eman ta zabal zazu

Universidad del País Vasco | Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

MASTER THESIS

MASTER DEGREE OF COMPUTER ENGINEERING AND INTELLIGENT SYSTEMS

A Tableau Method for the Realizability and Synthesis of Reactive Safety Specifications

Ander Alonso García

Advisors
Montserrat Hermo

September 15, 2022

Acknowledgments

I would like to express my sincere gratitude to my tutor Montserrat Hermo and Paqui Lucio for giving me the golden opportunity to develop this wonderful project, the entire project has been successful because of the relentless efforts of them.

I would also like to express thanks to all my friends who have been supporting me this year and to all those who have been part of my life the last 5 years for making me grow as a person; thank you very much Javi, María, Julia, Sanse, Josu, Iñaki, Aimar, Sara, Gil, Gonza, Alaitz, Marina, Gaizka, Lidia, Dani, Maider, Irene, Inma, Rubén, Javi, Aitor, and especially thank you Nerea for your unconditional support.

Last but not least, special thanks to my parents for giving me the opportunity to study in Donosti, without them it would not have been possible.

Abstract

Reactive systems are systems that continuously interact with the environment. In general, as they are critical systems, a failure or malfunction can result in serious consequences, such as loss of human lives or large economic investments. Therefore, correctly modeling the behavior and verification of the system is crucial and, for this, Linear-time Temporal Logic (LTL) and Realizabilty and Synthesis problem represent a promising approach for obtaining confidence in the correctness of a reactive system. The Realizability and Synthesis problem decides if there is a model that satisfies the given specification under all possible environmental behaviours. Moreover, it can be seen as a game between two players; the player who controls the inputs of the system to be synthesized (environment player) and the player who controls the outputs and tries to satisfy the specification for each environmental behaviour (system player).

In this Master thesis, we present both a tableau decision method for deciding the realizability of specifications expressed in a safety fragment of LTL and a prototype that builds a Realizability Tableau from a safety specification input. The prototype returns an open tableau (meaning the specification is realizable) or a closed tableau (when the specification is unrealizable). Finally, we present the future of the work and some of the improvements that will be implemented.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Traditionally, transformational systems were those that took a set of inputs, manipulated them and provided a given set of results. In transformational systems the relation of the input with the output was sufficient to specify the behaviour of the program. However, in the mid-1980s, the concept of reactive systems emerged after the widespread use of systems that continuously reacted to events generated by the environment. Consequently, traditional development techniques and tools used for transformational systems became deprecated for reactive systems due to fact that they can not be completely characterised in terms of the relation between input and output. Moreover, as continuously interacts with the environment they are more prone to errors.

Reactive systems are everywhere, for instance in industrial control systems, in interactive software systems, in avionic systems, in robot controllers, in electronic devices, and so on. Usually, critical systems are reactive systems and a failure or malfunction can have serious consequences, such as loss of human lives or large economic investments. Therefore, correctly modeling the behavior of reactive systems is crucial and, for this, formal methods such as Linear-time Temporal Logic (LTL) represent a promising approach for obtaining confidence in the correctness of a reactive system.

Currently, there are different automatic methods for verifying a formal system in the state of the art, being model-checking, realizability and synthesis the most important. Model checking tools takes a system model and a formal property as input and decides if the model satisfies the given property, whereas realizability tools takes only a formal specification and decides if exists a model that satisfies the given specification under all possible environmental inputs. Furthermore, when the specification is realizable, if the tool is able to return an implementation, it is said to solve the synthesis problem.

The problem addressed by this project is deciding whether a formal specification (written in temporal logic) is realizable or not by the construction of “Realizabilty Tableaux". Traditional tableau techniques for testing satisfiability does not directly work for realizability. As far as we know, tableau techniques has not been yet applied for solving the realizability problem of temporal formulas, beyond its auxiliary use in automata-based methods.

To illustrate why traditional tableaux do not work to decide realizability, consider the following three temporal formulas where e is an environment variable and s is a system variable: $\psi_1 = s \rightarrow \bigcirc e$, $\psi_2 = s \leftrightarrow \bigcirc e$, $\psi_3 = \bigcirc s \leftrightarrow \bigcirc e$. Note that $\bigcirc$ symbol is a temporal operator and it refers to the future. For instance, in ψ_2, whether s valuates to True, e must be True in the next instant of time for making the formula satisfiable. Below will appear another temporal operator, $\square$, which indicates that a formula must be satisfied both in the present and in all the following instants of time.

$\square\psi_1$ specification is realizable due to the fact that only depends on the truth value of the system. Actually, $\square(s \rightarrow \bigcirc e) = \square(\neg s \vee \bigcirc e)$ and by the semantics of the temporal formula (we will introduce it in Subsection 2.2.2), $\square(\neg s \vee \bigcirc e) = (\neg s \vee \bigcirc e) \wedge \bigcirc\square(\neg s \vee \bigcirc e)$. The tableau on the right applies these equivalences and jumps to the next instant of time getting a loop with the initial formula, $\square(s \rightarrow \bigcirc e)$. Therefore, a winning strategy for the system certifying the realizability of the specification. Such strategy consists of assigning *False* to the system variable s all the time.

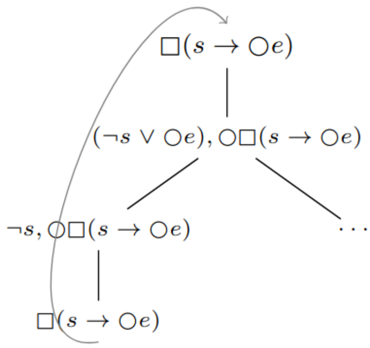


Figure 1.1: Tableau for $\square(s \rightarrow \bigcirc e)$

$\square\psi_2$ specification is not realizable, no matter what the environment does at the start, any choice of the system variable s forces the environment value in the next instant of time. Obviously, as the behaviour of the environment cannot be controlled, all branches in the tableau end up with inconsistencies (see the tableau in Figure 1.2).

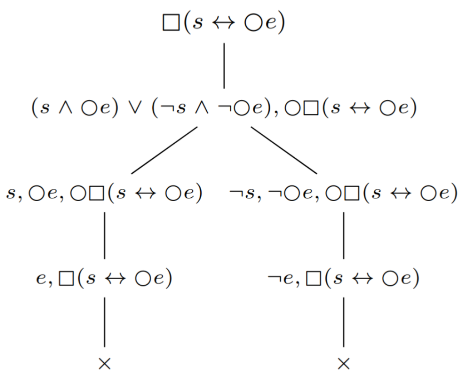


Figure 1.2: Tableau for $\square(s \leftrightarrow \bigcirc e)$

The two previous tableaux are correct for deciding realizability and they are classical tableaux, but the case of $\square\psi_3$ is different. Referring to the specification $\square\psi_3$, every time the environment establishes a value, the system only has to mimic the same value, being a winning strategy for the system. Hence, $\square\psi_3$ is a realizable specification. However, according to the rules of traditional tableaux, the first and second branch will be closed by e and $\neg e$, respectively. The tableau of Figure 1.3 shows this problem. As a consequence, the classical tableau rules do not provide a correct decision procedure for realizability.

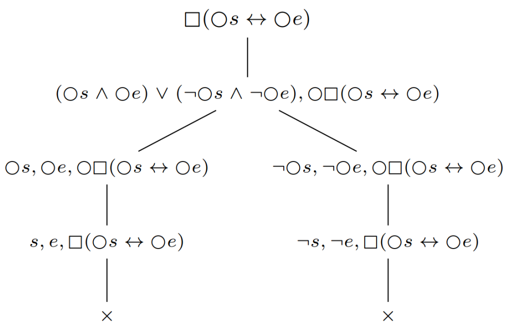


Figure 1.3: Tableau for $\Box(\bigcirc s \leftrightarrow \bigcirc e)$

To overcome this problem, we define new tableau rules and introduce the "Terse Normal Form", which prevents these incorrect splittings on formulas that reveal future choices too early. The following tableau is a correct one for $\Box\psi_3$ and is the result of the method developed in this project.

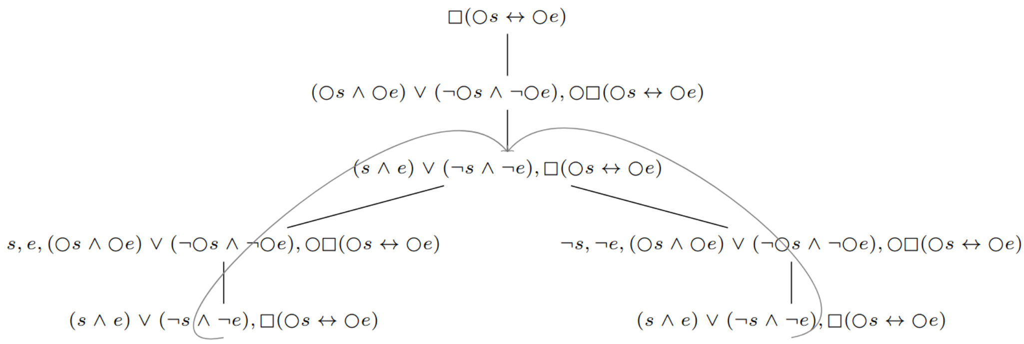


Figure 1.4: Realizability tableau for $\Box(\bigcirc s \leftrightarrow \bigcirc e)$

This master thesis is the continuation of the Computer Science End of Degree Project. However, it has changed substantially due to the fact that we detect that our tableaux had a very strong precondition and, therefore, could return that a specification was not realizable when it was. Consequently, both the rules and the construction of the tableaux have changed and new definitions have been included, such as the Terse Normal Form (TNF) or the concept of minimal covering, among others. The objectives of this project are divided into two groups: firstly, the explanation of all the theory necessary for the understanding and construction of Realizability Tableaux, and secondly, the briefly and superficially introduction to the most important aspects of the prototype implementation.

CHAPTER 2

Background

2.1 Propositional Logic

Propositional Logic is the branch of logic that studies the truth or falsehood of a propositional formula. The origins go back to antiquity and are due to Stoic school of philosophy (3rd century B.C.). However, the real development began in the mid-19th century and was initiated by mathematician G. Boole and first formulated as a formal axiomatic system by the logician G. Frege in 1879.

This Section will explain the syntax (SubSection 2.1.1) and semantics (SubSection 2.1.2) of propositional logic, the sematics tableaux for satisfiability (SubSection 2.1.4), SMT and SAT solvers (SubSection 2.1.5), the most common Normal Forms for representing boolean formulas (SubSection 2.1.3) and model minimization with prime implicants (SubSection 2.1.6) .

2.1.1 Syntax

A *propositional formula* is constructed combining together simple propositions and logic connectives such as Negation($\neg$), Conjunction($\wedge$), Disjunction($\vee$), Implication($\rightarrow$) and Double-Implication($\leftrightarrow$).

The simplest propositional formula, also called atomic formula, proposition or variable, is denoted as a string in lower case $\in PROP$, where $PROP$ is the set of atomic formulas. Moreover, each variable has a truth value: True (T) or False (F) and literals are either variables (positive literals) or the negation variables (negative literals).

2.1.2 Semantics

2.1 Definition (Model)**.** *Given a propositional formula φ, a model for φ, also called truth assignment or valuation is a mapping:*

$$\ell : Prop \rightarrow Bool$$

Given an atomic formula p, when $\ell(p)$ is the value True (respectively $\ell(p)$ is the value False), we write $p \mapsto T \in \ell$ (respectively $p \mapsto F \in \ell$).

