisfiability procedure implementations for CTL can be found in [48] (two of these procedures are tableau-based). The first method [12] is a two-pass tableau which, in the first pass creates a cyclic graph. In this graph, a 'bad loop' is a loop containing some eventuality that is not fulfilled along it. Therefore, in the second pass, 'bad loops' are pruned. The second method [1, 47] is a single-pass tableaux decision procedure based on Schwendimann's one-pass procedure for PLTL [79]. This tableau method uses an additional mechanism for collecting information on the set of formulae in the nodes and passing it to subsequent nodes along branches. The information on previously generated nodes helps detecting 'bad loops' without constructing the whole graph.

We have implemented a prototype called MomoCTL which, when it receives a set of formulae Σ as input, decides whether Σ is satisfiable or unsatisfiable. MomoCTL also returns a Kripke structure that certifies the satisfiability of Σ or a sequent calculus proof that certifies that Σ can be refuted. Recall that this was our aim: a deductive technique for branching-time logics with a reasoning mechanism capable of providing both formal proofs and models. The set Σ could contains any set of CTL formulae in NNF, also a representation of a transition system and a negated property. That is why we can also consider MomoCTL as a Certified Model Checker.

For the implementation we have used Dafny. Dafny is a language and a tool capable of implementation and verification. This makes Dafny an interesting option to bring reliability to the implementation of the method.

After implementing the method, MomoCTL has been tested against the benchmarks used in [48], available in `http://users.cecs.anu.edu.au/~rpg/CTLComparisonBenchmarks/`. We also compare our results with the Gore's CTL-prover presented in [1, 47], which we call TreeTab. Finally, we note that, to the best of our knowledge, there has been no explicit formulation of a tableau (one or two pass) method for ECTL.

The chapter consists of 3 sections: Section 5.1 introduces Dafny: the main ideas, the IDE, and features. The complete algorithm is explained in detail in Section 5.2, as well as its implementation using the Dafny language. Finally, Section 5.3 compares the results obtained by MomoCTL and TreeTab when running the set of benchmarks.

5.1 Dafny Language

Dafny [60] is a program verifier that can be used to verify the functional correctness of programs, enabling the user to create and verify both specifications and implementations. The Dafny specification language extends first-order logic with algebraic data types, inductive predicates, generic types, abstracting and refining modules, assertions, and many others built-in specification features that makes Dafny a good candidate for

our work. In this section, we briefly introduce the main notions of Dafny that facilitate the understanding of the implementation and experimentation explained in 5.2.2 and 5.3 respectively.

The basic unit of a Dafny program is the `method`. A method is a piece of executable code with a head where multiple named parameters and multiple named results are declared. Dafny has also built-in specification constructs for assertions, such as `requires` for preconditions, `ensures` for postconditions, and `assert` for inline assertions. Using `requires` and `ensures` we specify methods and lemmas. Assertions specify properties that are satisfied at some point. Assertions are mainly used to provide hints to the verifier. In other words, once the assertion is proved, it turns into a usable property for completing the proof. Indeed, "`assert` φ" tells Dafny to check that φ holds and to use the condition φ (as a lemma) to prove the properties beyond this point.

Dafny distinguishes between ghost entities and executable entities. Ghost entities are used only during verification; the compiler omits them from the executable code. The `lemma` declarations are like methods, but no code is generated for them, i.e. a lemma is equivalent to a ghost method. The body of a lemma is its proof. For lemma proofs, Dafny provides a special notation that is easy to read and understand: calculations that were presented by [64]. A calculation in Dafny is a statement that proves a property. This notation was extracted from the calculational method introduced by [9], whereby a theorem is established by a chain of formulae, each transformed in some way into the next. The relationship between successive formulae (for example, equality, implication, double implication, etc.) is indicated, or it can be omitted if it is the default relationship (equality). In addition, the hints (usually asserts or lemma calls) that justify a step can also be indicated (in curly brackets after the relationship). Calculations are written inside the environment `calc{ }`.

Dafny also provides built-in immutable types, such as `set`, `multiset`, `map`, and `seq` – which respectively denote the finite collections types of sets, multisets, maps, and sequences – that are very useful in specification. These built-in types are equipped with the usual operations, including set comprehension expressions:

```
set x1:T1,x2:T2,...,xi:Ti | P(x1,x2,...,xi) • E(x1,x2,...,xi)
```

for defining the set of all values given by the expression `E(x1,x2,...,xi)` for all tuples `(x1,x2,...,xi)` such that `P(x1,x2,...,xi)`.[1]

Dafny also offers user-defined specification constructs (which are ghost), such as `function` and `predicate` that can be defined using well-founded inductive definitions, built-in immutable types, polymorphic algebraic `datatype`s, `inductive predicate`s, `co\-inductive predicate`s, etc.

The Dafny specification constructor `inductive predicate` (resp. `co-inductive predicate`) was introduced by [62] and allows for the definition of a predicate as an extreme solution: a least fixpoint (resp. a greatest fixpoint) of a set of recursive rules.

For defining variables in methods, functions and proofs, Dafny includes a *let-such-that statement*: `var x :| P` that looks like a variable declaration but it includes a boolean expression `P` after the so-called *Hilbert epsilon operator* or *choose operator* `:|`. A statement `var x :| P` can be read as assigning to `x` any value that satisfies `P`. The verifier must

[1] For easy reading, in the Dafny code snippets, we show the usual mathematical symbols, instead of real Dafny notation. For example, we show • for :: (such that), ∧ for `&&`, ∀ for `forall`, etc.

be able to prove that a value satisfying condition `P` exists, but not to construct it. Then, for verification purposes, the variable `x` will stand for any value that fulfils `P`. Consequently, these let-such-that statements are not directly compilable into executable code. A discussion of the implementation problems of this operator in the Dafny language is given in [61] and [63] provides an example of a non-compilable function for which a compilable version is constructed.

The Dafny integrated development environment (IDE) is an extension of Microsoft Visual Studio (VS). From Dafny version 2.1.1 it is included in Visual Code Environment. The IDE is designed to reduce the effort required by the user to make use of the system. The IDE runs the program verifier in the background and provides design time feedback. Assertions are sent to the SMT solver Z3 [30] (a fully automatic theorem prover) to check its satisfiability, which will be reported to the Dafny user. Assertion violations in lemma proofs are reported, as well as verification errors, along with different information such as locations (of the properties) related to the error. The interested reader is referred to [65] for more information on the various ways in which the Dafny IDE helps to build both lemma proofs and verified software.

Like in other proof assistants (e.g. Isabelle and Coq) and verifiers (e.g. Why3 and KeY), Dafny allows proofs to be written in different styles and with different levels of description. Therefore, making proofs readable and easy to check by humans is part of the Dafny user's job.

Dafny is able to export executable files (`.exe`), libraries (`.dll`) and .Net source code (`.cs`) once the automatic verification has succeeded and all lemma have been proved. Also, from Dafny 3.0, users can export JavaScript, Java and C++ code. Precisely, as it is detailed in Section 5.2.3, we export Java code from the implementation of our prototype MomoCTL.

5.2 *Implementation of* MomoCTL

This section presents the implementation of MomoCTL. In Section 5.2.1 we explain the details of the complete algorithm. Section 5.2.2 presents the general structure of the Dafny code. Finally, Section 5.2.3 discusses the created console application and its integration with Dafny.

5.2.1 *Algorithm*

First we extend the algorithm $\mathcal{A}^{sys}$ (see Algorithm 7) to $\mathcal{E}$ (see Algorithm 9), so that depending on the value of *is_closed*, $\mathcal{E}$ is able to return either a proof or a model, but not both.

We can easily construct a model from an open tableau. Models are represented as trees. Each branch of the tree corresponds to a branch of the open tableau, which ends with a leaf pointing to some ancestor in the branch. Each non-leaf node in a branch of the tree is the set of atoms appearing in one of the stages of the corresponding branch in the tableau. Example 24 displays the model returned by $\mathcal{E}$ when executed over a set of formulae.

Algorithm 9: $\mathcal{E}$
Input = Σ: set of formulae, b: Branch, P: Proof, M: Model.
Output = *is_closed*: boolean, b': Branch, P': Proof, M': Model.

```
1  if Σ = ∅ then
2  |  is_closed, b' := false, b;
3  |  M' := EmptyLeaf_Model;
4  else if (F) applies to Σ then
5  |  is_closed, b' := true, b;
6  |  P' := ProofLeaf((F), Σ);
7  else if (Ctd) applies to Σ then
8  |  is_closed, b' := true, b;
9  |  P' := ProofLeaf((Ctd), Σ);
10 else if Σ ⊆ τ(s) for some stage s in b & Ev_Covered(b) then
11 |  is_closed, b' := false, b;
12 |  M' := ModelLeaf(s, b);
13 else if β+_is_applicable(Σ) then
14 |  select_eventuality(Σ) ;
15 |  is_closed, b', P', M' := apply_β+_rule(Σ, b, P, M);
16 else if α_β_is_applicable(Σ) then
17 |  is_closed, b', P', M' := apply_α_β_rule(Σ, b, P, M);
18 else // is_elementary(Σ)
19 |  Let Σ = Φ, A∘(Ψ), E∘σ_1, ..., E∘σ_k where k ≥ 0;
20 |  if k ≥ 1 then
21 |  |  Let Σ_i = Ψ, σ_i for all 1 ≤ i ≤ k ;
22 |  |  n := k;
23 |  else
24 |  |  Σ_1, n := Ψ, 1
25 |  end
26 |  i, is_closed :=0, false ;
27 |  ListModels := EmptyList ;
28 |  while is_closed = false & i < n do
29 |  |  i := i + 1 ;
30 |  |  is_closed, b', P_i, M_i := E(Σ_i, b + [[Σ_i]], P, M);
31 |  |  if is_closed = false then add(M_i, ListModels);
32 |  end
33 |  if is_closed then
34 |  |  if k = 0 then P' := ProofNode((∘A), Σ, [P_i]) ;
35 |  |  else P' := ProofNode((∘E), Σ, [P_i]) ;
36 |  else
37 |  |  M' := ModelNode(Atoms(Σ), ListModels)
38 |  end
39 end
```

The representation of a proof is also a tree whose nodes are steps of the proof. Thus, each leaf contains the inconsistent sequent and name of the premise-free rule (*Ctd*) or

($\mathbf{F}$) that applies to it. Non-leaf nodes contain the sequent Σ and the name of the rule (R) that applies to prove that $\Sigma \vdash \mathbf{F}$. Its children contain the proofs of the subgoals that result from applying (R) to $\Sigma \vdash \mathbf{F}$.