The formal semantics is defined by the satisfaction relation $\models$ of a truth valuation ℓ and a formula φ, inductively defined as follows:

$\ell \models$ p iff { p $\mapsto$ T } $\in \ell$

$\ell \models \neg$p iff { p $\mapsto$ F } $\in \ell$

$\ell \models \varphi \wedge \psi$ iff $\ell \models \varphi$ and $\ell \models \psi$

$\ell \models \varphi \vee \psi$ iff $\ell \models \varphi$ or $\ell \models \psi$

$\ell \models \varphi \rightarrow \psi$ iff $\ell \not\models \varphi$ or $\ell \models \psi$

$\ell \models \varphi \leftrightarrow \psi$ iff ($\ell \models \varphi$ and $\ell \models \psi$) or ($\ell \not\models \varphi$ and $\ell \not\models \psi$)

2.2 Definition (Satisfiable, Unsatisfiable and Tautology)**.** *A formula φ is said to be:*

- *Satisfiable iff exists at least one model ℓ such that $\ell \models \varphi$.*
- *Unsatisfiable or contradiction iff no model ℓ satisfies φ. It is denoted as $\not\models \varphi$*
- *Tautology or valid iff every model ℓ satisfies φ. It is denoted as $\models \varphi$*

2.3 Definition (Logical equivalence)**.** *Two formulas φ and ψ are logically equivalent if the formula $\varphi \leftrightarrow \psi$ is a tautology (or likewise, if the formula $\neg(\varphi \leftrightarrow \psi)$ is unsatisfiable). Note that sign $\equiv$ is sometimes used instead of $\leftrightarrow$ for logical equivalence.*

2.1 Example. *Given the propositional formula, $p \vee q$, there are four possible assignments and three of them are models.*

1. $\{p \mapsto T, q \mapsto T\} \models p \vee q,$
2. $\{p \mapsto F, q \mapsto T\} \models p \vee q,$
3. $\{p \mapsto F, q \mapsto F\} \not\models p \vee q,$
4. $\{p \mapsto T, q \mapsto F\} \models p \vee q$

We also denote models of a formula as sets of literals or as conjunctions of literals.

Represented as sets of literals:

1. $\{p, q\} \models p \vee q,$
2. $\{\neg p, q\} \models p \vee q,$
3. $\{p, \neg q\} \models p \vee q$

Represented as conjunction of literals:

1. $(p \wedge q) \models p \vee q,$
2. $(\neg p \wedge q) \models p \vee q,$
3. $(p \wedge \neg q) \models p \vee q$

2.1.3 Normal Forms

A normal form of a formula is a syntactic restriction. In propositional logic, there are three important normal forms:

1. Conjunctive Normal Form (CNF), a propositional formula φ is in Conjunctive Normal Form if φ is a conjunction of disjunction of literals. The disjunction of literals is called clause.

   2.2 Example (CNF). *$\varphi \equiv (p \vee q \vee \neg r) \wedge (p \vee \neg r \vee t) \wedge (s \vee q \vee \neg n)$ is in Conjunctive Normal Form where $(p \vee q \vee \neg r)$, $(p \vee \neg r \vee t)$ and $(s \vee q \vee \neg n)$ are the clauses.*

2. Disjunctive Normal Form (DNF), a propositional formula φ is in Disjunctive Normal Form if φ is a disjunction of conjunction of literals. The conjunction of literals is called term or implicant.

   2.3 Example (DNF). *$\varphi \equiv (p \wedge q \wedge \neg r) \vee (p \wedge \neg r \wedge t) \vee (s \wedge q \wedge \neg n)$ is in Disjunctive Normal Form where $(p \wedge q \wedge \neg r)$, $(p \wedge \neg r \wedge t)$ and $(s \wedge q \wedge \neg n)$ are the terms.*

3. Negation Normal Form (NNF), a propositional formula φ is in Negation Normal Form if it does not contain implication or equivalence symbols, and every negation symbol occurs directly in front of an atom [1]. Moreover, every propositional formula has an equivalent formula in NNF, which can be obtained by applying the following rules:

   Implication Rule

   $$(\rightarrow)\ \frac{p \rightarrow q}{\neg p \vee q}$$

   Double Implication Rule

   $$(\leftrightarrow)\ \frac{p \leftrightarrow q}{(p \rightarrow q) \wedge (q \rightarrow p)}$$

   Double negation

   $$(\neg\neg)\ \frac{\neg\neg p}{p}$$

   Negation propagation rules

   $$(\neg\wedge)\ \frac{\neg(p \wedge q)}{\neg p \vee \neg q}$$

   $$(\neg\vee)\ \frac{\neg(p \vee q)}{\neg p \wedge \neg q}$$

   2.4 Example (NNF). *$(\neg p \vee q \wedge r \vee m \vee t) \wedge \neg c \vee (\neg q \vee n)$ is in Negation Normal Form.*

   Furthermore, a formula in Negation Normal Form has its equivalent formula in Conjunctive Normal Form or Disjunctive Normal Form by applying distributivity.

2.1.4 Semantic Tableaux

One of the most common method to check the satisfiability or unsatisfiability of a propositional formula are the Semantic Tableaux. They were introduced by Evert William Beth in 1955 [2] and later simplified for classical logic by Raymond Smullyan, who presented the one-side tableaux [3].

The Semantic Tableaux method is very simple when it applies to NNF-formulas. Any formula is decomposed into its sub-formulas according to following rules:

And Connectives Rule

$$(\wedge)\ \frac{\alpha \wedge \beta}{\alpha, \beta}$$

Or Connectives Rule

$$(\vee)\ \frac{\alpha \vee \beta}{\alpha \mid \beta}$$

Applying inductively those rules, it results in a tree-like tableau where $(\wedge)$ rule generates a single branch and $(\vee)$ rule generates two branches. Each branch terminates by a leaf with a complementary pair of formulas (a closed branch) or by a leaf containing a set of non-contradictory literals (an open branch).

In the following examples, we represent closed branches by $\times$ and open branches by $\odot$. What's more, when a leaf generates an open branch, that leaf is a model of the formula.

2.5 Example. *Open tableau for* $p \wedge (\neg p \vee q)$ *(i.e.* $p \wedge (\neg p \vee q)$ *is satisfiable).*

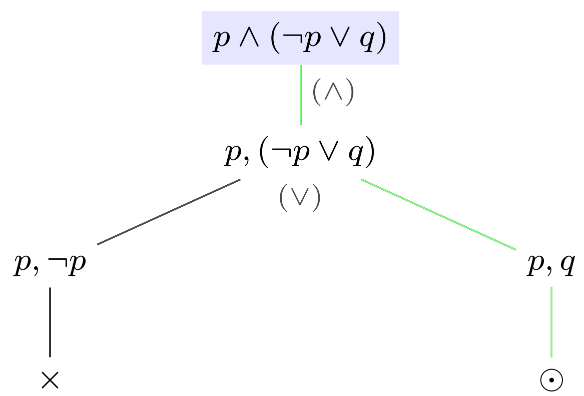


Figure 2.1: Open propositional tableau

2.6 Example. *Open tableau (with more than one open branch) for* $p \lor (\neg p \land q)$.

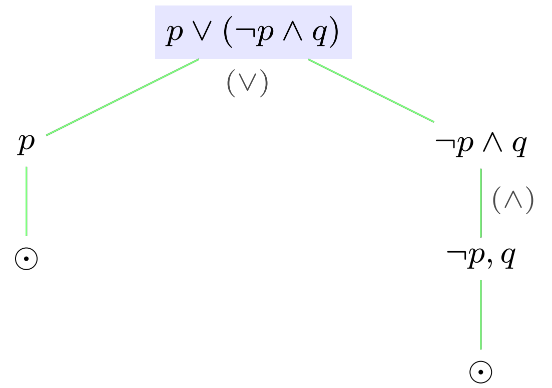


Figure 2.2: Open propositional tableau with more than one open branch

2.7 Example. *Closed tableau for* $(p \lor q) \land (\neg p \land \neg q)$ *(i.e.* $(p \lor q) \land (\neg p \land \neg q)$ *is unsatisfiable).*

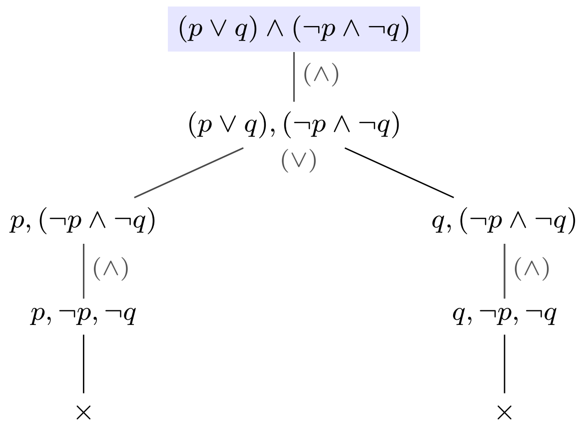


Figure 2.3: Closed propositional tableau

2.1.5 SMT/SAT Solvers

SAT Solvers are tools which aims to solve the boolean satisfiability problem. If the SAT Solver finds a model ℓ that satisfies the given formula φ, (i.e $\ell \models \varphi$) it returns "SAT" and, at the user's request, the model ℓ. Otherwise, if it proves that there is no model that satisfies the formula φ, it returns "UNSAT".

Research to improve SAT solvers is very popular, every year holds a competition to identify new challenging benchmarks and present new SAT solvers [4]. For example, one of the SAT Solvers that has obtained the best results in the last years is "Kissat SAT Solver" [5]. It is a condensed and improved reimplementation of CaDiCaL [5] in C.

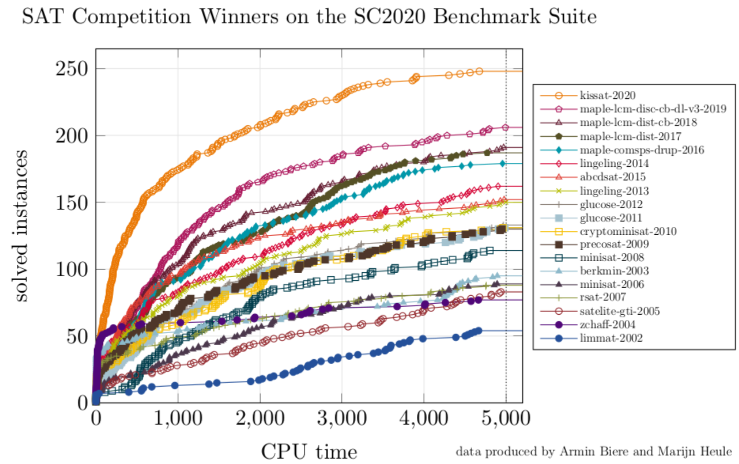


Figure 2.4: Main track SAT Competition results on 2020 instances

Another tools that allows to verify the satisfiability of a formula are SMT Solvers. Whereas SAT Solver inputs are propositional boolean formulas, SMT Solvers inputs are formulas in First-order-logic. In other words, SMT Solver extends from SAT Solvers by adding some Theory Solvers. Therefore, SMT Solver can solve a SAT problem but a SAT Solver can not solve a SMT problem.

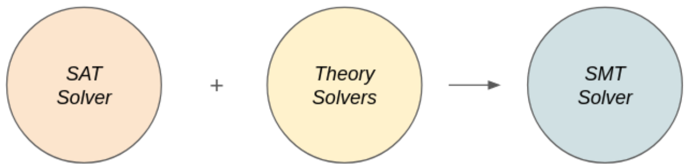


Figure 2.5: Basic SMT Solver structure

For example, one of the most popular SMT solver is Z3. It is an open source Theorem Prover and was developed by Research in Software Engineering (RiSE) group at Microsoft Research with the main target of solving problems in areas of software verification and software analysis [6].

2.1.6 Prime Implicants

Sometimes the valuation of a variable is irrelevant to the satisfaction of a propositional formula. For example, in Example 2.1, when $\{p \mapsto T\}$ the assignment of the variable q is irrelevant, that is, to satisfy the formula, it does not matter if $\{q \mapsto T\}$ or $\{q \mapsto F\}$. To some exent, this mean that the models $\{p \mapsto T, q \mapsto T\}$ and $\{p \mapsto T, q \mapsto F\}$ can be reduced to a unique model $\{p \mapsto T\}$

To reduce formula models, we use the well-known concept of prime implicants [7] and recent tool called BICA [8], which is able to compute the smallest size set of prime implicants equivalent to a given formula.

2.4 Definition (prime implicants). *A model ℓ is a prime implicant of a propositional formula φ iff seeing ℓ as a set of literal, no subset of ℓ is an implicant of φ.*

2.1 Proposition. *Let φ be a propositional boolean formula. The disjunction of all prime implicants of φ is a logically equivalent to φ.*

2.8 Example. *In reference to Example 2.1, $p \vee q$ has two prime implicants, $\{p \mapsto T\}$ and $\{q \mapsto T\}$.*

In the worst case, the number of prime implicants of a propositional formula is exponential with respect to the number of variables of the formula.