In the initial call to algorithm $\mathcal{E}$, the proof and the model are both empty, therefore to decide the satisfiability of a set Σ and generate a proof or model, we call $\mathcal{E}(\Sigma, [[\Sigma]], EmptyProof, EmptyModel)$. Note also that the branch contains just one stage that consists of Σ.

Lines 1-12 are the non-recursive cases. In line 3, the algorithm returns a model of the empty set of formulae, which is a linear model that consists of an empty state (every atom is false in it) that is infinitely repeated. In lines 6 and 9, it returns just one sequent which is a proof by one of the premise-free rules ($\mathbf{F}$) and (Ctd), respectively. In line 10, the algorithm detects that Σ gets an eventuality-covered loop at the stage s in the current branch b. Therefore, the algorithm creates a branch of the potential model with the atoms that are loaded (as formulae) at each stage of b from s to the current node. This branch terminates by a leaf which is just a pointer to stage s.

Recursive constructions of proofs and models, are performed in lines 15, 17 and 26-36 of Algorithm 9. The calls to $apply\_\beta^+\_rule$ and $apply\_\alpha\_\beta\_rule$ apply the corresponding rule to a designated formula in the node label, and recursively call $\mathcal{E}$ with the children produced by the applied rule. For example, see Algorithm 10, to apply the β^+-rule $(\mathsf{E}\mathcal{U})^+$ to $\Phi = \Sigma, \mathsf{E}(\sigma_1 \mathcal{U} \sigma_2)$. The procedure $\mathcal{E}$ is called on parameters $\Sigma_1 = \Sigma \cup \{\sigma_2\}$, the extended branch b_1, with the proof P, and the model M. Both P and M where the parameters of the call $apply\_\beta^+\_rule(\Sigma, b, P, M)$ (in line 15). If this recursive call returns in is_closed the value true, $\mathcal{E}$ is called with $\Sigma_2 = \Sigma \cup \{\sigma_1, \mathsf{E}((\sigma_1 \wedge \sim \Sigma')\mathcal{U}\sigma_2)\}$ and the value of this call is assigned to is_closed.

Algorithm 10: Apply $(\mathsf{E}\mathcal{U})^+$ to $\Phi = \Sigma, \mathsf{E}(\sigma_1 \mathcal{U} \sigma_2)$ that returns a proof, when it exists.

```
1  Σ := Φ \ {E(σ1 U σ2)};
2  is_closed, b', P1, M' := E(Σ1, b1, P, M) where
3                                  Σ1 = Σ ∪ {σ2} and b1 = update(b, Σ1);
4  if is_closed then
5  |  is_closed, b', P2, M' = E(Σ2, b2, P, M) where
6  |                               Σ2 = Σ ∪ {σ1, E((σ1 ∧ ∼Σ') U σ2)} and
7  |                               b2 = update(b, Σ2);
8  |  if is_closed then
9  |  |  P' := ProofNode((EU)^+, Σ, [P1, P2]);
10 |  end
11 end
```

The recursive calls for Σ_1 and Σ_2 also construct either a proof or a model depending on the value of is_closed. When both calls report that is_closed is true, then P_1 contains a proof for the sequent Σ_1 and P_2 contains a proof for the sequent Σ_2, then Algorithm 10 (line 9) returns in P' a proof for Σ whose last step is the application of the rule $(\mathsf{E}\mathcal{U})^+$ to the sequents proved by P_1 and P_2. Consequently, when at least one of the two calls (for Σ_1 and Σ_2) reports that the tableau is not closed, the tableau for Σ is open and Algorithm 10 returns the model M'.

The application of α, β and β^+ rules does not construct models, they only load atoms in stages that will be collected and structured according to the application of the next-state rules when a cycle is detected. In lines 26-36 of Algorithm 9, one next-state rule is applied to an elementary set Σ with a number $k \geq 0$ of formulae $\mathsf{E}\circ\sigma_i$. If $k = 0$ then $n := 1$ and, depending on the value of *is_closed* for the only recursive call, the algorithm constructs a proof for Σ from the returned proof P_1 of Σ_1 whose last step is the application of the rule ($\circ\mathsf{A}$). Otherwise, if $k > 0$, then $n := k$ and there is a recursive call for each AND-child of Σ, namely $\Sigma_1, \ldots, \Sigma_n$. If some of these calls for Σ_i returns in *is_closed* the value true, then it also returns in P_i a proof for Σ_i. Therefore it constructs in P' a proof for Σ that applies the rule ($\circ\mathsf{A}$). Otherwise, it collects a list a models $M_1, \ldots, M_n$, one for each of the $n \geq 1$ calls with respective parameters $\Sigma_1, \ldots, \Sigma_n$. Note that each call returns that *is_closed* is false. Then, this list of $n \geq 1$ models is used to compose a model of Σ whose root contains the atoms of Σ and this root has n children $M_1, \ldots, M_n$. Recall that the leaves of our models are either empty leaves (line 3) or pointers to some previous stage in the branch (line 12).

Example 24. *For the following set of formulae Σ:*

$$\{\ \mathsf{A}\Box(a \to \neg(b \vee c)),\ \mathsf{A}\Box(b \to \neg(a \vee c)),\ \mathsf{A}\Box(c \to \neg(b \vee a)),\\ \mathsf{E}\Diamond\mathsf{A}\Box a,\ \mathsf{E}\Diamond\mathsf{A}\Box b,\ \mathsf{E}\Diamond\mathsf{A}\Box c\ \}$$

our prototype returns the following Kripke structure as a model of Σ that certifies its satisfiability.

```
State 0: {}
Subtree 1 of State 0
---State 1: {c} --> cycle to State 1
Subtree 2 of State 0
---State 1: {b} --> cycle to State 1
Subtree 3 of State 0
---State 1: {a} --> cycle to State 1
```

Figure 5.1 shows a graphical representation of this model.

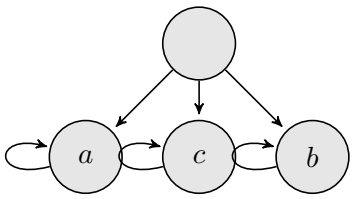


Figure 5.1: Graphic representation of the Kripke structure returned by the prototype.

Algorithm 9 is the basis of our implementation, however, we have implemented some improvements mainly aimed at the efficiency and the usability in CMC. The formulae of the form $\mathsf{A}\Box\varphi$, that we call *global invariants*, are the most common in a model checking problem. In model checking, the formula that characterises a computation tree of a transition system is a (usually large) conjunction (collection) of global invariants. To provide shorter, clearer and readable proofs in CMC applications, we take advantage of

the fact that, for each invariant $\mathsf{A}\Box\sigma$, either this formula itself or $\mathsf{A}\circ\mathsf{A}\Box\sigma$ are in the node of the tableau, and both formulae are at every stage. Hence, proofs are easier to follow if we discharge all global invariants (and all $(\mathsf{A}\circ\varphi)$ where φ is a global invariant). Indeed, the user could consult the transition system specification to follow the reasoning, though we specify in the proof when a rule is applied to an invariant formula (see sequent 2 in Example 25). Moreover, we do not load global invariants along the stages of the current branch.

Example 25. *Consider the set of formulae* $\Sigma = \{p, \mathsf{A}\Box(p \to \mathsf{A}\circ p), \mathsf{E}\Diamond\neg p\}$. *In the model checking framework the first formula could be seen as the initial state condition, the second formula as the model specification, and the third as the negation of the property* $\mathsf{A}\Box p$, *which the user is asking to check. Our prototype on this input* Σ *returns the expected result – that* Σ *is unsatisfiable (therefore, the property* $\mathsf{A}\Box p$ *holds in any model of* $\{p, \mathsf{A}\Box(p \to \mathsf{A}\circ p)\}$*) and also gives the following 'detailed' proof as certificate:*

> `0.` $\mathsf{E}\Diamond\neg p, p$. **apply** $(\mathsf{E}\Diamond^{+})$
> `-1.` $\neg p, p$. **by** *(Ctd)*
> `-2.` $\mathsf{E}\circ(\mathsf{E}(\neg p \mathcal{U} \neg p)), p$. **apply** $(\mathsf{A}\Box)$ **to** *Inv:* $\mathsf{A}\Box(\neg p \vee \mathsf{A}\circ p)$
> `--3.` $\neg p \vee \mathsf{A}\circ p, \mathsf{E}\circ(\mathsf{E}(\neg p \mathcal{U} \neg p)), p$. **apply** $(\vee)$
> `---4.` $\mathsf{E}\circ(\mathsf{E}(\neg p \mathcal{U} \neg p)), \neg p, p$. **by** (Ctd)
> `---5.` $\mathsf{E}\circ(\mathsf{E}(\neg p \mathcal{U} \neg p)), \mathsf{A}\circ p, p$. **apply** $(\circ\mathsf{E})$
> `----6.` $\mathsf{E}(\neg p \mathcal{U} \neg p), p$. **apply** $(\mathsf{E}\mathcal{U}^{+})$
> `-----7.` $\neg p, p$. **by** (Ctd)
> `-----8.` $\mathsf{E}\circ(\mathsf{E}(\neg p \mathcal{U} \neg p)), \neg p, p$. **by** (Ctd)

It is worth to note that sequent 2 is derived by the application of $(\mathsf{E}\Diamond^{+})$ *to sequent 0. In 0, the context of* $\mathsf{E}\Diamond\neg p$ *is just* $\{p\}$. *This context cannot be repeated from the next-state onwards until the eventuality would be fulfilled. Therefore, the contextualised variant* $\mathsf{E}(\neg p \mathcal{U} \neg p)$ *(of* $\mathsf{E}\Diamond\neg p$*) should be fulfilled from the next-state on.*

Our prototype provides proofs with two grades of granularity: small-step proofs and big-step proofs. A small-step proof includes all rule applications (as the proof in Example 25). A big-step proof includes only the sequents before the application of a next-step rule, i.e. the elementary sets that have been refuted. In particular, the big-step version for Example 25 consists of just one elementary sequent:

> `---5.` $\mathsf{E}\circ(\mathsf{E}(\neg p \mathcal{U} \neg p)), \mathsf{A}\circ p, p$.

Big-step proofs are useful when proofs are very long and especially in CMC, when the user wants just to see the evolution of the system that leads to contradiction. In the previous example, the unique sequent informs that the system evolves to satisfy $p, \mathsf{A}\circ p$ while it should satisfy $\mathsf{E}\circ\mathsf{E}(\neg p \mathcal{U} \neg p)$ (to satisfy the initial eventuality $\mathsf{E}\Diamond\neg p$), and that this leads to contradiction. Next, we show the big-step version of a larger proof.

Example 26. *The following set of formulae is obtained by substituting in Example 24 the formula* $\mathsf{E}\Diamond\mathsf{A}\Box a$ *by* $\mathsf{A}\Box\mathsf{E}\Diamond\mathsf{A}\Box a$, *which makes it unsatisfiable*

$$\{\ \mathsf{A}\Box(a \to \neg(b \vee c)),\ \mathsf{A}\Box(b \to \neg(a \vee c)),\ \mathsf{A}\Box(c \to \neg(b \vee a)),\\ \mathsf{A}\Box\mathsf{E}\Diamond\mathsf{A}\Box a,\ \mathsf{E}\Diamond\mathsf{A}\Box b,\ \mathsf{E}\Diamond\mathsf{A}\Box c\ \}$$

The small-step proof provided by our prototype has about 350 sequents, however the big-step proof contains the following five sequents:

> `--76.` $\mathsf{E}\circ\mathsf{E}(\mathbf{F}\mathcal{U}\mathsf{A}\Box b), \mathsf{E}\circ(\mathsf{E}\Diamond\mathsf{A}\Box a), \mathsf{A}\circ\mathsf{A}\Box c, \neg b, \neg a, c$
> `---145.` $\mathsf{E}\circ\mathsf{E}((\mathsf{A}\Box\mathsf{E}\Diamond\neg b)\mathcal{U}\mathsf{A}\Box c), \mathsf{E}\circ\mathsf{E}\Diamond\mathsf{A}\Box a, \mathsf{A}\circ\mathsf{A}\Box b, \neg c, \neg a, b$
> `--240.` $\mathsf{E}\circ\mathsf{E}((\mathsf{A}\Box\mathsf{E}\Diamond\neg b)\mathcal{U}\mathsf{A}\Box c)), \mathsf{E}\circ\mathsf{E}\Diamond\mathsf{A}\Box b, \mathsf{E}\circ\mathsf{E}\Diamond\mathsf{A}\Box a, \neg c, \neg b, \neg a$
> `----278.` $\mathsf{E}\circ\mathsf{E}((a \vee b)\mathcal{U}\mathsf{A}\Box a), \mathsf{A}\circ\mathsf{A}\Box c, \neg b, \neg a, c$
> `-----345.` $\mathsf{E}\circ\mathsf{E}(\mathbf{F}\mathcal{U}\mathsf{A}\Box c), \mathsf{E}\circ\mathsf{E}\Diamond\mathsf{A}\Box a, \mathsf{A}\circ\mathsf{A}\Box\mathsf{E}\Diamond\neg b, \neg c, \neg b, \neg a$

Since all sequents of a big-step proof are elementary sets of formulae, to which the appropriated next-state rule is applied, we do not report that implicit information. In the proof above, ($\circ\mathsf{E}$) *is applied to the indicated five sequents because all these sequents have one or more* $\mathsf{E}\circ$ *formulae. The two first lines (steps 76 and 145) show that constructing a path to fulfill* $\mathsf{A}\Box b$*, while* $\mathsf{A}\Box c$ *is satisfied, is not possible. The last three lines (steps 240, 278 and 345) show that constructing a path to fulfill* $\mathsf{A}\Box c$ *is also impossible, because once* $\mathsf{A}\Box c$ *is fulfilled,* $\mathsf{A}\Box a$ *cannot be fulfilled.* ∎

We have implemented the sets of formulae by ordered sequences of formulae. For that we have defined an ad-hoc ordering on formulae (in NNF) to quickly detect a selected eventuality and to check if a sequence is elementary by just looking at the first element. In addition, we exploit the ordering to detect more quickly that a sequence is not a subsequence of another sequence, hence branches also keep ordered sequences of stages, in particular to detect cycles. We have also implemented some rules of subsumption (such as ϕ subsumes $\phi \vee \psi$) in sequences and in the construction of the contextualised variants of eventualities –the latter is very important for the feasibility of our method. Of course, we have implemented the subsumption-like simplification rule (4.8) (See Section 4.3.1) by which any contextualised variant subsumes the original eventuality.

Our systematic tableau construction produces many repetitions of subtrees. We prevent to repeat the refutation (closed tableau) of any sequent that is a child of an elementary sequent. When one of them is refuted, it is loaded in a set called APA ('As Proved Above'). In the output proofs, we use the word APA to indicate that a sequent has been previously refuted. Moreover, we consider as refuted any sequent that is a superset of another already refuted. In other words, if a sequent $\Sigma \in$ APA and $\Sigma \subseteq \Sigma'$, then the sequent Σ' is immediately classify as refuted. Since loading all refuted sequents is really costly, we choose some sequents as candidates to be loaded in APA. This choice clearly determines the efficiency of the prototype. In the current prototype, we only add to APA the sequents produced after the application of the next-state rule.

In order to also keep data on the nodes whose tableaux have already been categorized as open, we employ a set called SAT. Dually to APA, if a set of formulae $\Sigma \in$ SAT and $\Sigma' \subseteq \Sigma$, then Σ' is immediately classified as open. Additionally, we know that the model created for Σ is likewise a model for Σ'. The candidates that occur after the next-state rule is applied are also those that are stored in SAT. Further experimentation is needed to evaluate and compare the performance of the implementation with different selection criteria of the candidates for the APA and for SAT.

As we have mentioned above, most of the code (written in Java) is automatically generated from the language Dafny. Though Dafny is a program verifier, we note that our implementation is only partially verified, i.e. only some crucial properties have been verified by the time of writing this thesis. For example, the ordering in formulae has been proved to be total. The methods that exploit this ordering to perform more efficient operations in sequences (e,g, insertion, deletion) have been proved to preserve the order. Other methods that decide properties of sequences more efficiently thanks to

the order (e.g. elementarity check) have also been proved correct. The verification of the functional correctness of our prototype remains as an encouraging and challenging future work.

5.2.2 General Structure of Dafny Code

The Dafny implementation is structured in the following modules: `Parsing CTL to NNF Sequents`, `Auxiliaries Tableau`, `Tableau`, `Print Models`, `Print Proofs` and `Tableau sat unsat` Each module corresponds to one .dfy file.

Module `Parsing_CTL_to_NNF_Sequents` contains data definitions, method, functions and lemmas to represent, manage and parse formulae in NNF. The following Dafny code represents the datatype `NNF_Formula`:

```
datatype NNF_Formula =   F
                       | T
                       | V(prop: string)
                       | NegV(prop: string)
                       | EX(f: NNF_Formula)
                       | AX(f: NNF_Formula)
                       | EG(f: NNF_Formula)
                       | AG(f: NNF_Formula)
                       | ER(f1: NNF_Formula,f2: NNF_Formula)
                       | AR(f1: NNF_Formula,f2: NNF_Formula)
                       | EF(f: NNF_Formula)
                       | AF(f: NNF_Formula)
                       | EU(f1: NNF_Formula,f2: NNF_Formula)
                       | AU(f1: NNF_Formula,f2: NNF_Formula)
                       | EUsel(f1: NNF_Formula,f2: NNF_Formula)
                       | AUsel(f1: NNF_Formula,f2: NNF_Formula)
                       | Or(f1: NNF_Formula,f2: NNF_Formula)
                       | And(f1: NNF_Formula,f2: NNF_Formula)
```

In the implementation we have defined the constants `T` and `F`, the constructor `V` for a variable, `NegV` for the negation of a variable, `E` and `A` for exists and forall respectively, `X` for next, `G` for always, `U` for until, `F` for eventually and `R` for release. `EUsel` and `AUsel` have been used to represent the selected `EU` and `AU` respectively.