2.9 Example. *Let φ be the following propositional formula [9]:*

$$\begin{aligned}&\varphi \equiv (x_1 \wedge x_2 \wedge \cdots \wedge x_n \wedge y_0) \vee (\neg x_1 \wedge x_2 \wedge \cdots \wedge x_n \wedge y_1)\\ &\vee\ (\neg x_2 \wedge \cdots \wedge x_n \wedge y_2) \vee \cdots \vee (\neg x_n \wedge y_n)\end{aligned}$$

This formula has at least 2^n prime implicants corresponding to:

$$(b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge y_0) \text{ where } b_i \text{ can be either } x_i \text{ or } y_i.$$

In addition to the $n+1$ prime implicants:

$$\begin{aligned}&(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge y_0), (\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge y_1),\\ &(\neg x_2 \wedge x_3 \wedge x_4 \wedge y_2), (\neg x_3 \wedge x_4 \wedge y_3), \cdots, (\neg x_n \wedge y_n)\end{aligned}$$

2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) is a formal system for reasoning about time. It has found extensive application in computer science, namely to specify and verify how systems behave over time. LTL interpretations are limited to transitions which are discrete, reflexive, transitive, linear and total [10].

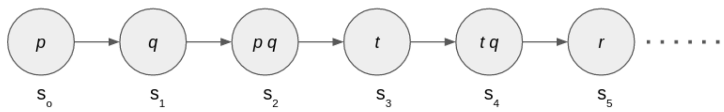


Figure 2.6: LTL example

In what follows, we will explain the LTL syntax (Subsection 2.2.1) and semantics (Subsection 2.2.2) that are similar to [11].

2.2.1 Syntax

LTL formulas are constructed using the classical operators of the propositional logic together with the temporal operators over a set of propositional formulas $PROP$.

Temporal operator are:

– Unary temporal operators: Next ($\bigcirc$ or X), Always ($\square$ or G), Eventually ($\diamondsuit$ or F)

– Binary temporal operators: Releases ($\mathcal{R}$) and Until ($\mathcal{U}$).

LTL formulas can be formally represented as a transition structure $\mathcal{M}$ = ($\mathcal{S}_{\mathcal{M}}$, $\mathcal{V}_{\mathcal{M}}$), where $\mathcal{S}_{\mathcal{M}}$ is a denumerable sequence of states s_0, s_1, s_2 ... and $\mathcal{V}_{\mathcal{M}}$ is a map $\mathcal{V}_{\mathcal{M}} : \mathcal{S}_{\mathcal{M}} \to 2^{\mathcal{E}}$. Intuitively, $\mathcal{V}_{\mathcal{M}}(s_j)$ specifies which atomic formulas are necessarily true in state s_j.

2.2.2 Semantics

The formal semantics is given by the truth of a formula φ in the state s_j of a structure $\mathcal{M}$, denoted by $\langle \mathcal{M}, s_j \rangle \models \varphi$, which is inductively defined as follows:

$\langle \mathcal{M}, s_j \rangle \models p$ iff p is a boolean variable and $p \in V_{\mathcal{M}}(s_j)$

$\langle \mathcal{M}, s_j \rangle \models \neg\varphi$ iff $\langle \mathcal{M}, s_j \rangle \not\models \varphi$

$\langle \mathcal{M}, s_j \rangle \models \varphi \wedge \psi$ iff $\langle \mathcal{M}, s_j \rangle \models \varphi$ and $\langle \mathcal{M}, s_j \rangle \models \psi$

$\langle \mathcal{M}, s_j \rangle \models \varphi \vee \psi$ iff $\langle \mathcal{M}, s_j \rangle \models \varphi$ or $\langle \mathcal{M}, s_j \rangle \models \psi$

$\langle \mathcal{M}, s_j \rangle \models \bigcirc\varphi$ iff $\langle \mathcal{M}, s_{j+1} \rangle \models \varphi$

$\langle \mathcal{M}, s_j \rangle \models \varphi \mathcal{U} \psi$ iff there exists $k \geq j$ such that $\langle \mathcal{M}, s_k \rangle \models \psi$ and for every $j \leq i < k$ it holds $\langle \mathcal{M}, s_i \rangle \models \varphi$.

$\langle \mathcal{M}, s_j \rangle \models \varphi \mathcal{R} \psi$ iff either $\langle \mathcal{M}, s_k \rangle \models \varphi$ holds for all $k \geq j$ or there exists $k \geq 0$ such that $\langle \mathcal{M}, s_k \rangle \models \varphi \wedge \psi$ and $\langle \mathcal{M}, s_i \rangle \models \varphi$ for all $j \leq i < k$.

$\langle \mathcal{M}, s_j \rangle \models \Box\varphi$ iff $\langle \mathcal{M}, s_k \rangle \models \varphi$ for all $k \geq j$

$\langle \mathcal{M}, s_j \rangle \models \Diamond\varphi$ iff $\langle \mathcal{M}, s_k \rangle \models \varphi$ for some $k \geq j$

2.10 Example. *Temporal logic operators*

1. *Next Operator ($\bigcirc$ or X): given current state S_j, $\bigcirc\ p$ will be satisfied iff in the state S_{j+1} p is satisfied.*

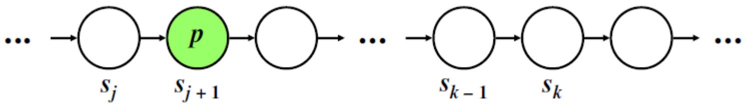


Figure 2.7: $\bigcirc\ p$

2. *Always Operator ($\Box$ or G): given current state S_j, $\Box\ p$ will be satisfied iff in the state S_j and in the following states p is satisfied.*

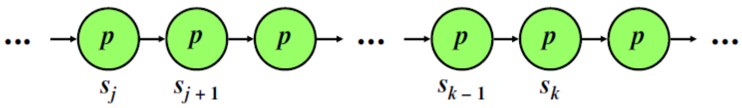


Figure 2.8: $\Box\ p$

3. *Eventually Operator ($\Diamond$ or F): given current state S_j, $\Diamond\ p$ will be satisfied iff in the state S_j and in the following states p is satisfied at least once.*

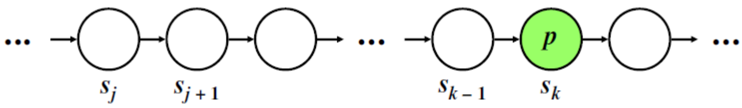


Figure 2.9: $\Diamond\ p$

4. *Releases Operator ($\mathcal{R}$ or R): given current state S_j , $p\ \mathcal{R}\ q$ will be satisfied iff p = True until and including the state where q becomes True.*

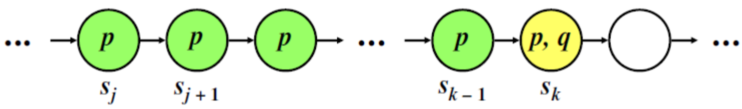


Figure 2.10: $p\ \mathcal{R}\ q$

if q never becomes true, p must be True forever.

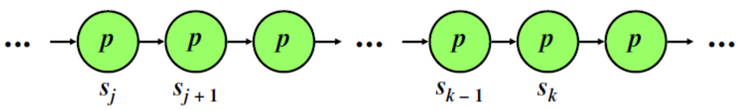


Figure 2.11: $p\ \mathcal{R}\ q$

5. *Until Operator ($\mathcal{U}$ or U): given current state S_j, $q\mathcal{U}p$ will be satisfied iff q hold $True$ at least until p becomes $True$. Moreover, p must be $True$ at least once.*

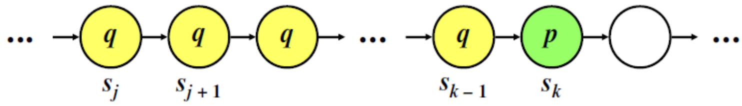


Figure 2.12: $\varphi\ \mathcal{U}\ \psi$

2.3 Reactive Systems: Definition, Specification and Verification

Reactive system concept was first introduced in 1985 by David Harel y Amir Pnueli in "On the Development of Reactive Systems" [12]. They are systems that maintains a permanent interaction with its environment and consequently are more prone to error.

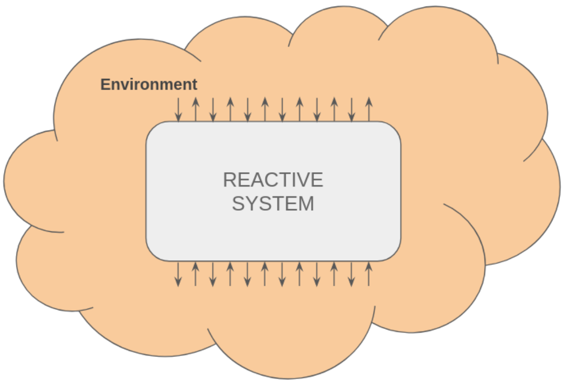


Figure 2.13: Reactive system

They can be found everywhere, for instance in industrial control systems (their principal use), in interactive software systems (such as human-machine interfaces), in avionic systems (used on airplanes, artificial satellites, and spacecraft), in robot controllers, in electronic devices (such as mobile phones), and so on.

Figure 2.14: Reactive system examples

Usually, critical systems are reactive systems and a failure or malfunction can have serious consequences, such as loss of human lives (Therac-25 radiation therapy machine kill 6 as a result of high radiation intensities exposure [13]), large economic investments (Intel's Pentium bug in floating point division unit [14] and Ariane 5 rocket explosion [15] due to a conversion of 64-bit real to 16-bit integer). Therefore, correctly modeling the behavior of reactive systems is crucial. For this, formal methods, in particular temporal logic such as Computational Tree Logic (CTL) or Linear-time Temporal Logic (LTL), represent a promising approach for obtaining confidence in the correctness of a reactive system.

Currently, there are different automatic methods for verifying a formal system in the state of the art, being model-checking and synthesis the most important. On the one hand, model checking tools takes a system model and a formal property as input and decides if the model satisfies given property. [16].

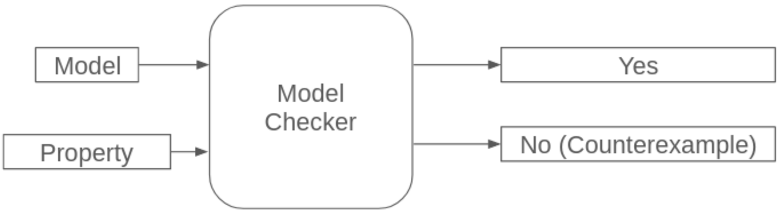


Figure 2.15: Model Checking

An example of a model checker is NuSMV [17]. It was developed by ITC-IRST and UniTN with the collaboration of CMU and UniGE and is the result of the reengineering, reimplementation, and, to a limited extent, extension of the CMU SMV model checker [18] which is based on Binary Decision Diagrams (BDDs).

On the other hand, synthesis tools takes only a formal specification and decides if exists a model that satisfies given specification under all possible environmental inputs [19]. In addition, when the answer to the synthesis problem is yes, the specification is realizable. More intuitively, synthesis can be seen as a game between two players; the player who controls the inputs of the system to be synthesized (environment player) and the player who controls the outputs and tries to satisfy the specification for each environment behaviour (system player).

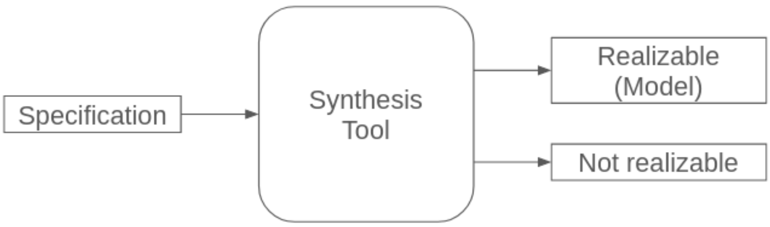


Figure 2.16: Synthesis

The synthesis of reactive systems from formal specifications, first defined by Church [20], is one of the major challenges of computer science. Every year since 2014 the SYNT-COMP [21] competition is held to compare different synthesis tools. For instance, some of those synthesis tools are:

Knorr is a synthesis tool for parity automata developed at the FMT group and it uses a effective of binary decision diagrams combined with symbolic parity game algorithms.

AbsSynthe [22] is a synthesis algorithm for safety specifications described as circuits. The algorithm is based on fixpoint computations, abstraction and refinement, it uses binary decision diagrams as symbolic data structure.

Strix [23, 24] was developed by P. J. Meyer, S. Sickert and M. Luttenberger. It combines a direct translation of temporal formulas into deterministic parity automata (DPA) with an efficient multi-threaded explicit state solver for parity games.

Ltlsynt[25] was developed by M. Colange and T. Michaud. They reduce the synthesis problem to a parity game, and solves the parity game using Zielonka's recursive algorithm.

CHAPTER 3

Safety Specifications

An important problem in reactive systems is the verification of safety properties which assert that nothing "bad" happens. They are types of linear time properties, along with liveness properties. Unlike liveness properties, if a safety property is violated there is always a finite execution that shows the contradiction. In this Chapter, we will explain the syntax and semantics for representing those safety properties, an introduction to safety games and an example of reactive system that will be used as running example to introduce new concepts along the memory.

3.1 Syntax

Our safety specifications are constructed over two different sets of variables $\mathcal{X}$ and $\mathcal{Y}$. On the one hand, the set of variables denoted as $\mathcal{X}$ and marked with e as a subscript are the variables controlled by the environment (e.g. $sensor_e$ or p_e). On the other hand, the set of variables controlled by the system is denoted as $\mathcal{Y}$ with no subscript (e.g. $controllable$ or c).

We consider a fragment of LTL specifications of the form $\alpha \wedge \Box\psi$ where α is an initial formula and ψ is a safety formula.

Initial formula α is a boolean formula that captures the initial states of the reactive system. It is constructed using variables along with the classical boolean connectives ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$). Note that we also consider boolean constants T (for truth) and F (for falsehood) as atomic formulas. More precisely, the grammar for any boolean formula β is:

$$\begin{array}{lcl} a & ::= & p \mid T \mid F \\ \beta & ::= & a \mid \neg\beta \mid \beta \wedge \beta \mid \beta \vee \beta \mid \beta \rightarrow \beta \mid \beta \leftrightarrow \beta \end{array}$$

In the always formula $\Box\psi$, the formula ψ is called the safety formula. Safety formulas are conjunctions of $n \geq 1$ temporal formulas in $\mathcal{X} \cup \mathcal{Y}$ representing the safety properties. Temporal formulas are constructed adding to the boolean formulas, the temporal operators next ($\bigcirc$), bounded eventually $\Diamond[n, m]$ and bounded always $\Box[n, m]$ for $0 \leq n \leq m$. Also, we abbreviate by $\bigcirc^i$ the sequence of i consecutive operators $\bigcirc$.

More precisely,the grammar for any temporal formula η is:

$$\eta \quad ::= \quad \beta \mid \neg\eta \mid \bigcirc\eta \mid \square_{[n,m]}\eta \mid \diamond_{[n,m]}\eta \mid \eta \vee \eta \mid \eta \wedge \eta \mid \eta \rightarrow \eta \mid \eta \leftrightarrow \eta$$

3.2 Semantics

We interpret the semantic of a safety specification in traces on their set of (occurring) variables $\mathcal{V}$. A *trace* σ is a denumerable sequence of states $\sigma_0, \sigma_1, \sigma_2, \ldots$ where each state σ_i is a valuation from $\mathcal{X} \cup \mathcal{Y}$ to $\{T, F\}$. We denote by *Val*($\mathcal{V}$) the set of all valuations on $\mathcal{V}$. For any $i \geq 0$, σ^i denotes the trace $\sigma_i, \sigma_{i+1}, \ldots$

Note that any trace $\sigma = \sigma_0, \sigma_1, \cdots$, according to the semantics defined in subsection 2.2.2, corresponds to a structure $\mathcal{M}$, being $\sigma_i = \langle \mathcal{M}, s_i \rangle$ for all $i \geq 0$.

Given a safety specification $\alpha \wedge \square\psi$, its interpretation in a trace σ is defined as follows.

$$\sigma \models \alpha \wedge \square\psi \quad \text{iff} \quad \sigma_0 \models \alpha \text{ and } \sigma^k \models \psi \text{ for all } k \geq 0$$

The meaning of $\sigma \models \psi$ for any trace σ is inductively defined as follow.

$$\begin{array}{lll}
\sigma \models p & \text{iff} & \sigma_0(p) = T \\
\sigma \models \neg\psi & \text{iff} & \sigma \not\models \psi \\
\sigma \models \varphi \wedge \psi & \text{iff} & \sigma \models \varphi \wedge \sigma \models \psi \\
\sigma \models \varphi \vee \psi & \text{iff} & \sigma \models \varphi \vee \sigma \models \psi \\
\sigma \models \varphi \rightarrow \psi & \text{iff} & \sigma \not\models \varphi \vee \sigma \models \psi \\
\sigma \models \varphi \leftrightarrow \psi & \text{iff} & (\sigma \models \varphi \wedge \sigma \models \psi) \vee (\sigma \not\models \varphi \wedge \sigma \not\models \psi) \\
\sigma \models \bigcirc^i\psi & \text{iff} & \sigma^i \models \psi \\
\sigma \models \square[n,m]\psi & \text{iff} & \sigma^j \models \psi \text{ for all } j \text{ such that } n \leq j \leq m \\
\sigma \models \diamond[n,m]\psi & \text{iff} & \text{there exists } j \text{ such that } n \leq j \leq m \text{ such that } \sigma^j \models \psi
\end{array}$$

Note that $\diamond_{[n,m]}$ can be expressed as a disjunction of formulas (e.g $\diamond_{[1,3]\varphi} \equiv \bigcirc^1\varphi \vee \bigcirc^2\varphi \vee \bigcirc^3\vee$) and $\square_{[n,m]}$ as conjunction of formulas (e.g $\square_{[0,2]\varphi} \equiv \varphi \wedge \bigcirc^1\varphi \wedge \bigcirc^2\varphi$) that only use $\bigcirc$ as temporal operator.

3.1 Example. *Bounded eventually operator ($\diamond_{[n,m]}$ or $F_{[n,m]}$), will be satisfied by the trace $\sigma = \sigma_0, \cdots, \sigma_j, \cdots \sigma_n, \cdots \sigma_m, \cdots$ iff from state σ_n to σ_m p is satisfied at least once.*

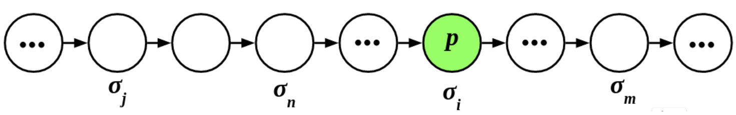


Figure 3.1: $\diamond_{[n,m]}p$

3.2 Example. *Bounded always operator ($\square_{[n,m]}$ or $G_{[n,m]}$), will be satisfied by the trace $\sigma = \sigma_0, \cdots, \sigma_j, \cdots \sigma_n, \cdots \sigma_m, \cdots$ iff every states form σ_n to σ_m p is satisfied.*

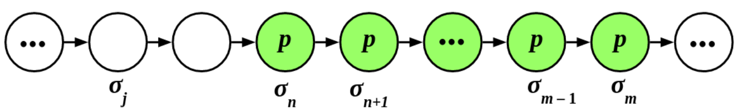


Figure 3.2: $\square_{[n,m]}p$

3.3 Safety games

LTL realizability and synthesis is usually represented by a game between two players, Eve and Sally. Eve player, denoted by E, controls the environment and the variables of the set $\mathcal{X}$ while Sally player, denoted by S, controls the system and the variables of the set $\mathcal{Y}$.

3.1 Definition (move, play)**.** *A* move *consists on the following: the player who owns the current position chooses a successor position. A* play *is an infinite sequence of moves starting from some positions within a predetermined set of initial positions.*

The outcome of a play of the game is determined as follows. Eve wins if some move during the play reaches some bad positions for a predetermined subset of bad positions. Otherwise, Sally wins. In other words, Sally wins a play if she avoids bad positions at all times during the play.

3.2 Definition. *Given a safety specification φ, φ is realizable if and only if exists a winning strategy for Sally.*

Traditional approaches based on automata games to LTL realizability and synthesis assume that Sally plays first, such as LTL synthesis tool Lily [26], whereas some successful LTL synthesis tools such as Unbeast [27] and Acacia+ [28] adopt an inverted turn game, where Eve plays first. Our tableaux for a safety specification $\varphi = \alpha \wedge \Box\psi$ analyze its realizability on the basis of a play where Eve play first choosing a move on its variables $\mathcal{X}$ and, then, the system choose its move on its variables $\mathcal{Y}$ according to the safety specification.

3.4 Running Example

We consider as running example a variant of a synthesis problem about a simple arbiter presented in [29]. The arbiter receives requests from two clients, represented by two environment variables $\mathcal{X} = \{r1_e, r2_e\}$, and responds by assigning grants, represented by two system variables $\mathcal{Y} = \{g1, g2\}$. Moreover, each request should eventually be followed by a grant in at most three second and both grants should never be assigned simultaneously.

Note that in this example there is no initial formula due to initially there are neither requests nor assigned grants, furthermore, an additional requirement is added to hinder the winning strategy. The safety specification is as follows.

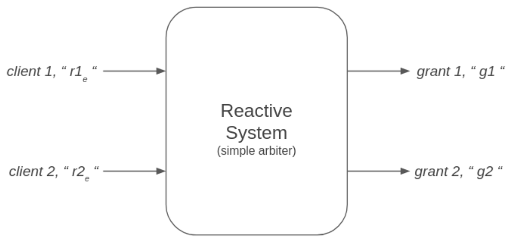


Figure 3.3: Simple arbiter running example

$$\Box\psi = \Box((r1_e \to \Diamond_{[0,3]} g_1) \wedge (r2_e \to \Diamond_{[0,3]}\, g_2) \wedge \neg(g_1 \wedge g_2) \wedge ((\neg r1_e \wedge \neg r2_e) \to \bigcirc\neg g_2))$$

CHAPTER 4

Terse Normal Form

Our tableau branches must represent a real play, so every formulas in nodes should determine the true strict-future possibilities of the game. Therefore, throughout this chapter we will introduce a new normal form for safety specifications, *Terse Normal Form*, which allows us to associate to any move the formula that any trace must satisfy in the (strict) future to be coherent with the safety specification.

4.1 Definition

First of all, given a safety specification $\alpha \wedge \Box\psi$, we consider two types of basic (sub)formulas that can be part of the safety formula ψ. Namely, all the formulas of the form ℓ, $\bigcirc^n \eta$, $\Diamond_{[n,m]}\eta$ or $\Box_{[n,m]}\eta$ are divided into two different classes of formulas:

1. *From-now* formulas refer to current state, that is, atomic formulas ℓ and bounded eventualities/always $\Diamond_{[0,m]}\eta, \Box_{[0,m]}\eta$ with a lower limit equal to 0.
2. *From-next* formulas refer to strict-future states, that is, next formulas $\bigcirc^i \eta$ and bounded eventualities/always $\Diamond_{[n,m]}\eta$ and $\Box_{[n,m]}\eta$ with a lower limit greater or equal than 1.

4.1 Definition (Strict-future and separated formulas)**.** *A safety formula is a strict-future formula if and only if it is a conjunction of from-next formulas. A safety formula is a separated formula if and only if it is the (possibly empty) conjunction of a set of Boolean literals, denoted as $\mathcal{L}(\pi)$, and (at most) a strict-future formula, denoted as $\mathcal{F}(\pi)$.*

4.2 Definition (TNF)**.** *A safety formula γ is in Terse Normal Form (*TNF*) if and only if it is a disjunction ($\bigvee_{i=1}^{n} \pi_i$) such that each π_i is a separated formula with $\mathcal{L}(\pi_i) \not\equiv F$ and for every pair of moves π_i and π_j where $1 \leq i \neq j \leq n$ there is at least one literal ℓ such that $\ell \in \mathcal{L}(\pi_i)$ and $\neg\ell \in \mathcal{L}(\pi_j)$.*

4.1 Example. *Let $\Box\varphi$ be a specification, where φ is the safety formula.*

$$\Box\varphi \equiv \Box(\underbrace{(p_e \wedge c \wedge \bigcirc c)}_{\pi_1} \vee \underbrace{(\neg p_e \wedge (\bigcirc c \vee \bigcirc^3 s))}_{\pi_2})$$

The safety formula φ is in Terse Normal Form due to is a disjuction of separated formulas, π_1 and π_2, and both separated formulas satisfy the required condition, that is, literals of both separated formulas are consistent, $\mathcal{L}(\pi_1) \equiv \{p_e, c\}$ and $\mathcal{L}(\pi_2) \equiv \{\neg p_e\}$, and the variable p_e occurs as a positive literal in π_1 and as a negative literal in π_2.