Module `Auxiliaries_Tableau` is created to represent in Dafny all the structures and functions that help to carry out the tableau and sequents. It defines for instance what a branch, a model and a proof are.

```
type Branch = seq<seq<NNF_Formula>>

datatype Model = EmptyM
               | LeafM(i: int)
               | NodeM(At: seq<NNF_Formula>,Seq: seq<Model>)

datatype Proof = EmptyP
               | LeafP(string,seq<NNF_Formula>)
               | NodeP(string,Seq: seq<NNF_Formula>,seq<Proof>)
```

It includes also methods, for example, to check contradictions in a sequence of formulae, to check whether a sequence of formulae is subset of another one and to add a a set of formulae to a context:

```
predicate method is_Ctd(Sigma: seq<NNF_Formula>)

method is_subset(Sigma1: seq<NNF_Formula>,
        Sigma2: seq<NNF_Formula>)
    returns (subset: bool)

method addContextWithSubsumption(
        previous: set<set<NNF_Formula>>,
        current: set<NNF_Formula>)
    returns (newC: set<set<NNF_Formula>>)
requires current ≠ {} ∧ ∀ S • S in previous ⟹ S ≠ {}
ensures ∀ S • S in newC ⟹ S ≠ {}
```

Note that there are `requires` and `ensures` in `addContextWithSubsumption`. Dafny checks that `addContextWithSubsumption` can only be used if `current` is not an empty set and if all sets of formulae of `previous` are not empty sets. Once the `requires` is fulfilled, it can be ensured that all sets of the resulting context (`newC`) are not empty.

Modules `Print_Models` and `Print_Proofs` contain auxiliary methods to treat and print models and proofs in order to be readable for users. These are some examples:

```
predicate method is_subModel(M1: Model,M2: Model)

method printModel(M: Model)

method printBranch(B: Branch)

method printSequent(Sigma: seq<NNF_Formula>)
```

Module `Tableau` contains method `is_refutable?`, which is the most important of the whole system. The method `requires` an ordered sequences (`Sigma` and `Inv`) and a non-empty ordered branch (`B`). It `ensures` that the resulting branch (`B'`) is ordered. Details of the method are shown in Section 5.2.1.

```
method is_refutable? (
        Sigma: seq<NNF_Formula>, //current sequent to be refuted
        B: Branch, //current branch
        M: Model, //current model
        n: nat, //CALL DEPTH
        proof: Proof, //current proof
        APA: set<seq<NNF_Formula>>,
        //As Proved Above (set of elementary sequents)
        Inv: seq<NNF_Formula>,
        //Input formulae of the form AG(_)
        SAT: set<(seq<NNF_Formula>,Model)>
        //Sequents already proved satisfiable.
        )
```

```
returns (b:bool, //T, if Sigma has been refuted, otherwise F
        B':Branch, //new branch to be explored
        M':Model, //new model
        comp:int,
        //companion node (≥0) of the last node in the
        //current branch, -1 if there is not cycle
        proof':Proof,
        APA': set<seq<NNF_Formula>>,
        SAT': set<(seq<NNF_Formula>,Model)>
        )
requires is_ordered(Sigma) ∧ is_ordered(Inv)
requires B ≠ []
requires ∀ k • 0 ≤ k < |B| ⟹ is_ordered(B[k])
ensures ∀ k • 0 ≤ k < |B'| ⟹ is_ordered(B'[k])
```

Finally we created an adaption of `Tableau` in order not to generate neither proof nor models and to only have a SAT/UNSAT answer. This module is `Tableau_Sat_Unsat`. The main method is `is_unsat?`.

```
method is_unsat? (
        Sigma:seq<NNF_Formula>, //current sequent to be refuted
        B:Branch, //current branch
        APA: set<seq<NNF_Formula>>,
        //As Proved Above (set of elementary sequents)
        Inv:seq<NNF_Formula>, //Input formulae of the form AG(_)
        n:nat, //CALL DEPTH
        SAT: set<(seq<NNF_Formula>,nat)>
        )
returns (b:bool, //T, if Sigma has been refuted, otherwise F
        B':Branch, //new branch to be explored
        comp:int
        //companion node (≥0) of the last node in the
        //current branch, -1 if there is not cycle
        APA': set<seq<NNF_Formula>>,
        SAT': set<(seq<NNF_Formula>,nat)>
        )
requires is_ordered(Sigma) ∧ is_ordered(Inv)
requires B ≠ []
requires ∀ k • 0 ≤ k < |B| ⟹ is_ordered(B[k])
ensures ∀ k • 0 ≤ k < |B'| ⟹ is_ordered(B'[k])
```

The `is_unsat?` method is used when running the benchmarks presented in Section 5.3.

5.2.3 *Console Application and Use of the Prototype*

Dafny does not include the ability to read files among its functionalities. This makes it impossible for us to use Dafny directly to read the sets of input formulae. To avoid this, we use the functionality that Dafny has to generate code automatically [60]. We generate Java code of all the modules presented in Section 5.2.2. The general overview of the whole application is shown in Figure 5.2.

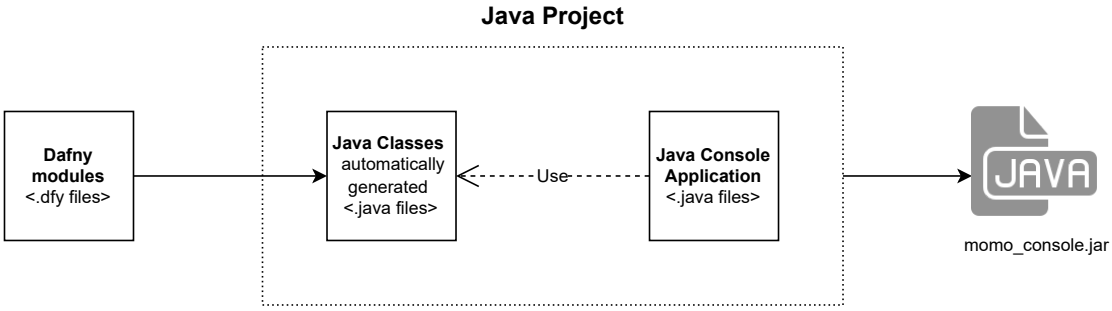


Figure 5.2: General overview of momo_console Application

After generating the code from Dafny, we create a Java class (not verified and apart from the generated code) that is able to read files and parse formulae. The sole purpose of this module is to provide users with the ability to read external files. The most important method of the module takes as input a file, reads the file line by line, creates `NNF_Formulae` by parsing them, and returns the obtained set of formulae.

All types of variables except `FileInputStream`, `BufferedReader`, and `String` comes from the code exported from Dafny. The method used to parse a line also comes from there. The code of this method is showed below (avoiding error control):

```
//Create an empty set of formulae
dafny.DafnySequence<? extends NNF__Formula> fseq =
    dafny.DafnySequence.<NNF__Formula>
    empty(NNF__Formula._typeDescriptor());
// Open the file
FileInputStream fstream =
    new FileInputStream(file.getAbsolutePath());
BufferedReader br =
    new BufferedReader(new InputStreamReader(fstream));
//Read File Line By Line
String line;
while ((line = br.readLine()) != null)   {
    //Create empty "string"
    dafny.DafnySequence<? extends Character> dafseq =
        dafny.DafnySequence
        .<Character>empty(dafny.TypeDescriptor.CHAR);
    //Assign value of line
    dafseq = dafny.DafnySequence.asString(line);
    //Parse string and obtain a NNF_Formula
    NNF__Formula f = Parsing__CTL__to__NNF__Sequents_Compile
        .__default.parseNNF(dafseq);
    //Insert the formula into the sequence of formulae,
    //and "save" it assigning it
    fseq = Parsing__CTL__to__NNF__Sequents_Compile.__default
        .insert__in__place(f, fseq);
}
//Close the input stream
```

```
fstream.close();
//Return set of formulae
return fseq;
```

As additional functionality, the application gives the user the option of passing a file or a directory (of files) as a parameter. In the case of passing a file, the method `is_refutable?` is executed and returns to the user a model in the case of SAT (counterexample for the CMC) or a proof in the case of UNSAT (proof of the property being checked). On the other hand, in the case of passing a folder, the application browses through the folder and file by file executes the method `is_unsat?`. This method only returns SAT/UNSAT as a response, without proofs or models.

Finally, the application also offers the user the option to export a table of results in csv format and to obtain a big-step or small-step proof. The Dafny files as well as the executable Java file `momo_console.jar` can be obtained from `https://github.com/alexlesaka/MomoCTL`.

5.3 *Experimental Results*

In order to assess the feasibility of our context-based tableau, we have tested the prototype MomoCTL on the collection of benchmarks, namely GBext, borrowed from `http://users.cecs.anu.edu.au/~rpg/CTLComparisonBenchmarks/` that was created for the comparison of various CTL-provers made in [48]. In this section we report on our performance results and compare them with the other one-pass tableau for CTL ([1]) that is called TreeTab in [48]. It is worth noting that our current Java code have been automatically generated from Dafny, therefore it is not an optimised Java code. GBext is very well elaborated and gives a comprehensive, fair, and objective collection of CTL theorem-proving problems. We expect that an in-depth analysis of our results will allow us to identify what type of heuristics, strategies, etc. are convenient to improve MomoCTL.