Separated formulas represent moves (see Definition 3.1) in a particular play between Eve and Sally.

4.2 Algorithm

First, any safety formula γ (for simplicity, suppose that γ is in NNF) can be converted into a disjunctive normal form-like formula, DNF(γ), applying classical logical equivalences on boolean connectives, as we have seen in Subsection 2.1.3, in addition to the following equivalences on temporal formulas:

$$\Diamond_{[n,n]}\beta \equiv \bigcirc^n\beta \qquad \Box_{[n,n]}\beta \equiv \bigcirc^n\beta$$
$$\Diamond_{[n,m]}\beta \equiv \bigcirc^n\beta \vee \bigcirc\Diamond_{[n..m-1]}\beta \qquad \Box_{[n,m]}\beta \equiv \bigcirc^n\beta \wedge \bigcirc\Box_{[n..m-1]}\beta$$

Then, we transform each pair of disjuncts in DNF(γ) with indexes $1 \leq i \neq j \leq n$ such that for all literal $\ell \in \mathcal{L}(\pi_i)$ it holds that $\neg\ell \notin \mathcal{L}(\pi_j)$ as follows. Let $\delta = \mathcal{L}(\pi_i) \cap \mathcal{L}(\pi_j)$, $\delta_1 = \mathcal{L}(\pi_i) \setminus \delta$ and $\delta_2 = \mathcal{L}(\pi_j) \setminus \delta$. Then, we apply

$$\begin{aligned}(\delta \wedge \delta_1 \wedge \eta_1) \vee (\delta \wedge \delta_2 \wedge \eta_2) \quad \equiv \quad & (\delta \wedge \delta_1 \wedge \delta_2 \wedge (\eta_1 \vee \eta_2)) \\ & \vee\ \mathrm{DNF}(\delta \wedge \delta_1 \wedge\neg\delta_2 \wedge \eta_1) \\ & \vee\ \mathrm{DNF}(\delta \wedge\neg\delta_1 \wedge \delta_2 \wedge \eta_2)\end{aligned} \tag{4.1}$$

where $\mathcal{F}(\pi_1) = \eta_1$ and $\mathcal{F}(\pi_j) = \eta_2$.

This equivalence is repeatedly applied until every pair (π_i, π_j) satisfies the required condition. Moreover, since we only apply logical equivalences to subformulas, by substitutivity, the resulting formula, denoted as TNF(γ), is logically equivalent to DNF(γ), consequently, equivalence to γ.

4.1 Proposition. *For any safety formula γ there is a logically equivalent formula, called TNF(γ), that is in* TNF.

Remark that the computed TNF from an initial DNF is not unique and the number of its moves could be exponential in the number of disjoints of the DNF, i.e. $O(2^{|\mathrm{DNF}|})$.

4.3 Examples

4.2 Example. *Let $\Box\psi$ be a safety specification, where $\mathcal{X} = \{p_e\}$, $\mathcal{Y} = \{s\}$ and* DNF*(ψ) is the following formula:*

$$\mathrm{DNF}(\psi) \equiv \underbrace{(p_e \wedge s \wedge \bigcirc s)}_{\pi_1} \vee \underbrace{(\neg s \wedge \bigcirc^2 s)}_{\pi_2} \vee \underbrace{(\neg p_e \wedge \neg s \wedge \bigcirc^3 s)}_{\pi_3}$$

The process of building a TNF*(ψ) could be as follows. First, we choose a pair of movements that do not fulfil the required conditions, π_2 and π_3.*

$$(\underbrace{\neg s}_{\delta} \wedge \underbrace{\bigcirc^2 s}_{\eta_2}) \vee (\underbrace{\neg p_e}_{\delta_3} \wedge \underbrace{\neg s}_{\delta} \wedge \underbrace{\bigcirc^3 s}_{\eta_3})$$

Then, after applying the equivalence 4.1 to π_2 and π_3

$$\begin{aligned}(\neg s \wedge \bigcirc^2 s) \vee (\neg p_e \wedge \neg s \wedge \bigcirc^3 s) \quad \equiv \quad & \neg s \wedge \neg p_e \wedge (\bigcirc^2 s \vee \bigcirc^3 s) \\ & \vee \ \mathrm{DNF}(\neg s \wedge p_e \wedge \bigcirc^2 s)\end{aligned}$$

we obtain the following formula which is already in TNF

$$\mathrm{TNF}(\psi) \equiv (p_e \wedge s \wedge \bigcirc s) \vee (\neg s \wedge \neg p_e \wedge (\bigcirc^2 s \vee \bigcirc^3 s)) \vee (\neg s \wedge p_e \wedge \bigcirc^2 s)$$

4.3 Example. *Given the safety specification $\Box\psi \equiv \Box((p_e \wedge ((\bigcirc b \wedge (a_1 \vee a_2)) \vee (\neg a_1 \wedge \bigcirc^2 c))) \vee (\neg p_e \wedge \bigcirc\neg b))$, let's see another* TNF *construction. First, we obtain the equivalent* DNF*(ψ),*

$$\mathrm{DNF}(\psi) \quad \equiv \quad (p_e \wedge a_1 \wedge \bigcirc b) \vee (p_e \wedge a_2 \wedge \bigcirc b) \vee (\neg a_1 \wedge \bigcirc^2 c) \vee (\neg p_e \wedge \bigcirc\neg b)$$

We choose a pair of moves, $(p_e \wedge a_1 \wedge \bigcirc b)$ and $(p_e \wedge a_2 \wedge \bigcirc b)$, that do not satisfy the TNF *conditions and then we apply the equivalence,*

$$\begin{aligned}(p_e \wedge a_1 \wedge \bigcirc b) \vee (p_e \wedge a_2 \wedge \bigcirc b) \quad \equiv \quad & (p_e \wedge a_1 \wedge a_2 \wedge \bigcirc b) \vee \\ & (p_e \wedge a_1 \wedge \neg a_2 \wedge \bigcirc b) \vee \\ & (p_e \wedge \neg a_1 \wedge a_2 \wedge \bigcirc b)\end{aligned}$$

obtaining the following formula, which also does not fulfil the conditions,

$$\begin{gathered}(p_e \wedge a_1 \wedge a_2 \wedge \bigcirc b) \vee (p_e \wedge a_1 \wedge \neg a_2 \wedge \bigcirc b) \vee \\ \underline{(p_e \wedge \neg a_1 \wedge a_2 \wedge \bigcirc b)} \vee \underline{(\neg a_1 \wedge \bigcirc^2 c)} \vee (\neg p_e \wedge \bigcirc\neg b)\end{gathered}$$

We apply the equivalence to the underlined formulas,

$$\begin{aligned}(p_e \wedge \neg a_1 \wedge a_2 \wedge \bigcirc b) \vee (\neg a_1 \wedge \bigcirc^2 c) \quad \equiv \quad & (p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc b \vee \bigcirc^2 c)) \vee \\ & \mathrm{DNF}(\neg a_1 \wedge \neg(p_e \wedge a_2) \wedge \bigcirc^2 c)\end{aligned}$$

where,

$$\mathrm{DNF}(\neg a_1 \wedge \neg(p_e \wedge a_2) \wedge \bigcirc^2 c) \quad \equiv \quad (\neg a_1 \wedge \neg a_2 \wedge \bigcirc^2 c) \vee (\neg a_1 \wedge \neg p_e \wedge \bigcirc^2 c)$$

We obtain the following formula, which also does not fulfil the conditions,

$$\begin{gathered}(p_e \wedge a_1 \wedge a_2 \wedge \bigcirc b) \vee (p_e \wedge a_1 \wedge \neg a_2 \wedge \bigcirc b) \vee \\ (p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc b \vee \bigcirc^2 c)) \vee (\neg a_1 \wedge \neg a_2 \wedge \bigcirc^2 c) \vee \\ \underline{(\neg a_1 \wedge \neg p_e \wedge \bigcirc^2 c)} \vee \underline{(\neg p_e \wedge \bigcirc\neg b)}\end{gathered}$$

We apply the equivalence,

$$\begin{aligned}(\neg a_1 \wedge \neg p_e \wedge \bigcirc^2 c) \vee (\neg p_e \wedge \bigcirc\neg b) \quad \equiv \quad & (\neg a_1 \wedge \neg p_e \wedge (\bigcirc^2 c \vee \bigcirc\neg b)) \vee \\ & (a_1 \wedge \neg p_e \wedge \bigcirc b)\end{aligned}$$

We obtain the following formula, which also does not fulfil the conditions,

$$\begin{aligned}&(p_e \wedge a_1 \wedge a_2 \wedge \bigcirc b) \vee (p_e \wedge a_1 \wedge \neg a_2 \wedge \bigcirc b) \vee (p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc b \vee \bigcirc^2 c)) \vee \\ &\quad \underline{(\neg a_1 \wedge \neg a_2 \wedge \bigcirc^2 c)} \vee \underline{(\neg a_1 \wedge \neg p_e \wedge (\bigcirc^2 c \vee \bigcirc\neg b))} \vee (a_1 \wedge \neg p_e \wedge \bigcirc b))\end{aligned}$$

again we apply the equivalence,

$$\begin{aligned}(\neg a_1 \wedge \neg a_2 \wedge \bigcirc^2 c) \vee \quad \equiv \quad & (\neg p_e \wedge \neg a_1 \wedge \neg a_2 \wedge (\bigcirc^2 c \vee \bigcirc\neg b)) \vee \\ (\neg a_1 \wedge \neg p_e \wedge (\bigcirc^2 c \vee \bigcirc\neg b)) \quad\quad & (p_e \wedge \neg a_1 \wedge \neg a_2 \wedge \bigcirc^2 c) \vee \\ & (\neg p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc^2 c \vee \bigcirc\neg b))\end{aligned}$$

and, finally, we obtain the equivalent TNF*(*ψ*):*

$$\begin{aligned}\mathrm{TNF}(\psi) \quad \equiv \quad & (p_e \wedge a_1 \wedge a_2 \wedge \bigcirc b) \vee \\ & (p_e \wedge a_1 \wedge \neg a_2 \wedge \bigcirc b) \vee \\ & (p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc b \vee \bigcirc^2 c)) \vee \\ & (\neg p_e \wedge \neg a_1 \wedge \neg a_2 \wedge (\bigcirc^2 c \vee \bigcirc\neg b)) \vee \\ & (p_e \wedge \neg a_1 \wedge \neg a_2 \wedge \bigcirc^2 c) \vee \\ & (\neg p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc^2 c \vee \bigcirc\neg b)) \vee \\ & (a_1 \wedge \neg p_e \wedge \bigcirc b))\end{aligned}$$

4.4 Example. *As mentioned above,* TNF *of a safety specification may not be unique. Given the safety specification* $\Box\psi \equiv (p_e \wedge \neg c_1 \wedge c_2 \wedge \bigcirc s) \vee (\neg c1 \wedge \bigcirc b)$*, we calculate the* TNF *by applying the equivalence:*

$$\begin{aligned}(p_e \wedge \neg c_1 \wedge c_2 \wedge \bigcirc b) \vee (\neg c1 \wedge \bigcirc s) \quad \equiv \quad & (p_e \wedge \neg c_1 \wedge c_2 \wedge (\bigcirc b \vee \bigcirc s)) \vee \\ & \mathrm{DNF}(\neg c_1 \wedge \neg(p_e \wedge c_2) \wedge \bigcirc s)\end{aligned}$$

$\mathrm{DNF}(\neg c_1 \wedge \neg(p_e \wedge c_2) \wedge \bigcirc s)$ *not only is equivalent to* $((\neg c_1 \wedge \neg c_2 \wedge \bigcirc s) \vee (\neg c_1 \wedge \neg p_e \wedge \bigcirc s))$ *but also to* $((\neg c_1 \wedge \neg c_2 \wedge \neg p_e \wedge \bigcirc s) \vee (\neg c_1 \wedge \neg c_2 \wedge p_e \wedge \bigcirc s) \vee (\neg c_1 \wedge c_2 \wedge \neg p_e \wedge \bigcirc s))$*. Therefore, there are at least two* TNF*(*ψ*):*

$$\begin{aligned}\mathit{TNF}_1(\psi) \quad \equiv \quad & ((p_e \wedge \neg c_1 \wedge c_2 \wedge (\bigcirc b \vee \bigcirc s)) \vee ((\neg c_1 \wedge \neg c_2 \wedge \bigcirc s) \vee \\ & (\neg c_1 \wedge \neg p_e \wedge \bigcirc s)))\end{aligned}$$

$$\begin{aligned}\mathit{TNF}_2(\psi) \quad \equiv \quad & (p_e \wedge \neg c_1 \wedge c_2 \wedge (\bigcirc b \vee \bigcirc s)) \vee ((\neg c_1 \wedge \neg c_2 \wedge \neg p_e \wedge \bigcirc s) \vee \\ & (\neg c_1 \wedge \neg c_2 \wedge p_e \wedge \bigcirc s) \vee (\neg c_1 \wedge c_2 \wedge \neg p_e \wedge \bigcirc s)\end{aligned}$$

4.5 Example. *Let* $\Box\psi$ *be the safety specification of Running Example 3.3, where* $\mathcal{X} = \{p_e, r_e\}$, $\mathcal{Y} = \{g1, g2\}$ *and* DNF(ψ) *formula be as follows:*

$$\begin{aligned}
DNF(\psi) \quad \equiv \quad & (\neg g2 \wedge r1_e \wedge g1 \wedge \neg r2_e) \vee \\
& (\neg g2 \wedge r1_e \wedge \neg r2_e \wedge \bigcirc\Diamond_{[0,2]}(g1)) \vee \\
& (\bigcirc\Diamond_{[0,2]}(g1) \wedge r2_e \wedge \neg g1 \wedge \bigcirc\Diamond_{[0,2]}(g2)) \vee \\
& (\neg g2 \wedge r1_e \wedge g1 \wedge \bigcirc\Diamond_{[0,2]}(g2)) \vee \\
& (g2 \wedge r1_e \wedge \neg g1 \wedge \bigcirc\Diamond_{[0,2]}(g1)) \vee \\
& (\neg g2 \wedge \neg r1_e \wedge \neg\bigcirc g2 \wedge \neg r2_e) \vee \\
& (\neg g2 \wedge \neg r1_e \wedge r2_e \wedge \bigcirc\Diamond_{[0,2]}(g2)) \vee \\
& (g2 \wedge \neg r1_e \wedge \neg g1 \wedge r2_e) \vee \\
& (\neg\bigcirc g2 \wedge \neg r1_e \wedge \neg g1 \wedge \neg r2_e)
\end{aligned}$$

After applying the equivalence 4.1 until all pairs satisfy the conditions, we obtain the following TNF.

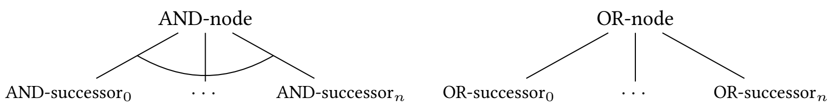
$$\begin{aligned}
\mathrm{TNF}(\psi) \quad \equiv \quad & (r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]} g1) \vee \\
& (r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge ((\bigcirc\Diamond_{[0,2]} g1 \wedge \bigcirc\Diamond_{[0,2]} g2) \vee (\bigcirc\Diamond_{[0,2]} g1))) \vee \\
& (r1_e \wedge r2_e \wedge \neg g1 \wedge \neg g2 \wedge (\bigcirc\Diamond_{[0,2]} g1 \wedge \bigcirc\Diamond_{[0,2]} g2)) \vee \\
& (r1_e \wedge \neg r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc\Diamond_{[0,2]} g1) \vee \\
& (r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge (\bigcirc\Diamond_{[0,2]} g1 \vee \bigcirc\Diamond_{[0,2]} g2 \vee \bigcirc True)) \vee \\
& (r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]} g2) \vee \\
& (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg\bigcirc g2) \vee \\
& (\neg r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge \neg\bigcirc g2) \vee \\
& (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge g2 \wedge \neg\bigcirc g2) \vee \\
& (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge \neg g2 \wedge ((\bigcirc\Diamond_{[0,2]} g1 \wedge \bigcirc\Diamond_{[0,2]} g2) \vee (\bigcirc\Diamond_{[0,2]} g1))) \vee \\
& (\neg r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]} g2)) \vee \\
& (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge (\bigcirc\Diamond_{[0,2]} g1 \vee \bigcirc\Diamond_{[0,2]} g2 \vee \bigcirc True))
\end{aligned}$$

Note that when an implicant does not contain futures, the implicit future that it represents is $\bigcirc True$. *Moreover, the subsumption of strict-future formulas as* $(\bigcirc\Diamond_{[0,2]} g1 \vee \bigcirc\Diamond_{[0,2]} g2 \vee \bigcirc True) \equiv \bigcirc True$ *will be explained in Section 5.1 with the aim of minimizing strict-future formulas.*

CHAPTER 5

Realizability Tableaux

Our realizability tableaux (from now only tableaux) are AND-OR trees of nodes, where each node is labeled by a set of formulas. A node is said to be the parent of its successor nodes, which may have 0, 1 or more successors. In addition, successors can be of two types, AND-successors or OR-successors. As we can see in Figure 5.1, the main visual difference representing the type of successors is that AND-successors are represented with a semicircle embracing all the edges.



(a) AND-successors **(b)** OR-successor

Figure 5.1: Types of successors

A tableau is constructed from an input safety specification in Terse Normal Form, which is the root of the tree, by applying a set of rules that determine its development. When no rule can be applied to a node, it is called leaf. There are two types of leaves:

1. Failure leaves, labelled by inconsistent sets of formulas[1], indicates that the branch from the root to the leaf is failed.

2. Successful leaves, labelled by sets of formulas, are subsumed[2] by some previous node in the branch from the root to the leaf.

5.1 Definition (successful/failed node)**.** *A node success (resp. failure) depends on the types of siblings it generates. An AND-node is successful (resp. failed) whether each successor returns an open branch (resp. if one successor returns a closed branch), whereas for a successful OR-node is enough if one of its siblings returns an open branch (resp. every successor returns a closed branch).*

[1] Subsumption concept is explained below in Section 5.1

[2] Inconsistency concept is explained below in Section 5.1

5.2 Definition. *A tableau is completed (or finished) when no further rule can be applied to it.*

The following definition formalizes our notion of tableau in terms of many concepts that will be precised below.

5.3 Definition. *A tableau for a safety specification $\varphi = \alpha \wedge \Box\psi$ is a labelled tree* $\mathrm{Tab}(\varphi) = (N, \tau, R)$*, where:*

- *N is a set of nodes*
- *τ is a mapping of the nodes with the set of formulas*
- *$R \subseteq N \times N$ represents the transition from one node to other,*

and such that the following conditions hold:

- *The root is labelled by the set $\{\alpha, \Box\psi\}$.*
- *For any pair of nodes $(n, n') \in R$, $\tau(n')$ is the set of formulas obtained as the result of the application of one of the tableau rules to $\tau(n)$. Given the applied rule is ρ, we term n' a ρ-successor of n*
- *For every success or failure leaf n there is no $n' \in N$ such that $(n, n') \in R$ where:*
  - *A failure leaf is a node such that $n \in N$ such that $\tau(n)$ is inconsistent.*
  - *A success leaf is a node $n \in N$ such that $\Box\psi \in \tau(n)$ and there exists $k \geq 0$, $n_0, \ldots, n_k \in N$ such that $(n_i, n_{i+1}) \in R$ for all $0 \leq i < k$, $(n_k, n) \in R$ and $\tau(n_0) \lessdot \tau(n)$[3].*

5.1 Subsumptions and Inconsistencies

Our tableaux nodes are labeled using a set of formulas that are subsumption- and inconsistency-free. First of all, we will introduce the subsumption concept in boolean and temporal formulas.

5.4 Definition (Subsumption in boolean formulas)**.** *Given two boolean formulas φ and ψ, φ subsumes ψ (i.e $\varphi \sqsubseteq \psi$) iff all models of ψ satisfies φ. Subsumption is related to logical implication or logical consequence in the sense that, if $\varphi \sqsubseteq \psi$, then $\models \varphi \to \psi$ or equivalently $\varphi \models \psi$.*

In temporal formulas, subsumption concept is slightly different due to the fact that a serie of requirements must be fulfilled in temporal intervals depending on the temporal operator. Remark that boolean formulas can be represented as bounded always/eventually with an interval of [0,0] (e.g. $\varphi \equiv \Box_{[0,0]}\varphi \equiv \Diamond_{[0,0]}\varphi$), as well as, next^i formulas can be represented as bounded always/eventually with an interval of [i,i] (e.g. $\bigcirc^i\varphi \equiv \Box_{[i,i]}\varphi \equiv \Diamond_{[i,i]}\varphi$). Therefore, subsumptions in temporal formulas (from now only subsumptions), that we will see in Definition 5.5, includes subsumptions in boolean formulas.

[3] This concept is formally explained in Definition 5.6.

5.5 Definition (Subsumption rules). *Given two boolean formulas, φ and ψ, the subsumption rules that apply in our tableau method are the following:*

For all n, m, n', m' where $0 \leq n \leq n' \leq m' \leq m$ (note that $[n', m'] \subseteq [n, m]$) and $\varphi \sqsubseteq \psi$:

- $\Diamond_{[n',m']}\varphi \sqsubseteq \Diamond_{[n,m]}\psi$,
- $\Diamond_{[n,n]}\varphi \sqsubseteq \Box_{[n,n]}\psi$,
- $\Box_{[n,m]}\varphi \sqsubseteq \Box_{[n',m']}\psi$,
- $\Box_{[n,m]}\varphi \sqsubseteq \Diamond_{[n',m']}\psi$,

5.1 Example. *Given two boolean formulas, a and $(a \vee b)$, and two intervals [2,3] and [0,5] where $a \sqsubseteq (a \vee b)$ and [2,3] $\subseteq$ [0,5]:*

- $\Diamond_{[2,3]}a \sqsubseteq \Diamond_{[0,5]}(a \vee b)$,
- $\Box_{[0,5]}a \sqsubseteq \Box_{[2,3]}(a \vee b)$,
- $\Box_{[0,5]}a \sqsubseteq \Diamond_{[2,3]}(a \vee b)$,

We define the following subsumption-based order relation between sets of formulas for detecting successful leaves.

5.6 Definition (Order relation). *For two given set of formulas Φ and Φ', we say that $\Phi \lessdot \Phi'$ iff for every formula $\varphi \in \Phi$ there exists some $\varphi' \in \Phi'$ such that $\varphi \sqsubseteq \varphi'$.*

5.2 Example. *Given the running example* TNF(ψ) *(see Example 4.5), we apply the following subsumptions to strict-future formulas sets:*

$$\begin{array}{rcl} \{\bigcirc\Diamond_{[0,2]}g1\} & \lessdot & \{\bigcirc\Diamond_{[0,2]}g1, \bigcirc\Diamond_{[0,2]}g2\} \\ \{\bigcirc True\} & \lessdot & \{\bigcirc\Diamond_{[0,2]}g1\} \\ \{\bigcirc True\} & \lessdot & \{\bigcirc\Diamond_{[0,2]}g2\} \end{array}$$

Thus, the TNF *in Example 4.5 becomes the following formula after applying subsumptions.*

$$\begin{array}{rcl} \text{TNF}(\psi) & \equiv & (r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}g1) \vee \\ & & (r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc\Diamond_{[0,2]}g1) \vee \\ & & (r1_e \wedge r2_e \wedge \neg g1 \wedge \neg g2 \wedge (\bigcirc\Diamond_{[0,2]}g1 \wedge \bigcirc\Diamond_{[0,2]}g2)) \vee \\ & & (r1_e \wedge \neg r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc\Diamond_{[0,2]}g1) \vee \\ & & (r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc True) \vee \\ & & (r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}g2) \vee \\ & & (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg\bigcirc g2) \vee \\ & & (\neg r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge \neg\bigcirc g2) \vee \\ & & (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge g2 \wedge \neg\bigcirc g2) \vee \\ & & (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}g1) \vee \\ & & (\neg r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}g2)) \vee \\ & & (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc True) \end{array}$$

Like subsumptions, inconsistencies of temporal formulas (from now only inconsistencies) are an extension of inconsistencies of boolean formulas.