In GBext there are three logically equivalent versions of each formula, which indicates that in [48], the comparison performed takes into account the syntactic form of the input. As expected, when translated to our input format –a set of formulae in NNF– the three versions give similar performance result. Therefore, we really use one version of each benchmark in GBext and this reduced set is called GB. We have automatically translated each unique formula in each file in GB to a file with the logically equivalent set of formulae in NNF. The obtained collection of benchmarks is denoted as MB. Both GB and MB collections are divided into different classes of benchmarks, some of which are further divided into subclasses for satisfiable and unsatisfiable instances or for different types of formula patterns. MB and MomoCTL are available at `https://github.com/alexlesaka/MomoCTL`. The executable (.JAR) runs full MomoCTL when it is called with a single file .ctl (hence, it returns certificates). However, when that .JAR runs over a folder of benchmarks files, it runs the version of MomoCTL that returns just SAT/UNSAT values.

Both TreeTab and MomoCTL have been applied, respectively, to GB and MB. To made a fair comparison, we have run a version of MomoCTL that does not return certificates. We know that the delay between this simplified version and full MomoCTL is

about 2%. These experiments were performed with an Intel Xeon Dual core 2.60GHz CPU with 64 GB RAM computer. As the authors of [48] did, we imposed a stack limit of 512MB and a 1000 seconds time limitation for each problem instance.

In the rest of this section, we compare MomoCTL and TreeTab performance results on the most significant classes and subclasses of MB and GB, which are: *Alternating Bit Protocol (abp), Business Processes (busproc), Exponential Formulae (exp_sat and exp_unsat), Montali's Formulae (montali_sat and montali_usat), Pattern_AE, and Reskill.*

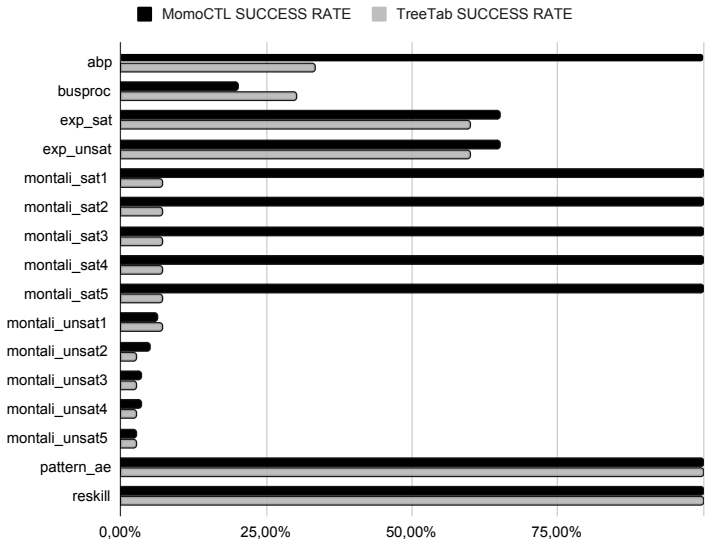


Figure 5.3: Percentage of solved instances (by class) within time limit

In Figure 5.3, we show the percentage of files in each class that are solved within the time limit, i.e those that return a value –satisfiable or unsatisfiable– and do not produce timeout or some error –e.g lack of memory. We use gray color for TreeTab and black color for MomoCTL. Instances are ordered by the increasing complexity. As we will detail bellow, for MomoCTL it takes longer than for TreeTab to solve smaller instances, but the former solves (in 1000 s.) greater instances –in *classes abp, exp_sat, exp_unsat, montali_sat*– that TreeTab cannot solve (in 1000 s.). TreeTab solves more instances than MomoCTL in *busproc and montali_unsat1.* Next, for each class, we compare the running times of both CTL-provers and analyse possible causes of big differences. In the plots, the abscissa is the number of different variable symbols in the input and the ordinate is the running time in milliseconds. The plots are given using a logarithmic scale (the same scale used in [48]).

1. Alternating Bit Protocol (abp). This class has three instances that encode whether three different properties are valid for a protocol specification (see [48]). All instances are unsatisfiable and the problem is quite difficult for CTL-provers based on tableaux. Tableaux methods necessarily should produce a huge number of closed branches. In the simplest problem (*abp5*), each branch contains four eventualities and a releases-formula that cannot be fulfilled in any order, therefore 120 combinations should

be tested. The other two instances (*abp8* and *abp9*) add to *abp5* one extra eventuality with the releases-formula included in it, which increases the combinations up to 720. As shown in Figure 5.4, only the simplest instance (*abp5*) is solved by TreeTab in 0.081 s. while MomoCTL is able to solve the three instances in 0.249 s., 11.952 s., and 13.370 s. respectively.

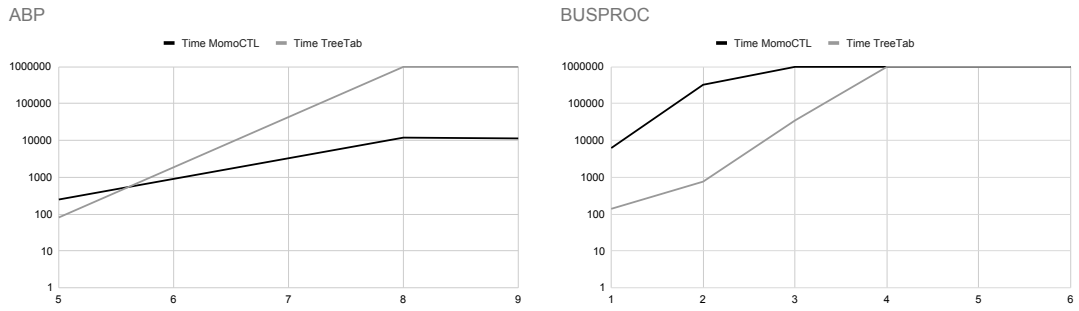


Figure 5.4: abp and busproc formulae

2. Business Processes (busproc). This class arises from a synthesis problem ([8]). All the instances have a huge amount of different models that exponentially increases with the input size. For tableau methods, to find just one model it suffices to decide the satisfiability. Consequently, (as showed in [48]) tableau methods perform better than other approaches for satisfiable inputs. However, CTL models of *busproc* problems are non-linear Kripke-structures. Figure 5.4 compares TreeTab and MomoCTL. TreeTab solves the three first instances –*busproc1* in 0.139 s., *busproc2* in 0.762 ms. and *busproc3* in 34.991 s. MomoCTL only solves *busproc1* and *busproc2* in 6,226 s. and 326,316 s. respectively. In the case of *busproc1*, (the full version) returns a model of depth five with four branches. For this construction, MomoCTL collected about 400 refuted sequents, hence in the tableau there are many closed branches that are constructed before the first model is found.

Analysing these refuted sequents (for *busproc1* and other benchmarks with many models), we saw that most of them are attempts to fulfill an eventuality at some state where it can not be fulfilled, although there are many possibilities to explore to get to decide it. We conclude that MomoCTL strategy of forcing eventualities to be satisfied "as soon as possible" does not work very well with satisfiable problems with a big amount of models. Indeed, contexts could produce a big explosion of tableau OR-branches when the selected eventuality is delayed to the next step. We think that MomoCTL should be equipped with some heuristics for eventuality selection. The class *busproc* is a useful collection of problems for our further work.

3. Exponential Formulae. Both satisfiable and unsatisfiable exponential formulae have a similar difficulty for TreeTab and MomoCTL. The subclass of satisfiable instances has models with exponentially larger paths. The tableau for unsatisfiable instances builds an exponentially increasing number of non-repeated tableau branches that are closed.

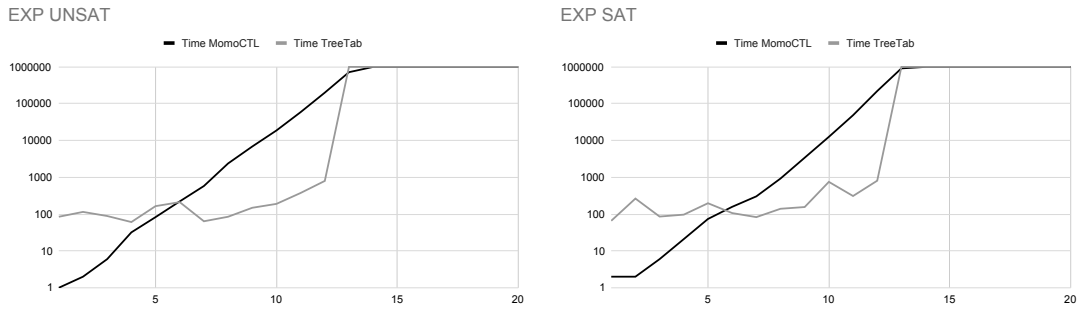


Figure 5.5: Exponential satisfiable and unsatisfiable formulae

In Figure 5.5 we can see that MomoCTL has an exponential growth of the running time from small instances in both subclasses, while TreeTab does not. Nevertheless TreeTab runs out of memory from size 12, whereas MomoCTL is able to solve the instance of size 13 inside the time limit. Allowing more time, MomoCTL solves the unsatisfiable instance of size 14 in 1,551 s. This is thanks to the strategy of not repeating refutations that have been made previously. To solve the satisfiable instance of size 14, MomoCTL needs 6,000 s. because the model of this formula is extraordinarily large. Therefore, MomoCTL solves more difficult instances as the time increases. This is one of the main features of our prototype which makes it different from the TreeTab.

4. Montali's Formulae. These formulae are CTL-reformulations of LTL-specifications of business processes ([73]) that has been used for comparing LTL-provers ([43]). Montali's formulae are parameterised by n and m.

$$\varphi_1^i = \mathsf{A}\Diamond p_i \qquad \varphi_m^i = \mathsf{A}\Diamond(p_i \wedge \mathsf{A}\circ\varphi_{m-1}^i) \qquad \varphi_n = \wedge_{i=0}^{n-1}\mathsf{A}\Box(\neg p_i \vee \mathsf{A}\circ\mathsf{A}\Diamond p_{i+1})$$

Satisfiable instances are of the form $\varphi_m^0 \wedge \varphi_n$, whereas unsatisfiable instances have the form $\varphi_m^0 \wedge \varphi_n \wedge \neg\varphi_m^n$.

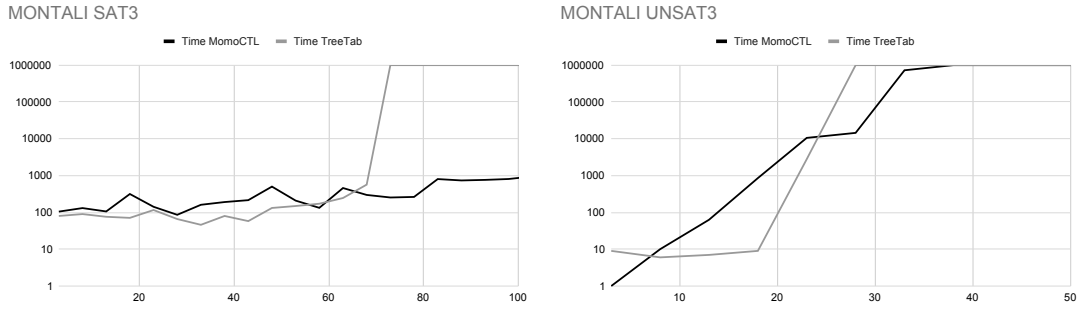


Figure 5.6: Montali's satisfiable and unsatisfiable formulae with depth 3

Figure 5.6 shows the results for satisfiable and unsatisfiable subclasses where $m = 3$ and n is in the abscissa. For $m = 1, 2, 4, 5$ the plots are very similar. In the case of satisfiable formulae, MomoCTL finds a model in the first (leftmost) branch when doing the depth-first search.

Regarding unsatisfiable formulae, MomoCTL is able to respond more (until size 33) but it requires more time than TreeTab, which runs out of memory at size 23. Note the similarity of the plots for *montali_unsat* and exponential formulae. As we mentioned above, MomoCTL can solve larger and larger instances as time increases. The main reason is that the systematic forcing of eventualities takes more advantage as the number of eventualities grows – they produce always-formulae that reduce the search space.

5. Pattern_AE and Reskill. In [48], the authors introduce these two classes and show that methods based on BDDs and resolution perform badly on inputs containing many conflicting E□ temporal formulae (in the case of *pattern_ae*) and when there are many potential resolution-steps in a satisfiable formula (in the case of *reskill*). The class *pattern_ae* contains formulae of the form

$$(\bigwedge_{i=0}^{n} \mathsf{A}\Box\mathsf{E}\Box p_i) \vee (\bigwedge_{i=0}^{n} \mathsf{A}\Box\mathsf{E}\Box\neg p_i).$$

while the class *reskill* contains these kinds of formulae

$$\neg p \wedge \ (\neg p \vee \ (\ p \ \wedge \ \mathsf{A}\Box(\bigwedge_{i=0}^{n-1} \mathsf{A}\Diamond q_i \ ) \wedge \bigwedge_{i=0}^{n-1} \mathsf{A}\Box(\ \neg q_i \ \vee \bigvee_{j=0, j\neq i}^{n-1} \mathsf{E}\bigcirc q_j \ ) \wedge \bigwedge_{i=0}^{n-1} \mathsf{A}\Box(\ \neg q_i \ \vee \bigvee_{j=0, j\neq i}^{n-1} \neg q_j) \ ) \ )$$

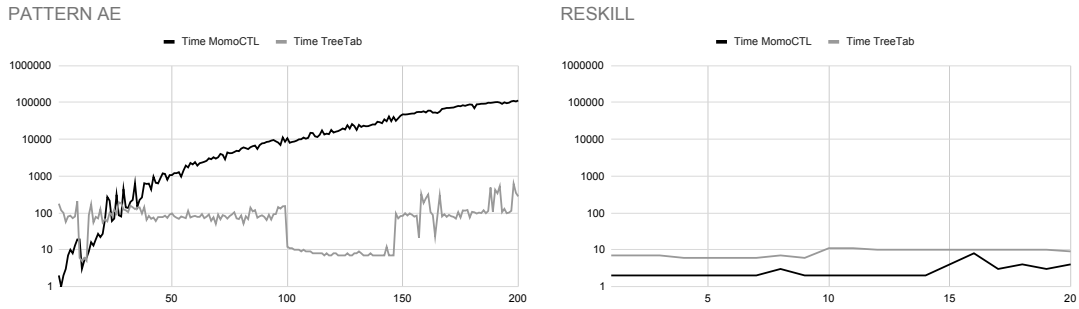


Figure 5.7: Pattern_AE and Reskill formulae

The instances of both classes are satisfiable. Figure 5.7 shows that both –TreeTab and MomoCTL– run very fast on the two classes of formulae. In *pattern_ae*, MomoCTL is faster than TreeTab in small instances until $n = 25$. MomoCTL performs with a moderate increase of time, while TreeTab is more constant with some insignificant increase or decrease of the number of steps. The model returned by MomoCTL to pattern_ae of size n is:

```
State 0: {p0,...,pn-1} --> cycle to State 0
```

Both MomoCTL and TreeTab perform extremely fast on the class *reskill* because they are able to find the simplest model, which is returned by MomoCTL as follows:

```
State 0: {}
---State 1:{} --> cycle to State 1
```

Finally, we again remark that MomoCTL does not run out of memory on any benchmark. Indeed, with unlimited execution time, MomoCTL solves some benchmarks, on which TreeTab runs out of memory. We believe that these errors of TreeTab are due to the fact that the branches it produces are too large. We know that the way how we use contexts to handle eventualities prevents the generation of such extra large branches. However, the price we pay to avoid large branches is that we have to analyse many more of them. This makes MomoCTL run slower than TreeTab for some not very complex problems. We are convinced that a more intelligent eventuality selection procedure, in addition to different heuristic strategies could solve this problem and improve notably MomoCTL's performance.

6. CONCLUSIONS AND FUTURE LINES

This chapter presents the central results and main contributions of the thesis, a list of our publications and relevant research activities related to the dissertation, as well as proposals for future lines of research. The chapter consists of three sections: Section 6.1 summarises the main contributions. Section 6.2 presents the publications and activities done during the developed of the thesis. Section 6.3 outlines possible lines of future research.

6.1 Results and Contributions

The aim of this thesis was to contribute new ideas in the field of certifying the answers provided by automated reasoners when they work with temporal logics. Specifically, we wanted to certify these two issues: why a temporal formula is satisfiable or unsatisfiable, and why a transition system satisfies a temporal property or not.

We began by adapting the dual system of tableaux and sequents for PLTL first described in [40, 41]. This adaptation consisted of improving the system by using the negation normal form of the formulae and assigning some of the work to be done to a SAT solver. From there, we have applied the method (the aforementioned adaptation) to the certification of the model-checking problem for PLTL. Our method is capable of producing automatically formal proofs, which can be independently verified by a separate tool. We have implemented the sequent calculus in the interactive theorem prover, Isabelle, which makes it easy for users to review, explain, and navigate through such formal proofs. As an extra benefit we have an efficient method to certify PLTL (un)satisfiability. This is happening because the fact that the PLTL-satisfiability problem can be reduced to the PLTL-model-checking problem.

Secondly, we have dealt with the branching logics CTL and ECTL. For these logics, we have introduced new dual context-based methods of tableaux and sequents to certify the satisfiability problem, and proved the correction (soundness, completeness and termination) of the proposed methods. Not only that, but we have created a prototype called MomoCTL, which applies to CTL. To implement it, we have used the Dafny language, which has allowed us to specify and develop the algorithm, generating the code automatically (available in `https://github.com/alexlesaka/MomoCTL`). The tool can be also used to certify CTL-model checking because this problem reduces to the CTL-satisfiability problem. We have obtained good results when the prototype has been run on a set of well-known benchmarks and has been compared with the Gore's tool.

On the other hand, we would like to emphasise that during the development of the thesis we asked ourselves the classic question. "who verifies the verifier?", due to the possibility of errors in our methods and algorithms. We have analysed the use of Isabelle

and Dafny to address this question. When we implemented the PLTL sequent calculus in Isabelle we also proved the soundness of the calculus. Our experience in doing so has been very rewarding. Although the completeness of the method remains to be proven, the verification of partial correctness increases the level of confidence in our calculus. Dafny has been used to implement MomoCTL. We have found that Dafny incorporates most of the good features of modern programming and specification languages. Dafny allows correctness proofs to be written as part of the program text. These proofs explain the reasons why programs are correct. While the ingredients of the proofs are provided by the user, the steps of the proofs themselves are performed automatically by the verifier. Using Dafny has been very beneficial in building a reliable tool.