5.7 Definition (Inconsistencies of boolean formula)**.** *Two boolean formulas φ and ψ are inconsistent iff there is no model that satisfies $\varphi \wedge \psi$ which is denoted as $\not\models \varphi \wedge \psi$.*

5.8 Definition (Inconsistencies)**.** *Let the conjunction of φ and ψ be inconsistent, being φ and ψ boolean formulas. The inconsistent rules that apply our tableau method to temporal formulas are the following:*

- $\Diamond[n,m]\varphi$ *and* $\Diamond[n,m]\psi$ *are inconsistent.*
- $\Box[n',m']\varphi$ *and* $\Box[n,m]\psi$ *are inconsistent whenever there exists k such that $k \in [n',m']$ and $k \in [n,m]$.*

5.3 Example. *Given two inconsistent boolean formulas, a and $(\neg a \wedge b)$,*

- $\Diamond[1,3]a$ *and* $\Diamond[1,3](\neg a \wedge b)$ *are inconsistent,*
- $\Box[6,7]a$ *and* $\Box[4,8](\neg a \wedge b)$ *are inconsistent,*
- $\Box[2,6]a$ *and* $\Box[4,8](\neg a \wedge b)$ *are inconsistent,*
- $\Box[7,10]a$ *and* $\Box[4,8](\neg a \wedge b)$ *are inconsistent,*

Inconsistencies are used to close tableau branches. No rule is applied to a node labelled by an inconsistent set, and this node is called a failure leaf.

5.9 Definition. *A node labelled by a set of formulas is inconsistent iff at least two formulas of the set are inconsistent.*

5.2 Minimal covering

Given a current node $\{\Phi, \Box\psi\}$, we define the concept of minimal covering as a possible strategy of the system against environment. In addition, the set of all minimal coverings represents all system possible strategies. Each move in an strategy contains all the strict-future possibilities for this move. Formally,

5.10 Definition ($\mathcal{X}$-covering)**.** *Given a formula $\psi \equiv \bigvee_{i=1}^{n} \pi_i$ in TNF, ψ is a $\mathcal{X}$-covering if and only if*

$$Val \bigcup_{i=1}^{n} Val_{\pi_i}(\mathcal{X}) = Val(\mathcal{X}).$$

5.11 Definition (Minimal $\mathcal{X}$-covering)**.** *Given a formula $\psi \equiv \bigvee_{i=1}^{n} \pi_i$ in TNF, ψ is a minimal $\mathcal{X}$-covering iff it holds the following conditions:*

- *ψ is a $\mathcal{X}$-covering*
- *for every $1 \leq j \leq n$, $\bigvee_{i=1, i\neq j}^{n} \pi_i$ is not an $\mathcal{X}$-covering*

5.4 Example. *Let* TNF(φ) $= (p_e \wedge s \wedge \eta_1) \vee (\neg p_e \wedge s \wedge \eta_2) \vee (\neg s \wedge \eta_3)$ *where* η_1, η_2, η_3 *are strict-future formulas and* $\mathcal{X} = \{p_e\}$*. It contains five* $\mathcal{X}$*-coverings:*

- $(p_e \wedge s \wedge \eta_1) \vee (\neg p_e \wedge s \wedge \eta_2) \vee (\neg s \wedge \eta_3)$
- $(p_e \wedge s \wedge \eta_1) \vee (\neg p_e \wedge s \wedge \eta_2)$
- $(\neg s \wedge \eta_3)$
- $(p_e \wedge s \wedge \eta_1) \vee (\neg s \wedge \eta_3)$
- $(\neg p_e \wedge s \wedge \eta_2) \vee (\neg s \wedge \eta_3)$

two of which are minimal $\mathcal{X}$*-coverings:*

- $(p_e \wedge s \wedge \eta_1) \vee (\neg p_e \wedge s \wedge \eta_2)$
- $(\neg s \wedge \eta_3)$

5.5 Example. *Given Example 4.2,* TNF*(*ψ*)* $\equiv (p_e \wedge s \wedge \bigcirc s) \vee (\neg s \wedge \neg p_e \wedge (\bigcirc^2 s \vee \bigcirc^3 s)) \vee (\neg s \wedge p_e \wedge \bigcirc^2 s)$*, it contains two minimal* $\mathcal{X}$*-covering:*

- $(p_e \wedge s \wedge \bigcirc s) \vee (\neg s \wedge \neg p_e \wedge (\bigcirc^2 s \vee \bigcirc^3 s))$
- $(\neg s \wedge p_e \wedge \bigcirc^2 s) \vee (\neg s \wedge \neg p_e \wedge (\bigcirc^2 s \vee \bigcirc^3 s))$

5.6 Example. *Referring to* TNF *of the Running Example 4.5, it generates 81 minimal* $\mathcal{X}$*-covering due to each four environment valuations has three possible moves. Some of the minimal* $\mathcal{X}$*-covering are* [4]*:*

$$\begin{aligned} C_1 \equiv\ & \{(r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc True), \\ & (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc True), \\ & (r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc \Diamond_{[0,2]} g1), \\ & (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg \bigcirc g2)\} \end{aligned}$$

$$\begin{aligned} C_2 \equiv\ & \{((r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \bigcirc \Diamond_{[0,2]} g1), \\ & (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc True), \\ & (r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc \Diamond_{[0,2]} g1), \\ & (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg \bigcirc g2)\} \end{aligned}$$

$$\begin{aligned} C_3 \equiv\ & \{(r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc True), \\ & (\neg r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc \Diamond_{[0,2]} g2)), \\ & (r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc \Diamond_{[0,2]} g1), \\ & (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg \bigcirc g2)\} \end{aligned}$$

[4] Whenever convenient, we identify the minimal coverings with set of moves.

$$\begin{aligned} C_4 \;\; \equiv \;\; & \{(r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc True), \\ & (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc True), \\ & (r1_e \wedge r2_e \wedge \neg g1 \wedge \neg g2 \wedge (\bigcirc \Diamond_{[0,2]} g1 \wedge \bigcirc \Diamond_{[0,2]} g2)), \\ & (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg \bigcirc g2)\} \end{aligned}$$

The choice of a good minimal $\mathcal{X}$-covering for the construction of the tableau is very important, therefore, we introduce the concepts of weaker moves and weaker minimal covering.

5.12 Definition (weaker moves). *Let π_1, π_2 be two moves such that both contain the same environment literals, $(\mathcal{L}(\pi_1) \cap \mathcal{X}) = (\mathcal{L}(\pi_2) \cap \mathcal{X})$, and η_1 and η_2 be their respective strict-future formulas. If η_2 is a logical consequence of η_1, $\eta_1 \rightarrow \eta_2$, we say that π_2 is weaker than π_1.*

The relation weaker is a partial order and can be generalized to minimal coverings in the following sense.

5.13 Definition (weaker minimal coverings). *Let $\hat{C} = (\hat{\pi}_1 \vee \cdots \vee \hat{\pi}_m)$ and $C = (\pi_1 \vee \cdots \vee \pi_n)$ be two minimal coverings. We say that $\hat{C}$ is weaker than C, denoted as $\hat{C} \leq C$, if for all $1 \leq i \leq m$ exists $1 \leq j \leq n$ such that $\hat{\pi}_i$ is weaker than π_j.*

Suppose you have a safety specification $\alpha \wedge \Box\psi$ whose minimal $\mathcal{X}$-covering are exactly C_1 and C_2 such that C_2 is weaker than C_1, $C_2 \leq C_1$. Our tableau has to choose one of the minimal $\mathcal{X}$-covering to start with and the best action is to start with C_2 due to the fact that if the tableau is closed for C_2, then, we ensure that the tableau C_1 is also closed. Consequently, we do not need to develop C_1.

5.7 Example. *Let $\mathcal{X} = \{p_e\}$ and $\mathcal{Y} = \{a, b, c\}$. Let*

$$\begin{aligned} C_1 &= (p_e \wedge \neg a \wedge \neg c) \vee (\neg p_e \wedge \neg a \wedge \neg c \wedge \bigcirc^2 a) \\ C_2 &= (p_e \wedge \neg a \wedge \neg c) \vee (\neg p_e \wedge c \wedge a \wedge (\bigcirc \neg c \vee \bigcirc^2 a)) \end{aligned}$$

For the evaluation of p_e to True, the two minimal covering have the same future, $\bigcirc True$, and for the evaluation of p_e to False, $(\bigcirc \neg c \vee \bigcirc^2 a)$ is logical consequence of $\bigcirc^2 a$, therefore, $C_2 \leq C_1$ and our tableau will start with C_2. In case of returning a closed tableau for C_2 as a result of its unrealizability, C_1 will not be realizable too.

We formalize these ideas in the next propositions.

5.1 Proposition. *Let $\hat{C} = \{\hat{\pi}_1, \cdots \hat{\pi}_m\}$ and $C = \{\pi_1, \cdots \pi_n\}$ be two minimal coverings. Let $\hat{C} \leq C$. If C is realizable, then $\hat{C}$ is realizable.*

Proof. Suppose C is realizable, that means, for all $1 \leq j \leq n$, π_j is realizable and, consequently, $\mathcal{F}(\pi_j)$ is realizable. Since $\hat{C} \leq C$, by Definitions (5.12) and (5.13), for all $1 \leq i \leq m$, exists $1 \leq j \leq n$ such that $\mathcal{F}(\hat{\pi}_i)$ is a logical consequence of $\mathcal{F}(\pi_j)$. Hence, for all $1 \leq i \leq m$, $\hat{\pi}_i$ is realizable and $\hat{C}$ is realizable.

5.2 Proposition. *Let $\alpha \wedge \Box\psi$ be a safety specification. Let $\hat{C}_1, \cdots, \hat{C}_m$ be all the weakest minimal coverings included in* TNF$(\alpha \wedge \psi)$*. The specification $\alpha \wedge \Box\psi$ is realizable iff there exists $1 \leq i \leq m$ such that $(\hat{C}_i \wedge \bigcirc\Box\psi)$ is realizable.*

Proof. (Backward direction) If there exists a weakest minimal covering such that $(\hat{C}_i \wedge \bigcirc\Box\psi)$ is realizable, in particular exists a minimal covering.

Proof. (Forward direction) Suppose that $\alpha \wedge \Box\psi$ is realizable. Hence, there exists $C = \{\pi_1, \cdots \pi_n\}$ a minimal covering included in TNF$(\alpha \wedge \psi)$, such that $(C \wedge \bigcirc\Box\psi)$ is realizable. If C is a weakest minimal covering we are done. Otherwise, there exists $1 \leq i \leq m$ such that $\hat{C}_i < C$. Consequently, by Proposition 5.2, $\hat{C}_i$ is realizable and $(\hat{C}_i \wedge \bigcirc\Box\psi)$ is realizable

Once a minimal covering has been chosen, we are interested in starting to build the branch with the move that contains the strongest strict-futures formulas.

5.14 Definition (stronger move). *Let π_1, π_2 be two moves of a minimal covering C such that η_1 and η_2 are their respective strict-future formulas. If η_2 is a logical consequence of η_1, $\eta_1 \to \eta_2$, we say that π_1 has stronger strict future formulas than π_2, consequently, π_1 is stronger than π_2.*

5.8 Example. *Given the following minimal covering:*

$$C = \underbrace{(p_e \wedge \neg a \wedge \neg c \wedge \bigcirc^2 a)}_{\pi_1} \vee \underbrace{(\neg p_e \wedge c \wedge a \wedge (\bigcirc\neg c \vee \bigcirc^2 a))}_{\pi_2}$$

Our tableau will develop first the branch with the move π_1 due to the fact that $\mathcal{F}(\pi_1)$ is stronger than $\mathcal{F}(\pi_2)$, (i.e. $\bigcirc^2 a \to (\bigcirc\neg c \vee \bigcirc^2 a)$). Then, if π_1 returns a closed tableau because its not realizable, all the minimal covering C will be not realizable without the need of developing π_2. On the other hand, if π_1 returns an open tableau, realizability of C will depend on π_2.