To sum up, we have contributed new methods to address the problems of satisfiability and model checking for a variety of temporal logics. They are able to provide, with the same mechanism, the two types of certificates: models and proofs. We believe that automated certification in temporal logic can benefit from the methods presented in this dissertation.

6.2 Related Publications, Presentations and Research Activity

This section lists journal publications, contributions to conferences and workshops and research visits that have been made during the development of the thesis.

Journal Publications

- **Verified Model Checking for Conjunctive Positive Logic**
  A. Abuin, U. Diaz de Cerio, M. Hermo, P. Lucio
  SN Computer Science 2 (5), 1-24
  DOI: `https://doi.org/10.1007/s42979-020-00417-3`
  Published: 19 June 2021
- **Optimization Techniques and Formal Verification for the Software Design of Boolean Algebra Based Safety-Critical Systems**
  J. Perez, J.L. Flores, C. Blum, J. Cerquides, A. Abuin
  IEEE Transactions on Industrial Informatics, vol. 18, no. 1, pp. 620-630
  DOI: `https://doi.org/10.1109/TII.2021.3074394`
  Published: January 2022
- **Tableaux and Sequent Calculi for CTL and ECTL: Satisfiability Test with Certifying Proofs and Models**
  A. Abuin, A. Bolotov, M. Hermo, P. Lucio
  Journal of Logical and Algebraic Methods in Programming, vol. 130, 100828
  DOI: `https://doi.org/10.1016/j.jlamp.2022.100828`
  Published: January 2023

Conference Proceedings

- **Context-based Model Checking using SMT-solvers (Work in Progress)**
  A. Abuin, U. Diaz de Cerio, M. Hermo, P. Lucio
  Proceedings XVIII Jornadas sobre Programación y Lenguajes (PROLE 2018)
  `http://hdl.handle.net/11705/PROLE/2018/019`

- **Towards certified model checking for PLTL using one-pass tableaux**
  A. Abuin, A. Bolotov, U. Diaz de Cerio, M. Hermo, P. Lucio
  Proceedings 26th International Symposium on Temporal Representation and Reasoning (TIME 2019)
  DOI: https://doi.org/10.4230/LIPIcs.TIME.2019.12
- **One-pass Context-based Tableaux Systems for CTL and ECTL**
  A. Abuin, A. Bolotov, M. Hermo, P. Lucio
  Proceedings 27th International Symposium on Temporal Representation and Reasoning (TIME 2020)
  DOI: https://doi.org/10.4230/LIPIcs.TIME.2020.14

Workshop Proceedings

- **Using Contexts in Tableaux for PLTL: An illustrative Example**
  A. Abuin, A. Bolotov, U. Diaz de Cerio, M. Hermo, P. Lucio
  Automatic Reasoning Workshop 2019 (ARW 2019)
  University of Middlesex - London (UK), 2nd-3rd September 2019
  https://arw.csc.liv.ac.uk/2019.html
- **Complementing Tableaux-based Satisfiability Algorithm for CTL by Certifying Sequent Proofs and Models (regular talk)**
  A. Bolotov, A. Abuin, P. Lucio, M. Hermo
  XII Workshop Program Semantics, Specification and Verification: Theory and Applications (PSSV-2021)
  Innopolis (Russia) - online, 4th-5th November 2021
  https://persons.iis.nsk.su/en/pssv21

Research Visit

- **Research Area:**
  Certified Model Checking for PLTL based on Tableau and Sequents method
  Research Visitor: Alex Abuin
  Supervisor: Alexander Bolotov
  Software Systems Engineering Research Group
  School of Computer Science and Electronics, University of Westminster.
  London, United Kingdom, from 28 February 2019 to 15 June 2019
- **Research Area:**
  Application of Model Checking in Safety Critical Systems
  Research Visitor: Alex Abuin
  Supervisor: Franco Raimondi
  Computer Science Department
  School Science & Technology, University of Middlesex.
  London, United Kingdom, from 28 February 2019 to 15 June 2019

6.3 Future Work

Many ideas have emerged during the development of this thesis. In this section, we outline several promising lines of future work.

- **An integrated algorithm for PLTL.** Tableaux development and proof generation have each been studied separately in the case of PLTL. It is straightforward to design an algorithm for PLTL that integrates the generation of both models and formal proofs after the work done for CTL.
- **Implementation and automated correctness proof of the algorithm for PLTL.** Dafny has shown to be very helpful for implementing the CTL algorithm, as well as for enabling its verification. Similar to how it was done for CTL, an integrated algorithm for PLTL can be implemented in Dafny. As a result, in addition to generate code automatically, we could verify some properties of the algorithm, such as its correctness.
- **Comparison with other (Certified) Model Checking for PLTL.** This thesis proposes utilising a SAT solver to optimise the tableau and sequent system for CMC in PLTL. Partial prototypes and proofs have been developed, as well as illustrative examples to explain this optimisation. To measure the advantages of our proposal, a comparison with other model checkers for PLTL would be necessary.
- **Use of a SAT solver in branching logics.** Since most of the formulae used in CMC are very simple temporal formulae (especially those describing transition systems), we have proposed (for PLTL) to delegate the analysis of these formulae to a SAT solver. Clearly, incorporating the use of the SAT solver, as has been done for PLTL, would significantly improve MomoCTL. In fact, we can do the same in other more expressive branching logics.
- MomoCTL **in industrial use cases** The examples with which we have illustrated the main ideas and concepts of this thesis have been created ad hoc. For instance, the satisfiable or unsatisfiable sets of formulae, the examples representing transition systems and properties, etc. Although the collection of benchmarks we have used to test MomoCTL contains more real-world examples, we would still like to evaluate MomoCTL in a fully industrial environment. The use of MomoCTL implementation in real use cases could give evidence of its actual potential value.
- **The dual method as a verification tool for safety-critical systems.** Safety-critical systems are subject to very strict development and regulations. Therefore, each of the tools and techniques used in this scenario must be accompanied by evidence of their correct functioning. We believe that the dual context-based tableaux and sequent calculi method is suitable for safety-critical systems: if a tableau receives as input a given set of formulae and returns a proof guaranteeing that the input is unsatisfiable, an independent tool like Isabelle could easily corroborate the same result. On the other hand, the Dafny tool, which has been used to implement MomoCTL, allows us to verify the crucial properties that MomoCTL must fulfill, such as termination, soundness, etc. This gives confidence when using MomoCTL.

In addition, we would like to explore how to extend the dual method to address problems that are defined in terms of uncontrolled external variables and evolve over time. The realizability and synthesis of PLTL specifications are two issues in this area.

BIBLIOGRAPHY

[1] P. Abate, R. Goré, and F. Widmann. One-pass tableaux for computation tree logic. In N. Dershowitz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, volume 4790 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2007.

[2] A. Abuin, A. Bolotov, U. D. de Cerio, M. Hermo, and P. Lucio. Using contexts in tableaux for pltl: An illustrative example. In *Proceedings of the 26th Automated Reasoning Workshop 2019 Bridging the Gap between Theory and Practice*, page 23, 2019.

[3] A. Abuin, A. Bolotov, U. Díaz-de-Cerio, M. Hermo, and P. Lucio. Towards certified model checking for PLTL using one-pass tableaux. In J. Gamper, S. Pinchinat, and G. Sciavicco, editors, *26th International Symposium on Temporal Representation and Reasoning, TIME 2019, October 16-19, 2019, Málaga, Spain*, volume 147 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[4] A. Abuin, A. Bolotov, M. Hermo, and P. Lucio. One-Pass Context-Based Tableaux Systems for CTL and ECTL. In E. Muñoz-Velasco, A. Ozaki, and M. Theobald, editors, *27th International Symposium on Temporal Representation and Reasoning (TIME 2020)*, volume 178 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:20, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[5] A. Abuin, A. Bolotov, M. Hermo, and P. Lucio. Tableaux and sequent calculi for CTL and ECTL: satisfiability test with certifying proofs and models. *J. Log. Algebraic Methods Program.*, 130:100828, 2023.

[6] A. Abuin, U. D. de Cerio, M. Hermo, and P. Lucio. Context-based model checking using smt-solvers (work in progress). In Y. Ortega Mallén, editor, *PROLE2018*. SISTEDES, 2018.

[7] H. Amjad. Programming a symbolic model checker in a fully expansive theorem prover. In D. A. Basin and B. Wolff, editors, *Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs 2003, Rom, Italy, September 8-12, 2003, Proceedings*, volume 2758 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2003.

[8] A. Awad, R. Goré, J. Thomson, and M. Weidlich. An iterative approach for business process template synthesis from compliance rules. In *Proceedings of the 23rd In-*

ternational Conference on Advanced Information Systems Engineering, CAiSE'11, page 406–421, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] R. Backhouse, editor. *The calculational method*, volume 53 of *Information Processing Letters*. Elsevier, 1995.

[10] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.

[11] A. Bauer, M. Pister, and M. Tautschnig. Tool-support for the analysis of hybrid systems and models. In R. Lauwereins and J. Madsen, editors, *2007 Design, Automation and Test in Europe Conference and Exposition, DATE 2007, Nice, France, April 16-20, 2007*, pages 924–929. EDA Consortium, San Jose, CA, USA, 2007.

[12] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[13] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of bdds. In M. J. Irwin, editor, *Proceedings of the 36th Conference on Design Automation, New Orleans, LA, USA, June 21-25, 1999*, pages 317–320. ACM Press, 1999.

[14] A. Biere and D. Kröning. Sat-based model checking. In E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors, *Handbook of Model Checking*, pages 277–303. Springer, 2018.

[15] A. Bolotov, M. Hermo, and P. Lucio. Extending fairness expressibility of ECTL+: A tree-style one-pass tableau approach. In N. Alechina, K. Nørvåg, and W. Penczek, editors, *25th International Symposium on Temporal Representation and Reasoning, TIME 2018, Warsaw, Poland, October 15-17, 2018*, volume 120 of *LIPIcs*, pages 5:1–5:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[16] A. Bolotov, M. Hermo, and P. Lucio. Branching-time logic ECTL# and its tree-style one-pass tableau: Extending fairness expressibility of ECTL+. *Theor. Comput. Sci.*, 813:428–451, 2020.

[17] R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich. The opensmt solver. In J. Esparza and R. Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153. Springer, 2010.

[18] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[19] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10ˆ20 states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[20] CENELEC and EN50128. 50128. *Railway applications-Communication, Signaling and Processing Systems-Software for Railway Control and Protection Systems*, 2011.

[21] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.

[22] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new symbolic model checker. *Int. J. Softw. Tools Technol. Transf.*, 2(4):410–425, 2000.

[23] K. Claessen and N. Sörensson. A liveness checking algorithm that counts. In G. Cabodi and S. Singh, editors, *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*, pages 52–59. IEEE, 2012.

[24] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[25] E. M. Clarke and E. A. Emerson. Using Branching Time Temporal Logic to Synthesise Synchronisation Skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[26] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[27] E. M. Clarke and D. A. Grumberg, O.and Peled. *Model checking*. MIT Press, London, Cambridge, 1999.

[28] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2001.

[29] M. Comini, L. Titolo, and A. Villanueva. Abstract diagnosis for tccp using a linear temporal logic. *Theory Pract. Log. Program.*, 14(4-5):787–801, 2014.

[30] L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[31] B. Dutertre. Yices 2.2. In A. Biere and R. Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, 2014.

[32] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 995–1072. Elsevier and MIT Press, 1990.

[33] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.

[34] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.

[35] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[36] J. Esparza, P. Lammich, R. Neumann, T. Nipkow, A. Schimpf, and J. Smaus. A fully verified executable LTL model checker. In N. Sharygina and H. Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 463–478. Springer, 2013.

[37] M. Fitting. Tableau methods of proof for modal logics. *Notre Dame J. Formal Log.*, 13(2):237–247, 1972.

[38] D. M. Gabbay. *Expressive Functional Completeness in Tense Logic (Preliminary report)*, pages 91–117. Springer Netherlands, Dordrecht, 1981.

[39] D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer, 1987.

[40] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. A cut-free and invariant-free sequent calculus for PLTL. In J. Duparc and T. A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2007.

[41] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. Dual systems of tableaux and sequents for PLTL. *J. Log. Algebraic Methods Program.*, 78(8):701–722, 2009.

[42] G. Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935.

[43] V. Goranko, A. Kyrilov, and D. Shkatov. Tableau tool for testing satisfiability in LTL: implementation and experimental analysis. *Electron. Notes Theor. Comput. Sci.*, 262:113–125, 2010.

[44] V. Goranko and A. Zanardo. From linear to branching-time temporal logics: Transfer of semantics and definability. *Log. J. IGPL*, 15(1):53–76, 2007.

[45] R. Goré. *Tableau Methods for Modal and Temporal Logics*, pages 297–396. Springer Netherlands, Dordrecht, 1999.

[46] R. Goré. And-or tableaux for fixpoint logics with converse: Ltl, ctl, PDL and CPDL. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, volume 8562 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 2014.

[47] R. Goré. And-or tableaux for fixpoint logics with converse: LTL, CTL, PDL and CPDL. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, volume 8562 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 2014.

[48] R. Goré, J. Thomson, and F. Widmann. An experimental comparison of theorem provers for CTL. In C. Combi, M. Leucker, and F. Wolter, editors, *Eighteenth International Symposium on Temporal Representation and Reasoning, TIME 2011, Lübeck , Germany, September 12-14, 2011*, pages 49–56. IEEE, 2011.

[49] A. Griggio, M. Roveri, and S. Tonetta. Certifying proofs for LTL model checking. In N. Bjørner and A. Gurfinkel, editors, *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*, pages 1–9. IEEE, 2018.

[50] G. J. Holzmann. *The SPIN Model Checker - primer and reference manual*. Addison-Wesley, 2004.

[51] IEC. Iec 61508 functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

[52] ISO. Road vehicles – Functional safety, 2011.

[53] R. Kashima. An axiomatization of ECTL. *J. Log. Comput.*, 24(1):117–133, 2014.

[54] H. A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In W. J. Clancey and D. S. Weld, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, pages 1194–1201. AAAI Press / The MIT Press, 1996.

[55] R. Kowalski. AND/OR graphs, theorem-proving graphs and bi-directional search. In *Machine Intelligence 7*, pages 167–194, 1972.

[56] J. Kreiker, A. Tarlecki, M. Y. Vardi, and R. Wilhelm. Modeling, analysis, and verification - the formal methods manifesto 2010 (dagstuhl perspectives workshop 10482). *Dagstuhl Manifestos*, 1(1):21–40, 2011.

[57] D. Kroening and O. Strichman. *Decision Procedures - An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2008.

[58] T. Kuismin and K. Heljanko. Increasing confidence in liveness model checking results with proofs. In V. Bertacco and A. Legay, editors, *Hardware and Software: Verification and Testing - 9th International Haifa Verification Conference, HVC 2013, Haifa, Israel, November 5-7, 2013, Proceedings*, volume 8244 of *Lecture Notes in Computer Science*, pages 32–43. Springer, 2013.

[59] O. Kupferman and M. Y. Vardi. From complementation to certification. *Theor. Comput. Sci.*, 345(1):83–100, 2005.

[60] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In E. M. Clarke and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer, 2010.

[61] K. R. M. Leino. Compiling Hilbert's epsilon operator. In A. Fehnker, A. McIver, G. Sutcliffe, and A. Voronkov, editors, *LPAR-20. 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations*, volume 35 of *EPiC Series in Computing*, pages 106–118. EasyChair, 2015.

[62] K. R. M. Leino. Well-founded functions and extreme predicates in Dafny: A tutorial. In B. Konev, S. Schulz, and L. Simon, editors, *IWIL-2015. 11th International Workshop on the Implementation of Logics*, volume 40 of *EPiC Series in Computing*, pages 52–66. EasyChair, 2016.

[63] K. R. M. Leino. Dafny power user: Iterating over a collection. manuscript krml 275, 2020.

[64] K. R. M. Leino and N. Polikarpova. Verified calculations. In E. Cohen and A. Rybalchenko, editors, *Verified Software: Theories, Tools, Experiments — 5th International Conference, VSTTE 2013, Revised Selected Papers*, volume 8164 of *Lecture Notes in Computer Science*, pages 170–190. Springer, 2014.

[65] K. R. M. Leino and V. Wüstholz. The Dafny integrated development environment. In C. Dubois, D. Giannakopoulou, and D. Méry, editors, *Proceedings 1st Workshop on Formal Integrated Development Environment, F-IDE 2014*, volume 149 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–15. Open Publishing Association, 2014.

[66] J. Li, G. Pu, L. Zhang, M. Y. Vardi, and J. He. Accelerating LTL satisfiability checking by SAT solvers. *J. Log. Comput.*, 28(6):1011–1030, 2018.

[67] J. Li, S. Zhu, G. Pu, L. Zhang, and M. Y. Vardi. Sat-based explicit LTL reasoning and its application to satisfiability checking. *Formal Methods Syst. Des.*, 54(2):164–190, 2019.

[68] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.

[69] N. Markey. Temporal logics. Course notes, Master Parisien de Recherche en Informatique, Paris, France, 2013.

[70] D. Matichuk, T. C. Murray, and M. Wenzel. Eisbach: A proof method language for isabelle. *J. Autom. Reason.*, 56(3):261–282, 2016.

[71] K. L. McMillan. *Symbolic model checking*. Kluwer, 1993.

[72] A. Mebsout and C. Tinelli. Proof certificates for smt-based model checkers for infinite-state systems. In R. Piskac and M. Talupur, editors, *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, pages 117–124. IEEE, 2016.

[73] M. Montali, P. Torroni, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, and P. Mello. Verification from declarative specifications using logic programming. In M. G. de la Banda and E. Pontelli, editors, *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 of *Lecture Notes in Computer Science*, pages 440–454. Springer, 2008.

[74] K. S. Namjoshi. Certifying model checkers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 2–13. Springer, 2001.

[75] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[76] M. R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in sat-based formal verification. *Int. J. Softw. Tools Technol. Transf.*, 7(2):156–173, 2005.

[77] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[78] K. Y. Rozier and M. Y. Vardi. LTL satisfiability checking. In D. Bosnacki and S. Edelkamp, editors, *Model Checking Software, 14th International SPIN Workshop, Berlin, Germany, July 1-3, 2007, Proceedings*, volume 4595 of *Lecture Notes in Computer Science*, pages 149–167. Springer, 2007.

[79] S. Schwendimann. A new one-pass tableau calculus for PLTL. In H. C. M. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '98, Oisterwijk, The Netherlands, May 5-8, 1998, Proceedings*, volume 1397 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 1998.

[80] R. Sebastiani. Lazy satisability modulo theories. *J. Satisf. Boolean Model. Comput.*, 3(3-4):141–224, 2007.

[81] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[82] R. M. Smullyan. *First-Order Logic*. Springer-Verlag Berlin, New York [etc.], 1968.

[83] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, pages 119–136, 1985.