5.3 SAT-Based TNF Computation

Any tableau for a safety specification, $\alpha \wedge \Box\psi$, has as its first objective to find a set of minimal coverings from the TNF$(\alpha \wedge \psi)$.

In Section 4.2, we proposed a theoretical method to achieve the full TNF, nevertheless, in this section we will explain how to compute it in an efficient way in order to obtain the weakest minimal coverings in a more direct way.

Given a safety specification, $\alpha \wedge \Box\psi$, the first step is to interpret each strict-future formulas as boolean literals in order to calculate a short DNF for $(\alpha \wedge \psi)$, i.e., a DNF logically equivalent to $(\alpha \wedge \psi)$. This DNF is the representation of all possible moves at some state of the game.

There are automatic tools that find DNFs for propositional formulas reasonably well, but this task is not easy: the problem of deciding whether a propositional formula has a DNF of size is EXPTIME complete [30]. We are using a recent tool called BICA [8], which is able to compute the minimum (size) prime implicants (see Section 2.4) equivalent to a propositional formula in an arbitrary form.

Once we have a DNF for $(\alpha \wedge \psi)$, the next step consists in associating with each move all the possibilities for the (strict) future which are coherent with $\alpha \wedge \psi$. For that purpose we construct the TNF$(\alpha \wedge \psi)$. After that, we are in a position to choose a weakest minimal covering to start with.

5.9 Example. *Let be the safety specification of Example 4.3 with the following equivalent* TNF*:*

$$
\begin{aligned}
\mathrm{TNF}(\psi) \quad \equiv \quad & (p_e \wedge a_1 \wedge a_2 \wedge \bigcirc b) \vee \\
& (p_e \wedge a_1 \wedge \neg a_2 \wedge \bigcirc b) \vee \\
& (p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc b \vee \bigcirc^2 c)) \vee \\
& (\neg p_e \wedge \neg a_1 \wedge \neg a_2 \wedge (\bigcirc^2 c \vee \bigcirc \neg b)) \vee \\
& (p_e \wedge \neg a_1 \wedge \neg a_2 \wedge \bigcirc^2 c) \vee \\
& (\neg p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc^2 c \vee \bigcirc \neg b)) \vee \\
& (a_1 \wedge \neg p_e \wedge \bigcirc b))
\end{aligned}
$$

Now we have to choose a minimal covering. According to Proposition 5.2, any tableau that decides the realizability of $\Box\psi$ should take a weakest one. In this case,

$$\{(p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc b \vee \bigcirc^2 c)), (\neg p_e \wedge \neg a_1 \wedge \neg a_2 \wedge (\bigcirc^2 c \vee \bigcirc \neg b))\}$$

Consequently, moves $(p_e \wedge a_1 \wedge a_2 \wedge \bigcirc b), (p_e \wedge \neg a_1 \wedge \neg a_2 \wedge \bigcirc^2 c), (\neg p_e \wedge \neg a_1 \wedge a_2 \wedge (\bigcirc^2 c \vee \bigcirc \neg b)), (p_e \wedge a_1 \wedge \neg a_2 \wedge \bigcirc b), (\neg p_e \wedge a_1 \wedge \bigcirc \neg b)$ are irrelevant for the tableau.

Previous example clearly shows that some information of TNF is redundant. Hence, it suggests that a clever process could be find to construct a TNF with only weakest minimal coverings.

Before presenting the improvement algorithm, it is necessary to formalize the definition of join operator, compatible sets and the notion of extending a move with environment literals.

5.15 Definition (join operator and compatible sets)**.** *Let* $\Pi = \{\pi_1, \cdots, \pi_n\}$ *a set of moves. The set* Π *is compatible if for all* $1 \leq i \neq j \leq n$ *and for all literal* $\ell \in \mathcal{L}(\pi_i)$*, it holds that* $\neg\ell \notin \mathcal{L}(\pi_j)$*. The join of moves in a compatible set* Π*, denoted as* $join(\Pi)$*, is a new move with* $\mathcal{L}(join(\Pi)) = \mathcal{L}(\pi_1) \cup \cdots \cup \mathcal{L}(\pi_n)$ *and* $\mathcal{F}(join(\Pi)) = \mathcal{F}(\pi_1) \vee \mathcal{F}(\pi_2) \cdots \vee \mathcal{F}(\pi_n)$*.*

5.3 Proposition. *Let* $\Pi = \{\pi_1, \cdots, \pi_n\}$ *a compatible set of moves. The following holds:*

1. *The move* $join(\Pi)$ *is weaker than any move of* Π*.*
2. $\bigvee_{i=1}^{n} \pi_i$ *is a logical consequence of* $join(\Pi)$*.*

Proof. Item 1. holds by Definition 5.13 and the fact that the disjunction $\mathcal{F}(\pi_1) \vee \mathcal{F}(\pi_2) \vee \cdots \vee \mathcal{F}(\pi_n)$ is a logical consequence of each strict-future formula of π_i for $1 \leq i \leq n$.

Proof. Item 2. is based on the process explained in Proposition 4.1. There, Equation (4.1) can be seen in terms of the $join$ operator.

$$\begin{aligned}(\delta \wedge \delta_1 \wedge \eta_1) \vee (\delta \wedge \delta_2 \wedge \eta_2) \quad \equiv \quad & join((\delta \wedge \delta_1 \wedge \eta_1), (\delta \wedge \delta_2 \wedge \eta_2)) \\ & \vee \ \mathrm{DNF}(\delta \wedge \delta_1 \wedge \neg\delta_2 \wedge \eta_1) \\ & \vee \ \mathrm{DNF}(\delta \wedge \neg\delta_1 \wedge \delta_2 \wedge \eta_2)\end{aligned}$$

Here, the formula $(\delta \wedge \delta_1 \wedge \eta_1) \vee (\delta \wedge \delta_2 \wedge \eta_2)$ clearly is a logical consequence of $join((\delta \wedge \delta_1 \wedge \eta_1), (\delta \wedge \delta_2 \wedge \eta_2))$ but not the other way round. Therefore, Π is also a logical consequence of $join(\Pi)$.

5.16 Definition (extension of moves with $\mathcal{X}$-literals)**.** *Let* ℓ_e *be a literal of* $\mathcal{X}$ *and* π *be a move. The extension of* π *with* ℓ_e*,* $ext(\pi, \ell_e)$*, is a new move with* $\mathcal{F}(ext(\pi, \ell_e)) = \mathcal{F}(\pi)$ *and* $\mathcal{L}(ext(\pi, \ell_e)) = \mathcal{L}(\pi) \cup \{\ell_e\}$*.*

Note that when $\neg\ell_e \in \pi$, then $ext(\pi, \ell_e) = False$. We can extend the previous definition in to ways.

5.17 Definition (extension of moves with sets of $\mathcal{X}$)**.** *Let* $\mathcal{S}$ *be a set of* $\mathcal{X}$ *and let* π *be a move. The extension of* π *with* $\mathcal{S}$*,* $set\_ext(\pi, \mathcal{S})$ *is the successive extensions of* π *with the variables of* $\mathcal{S}$ *and the negation of variables in* $\mathcal{X} \setminus \mathcal{S}$*. For convenience, we interpret the empty set of moves as* ***T***

When M is a set of moves $M = \{\pi_1, \cdots, \pi_n\}$ and $\mathcal{S}$ is a set of $\mathcal{X}$, we define the extension of M with $\mathcal{S}$ as the set $\{set\_ext(\pi_1, \mathcal{S}), \cdots, set\_ext(\pi_n, \mathcal{S})\}$.

5.10 Example. *Let* $\mathcal{X} = \{p_e, q_e, r_e\}$ *and* $\mathcal{Y} = \{a\}$*, when* $\mathcal{S} = \{p_e, q_e\}$ *and* $M = \{(p_e \wedge a \wedge \eta_1), (r_e \wedge \neg a \wedge \eta_2), (a \wedge \eta_4)\}$*,*

- $set\_ext(M, \mathcal{S}) = \{(p_e \wedge q_e \wedge \neg r_e \wedge a \wedge \eta_1), (p_e \wedge q_e \wedge \neg r_e \wedge a \wedge \eta_4)\}$
- $set\_ext(M, \emptyset) = \{(\neg p_e \wedge \neg q_e \wedge \neg r_e \wedge a \wedge \eta_4)\}$
- $set\_ext(\emptyset, \mathcal{S}) = \{(p_e \wedge q_e \wedge \neg r_e)\}$

Algorithm 1: TNF_Construction(DNF(γ)) returns $\mathcal{T}$

```
% The formula γ is over variables 𝒳 ∪ 𝒴
M := {π : π is a move in DNF(γ)};
𝒯 := ∅;
for any set 𝒮 ∈ 2^(𝒳∩var(M)) do
    Calculate the largest compatible sets
        Π_1^𝒮, ··· Π_n^𝒮 in ext(M, 𝒮);
    𝒥 := {join(Π_1^𝒮), ··· , join(Π_n^𝒮)};
    for 1 ≤ i ≠ j ≤ n do
        if join(Π_i^𝒮) is weaker than join(Π_j^𝒮) then
            𝒥 := 𝒥 \ {join(Π_j^𝒮)}
        end
    end
    𝒯 := 𝒯 ∪ 𝒥;
end
return 𝒯;
```

Algorithm 1 shows the process of building a TNF [5], $\mathcal{T}$, whose minimal coverings are the weakest ones. The size of $\mathcal{T}$ is $O(2^{|\mathcal{X}|} \times |\text{DNF}(\gamma)|)$.

5.11 Example. *We will start with Example 4.2 , where* $\mathcal{X} = \{p_e\}$, $\mathcal{Y} = \{s\}$ *and* $\text{DNF}(\psi) \equiv (p_e \wedge s \wedge \bigcirc s) \vee (\neg s \wedge \bigcirc^2 s) \vee (\neg p_e \wedge \neg s \wedge \bigcirc^3 s)$. *The* TNF *construction is based on Algorithm 1 as follows:*

In the first iteration, it calculates the biggest compatible sets with $\{p_e\}$, $set\_ext(\text{DNF}(\psi), \{p_e\})$*:*

$$\Pi_1^{\{p_e\}} = join(\Pi_1^{\{p_e\}}) = \{(p_e \wedge s \wedge \bigcirc s)\}$$

$$\Pi_2^{\{p_e\}} = join(\Pi_2^{\{p_e\}}) = \{(p_e \wedge \neg s \wedge \bigcirc^2 s)\}$$

consequently, the TNF, $\mathcal{T}$, *increases with the* $join$*s of* $\Pi_1^{\{p_e\}}$ *and* $\Pi_2^{\{p_e\}}$

$$\mathcal{T} = \{(p_e \wedge s \wedge \bigcirc s), (p_e \wedge \neg s \wedge \bigcirc^2 s)\}$$

In the second iteration, $ext(\text{DNF}(\psi), \emptyset)$ *and its corresponding* $join$ *of two moves is calculated,*

$$\Pi_1^{\emptyset} = \{(\neg p_e \wedge \neg s \wedge \bigcirc^3 s), (\neg p_e \wedge \neg s \wedge \bigcirc^2 s)\}$$

$$join(\Pi_1^{\emptyset}) = \{(\neg p_e \wedge \neg s \wedge (\bigcirc^3 s \vee \bigcirc^2 s)\}$$

increasing $\mathcal{T}$ *with the* $join(\Pi_1^{\emptyset})$ *and returning*

$$\mathcal{T} = \{(p_e \wedge s \wedge \bigcirc s), (p_e \wedge \neg s \wedge \bigcirc^2 s), (\neg p_e \wedge \neg s \wedge (\bigcirc^3 s \vee \bigcirc^2 s)\}$$

5.12 Example. *Given the* DNF *of the safety specification of Example 4.3 where* $\mathcal{X} = \{p_e\}$*:*

$$\text{DNF}(\psi) \equiv (p_e \wedge a_1 \wedge \bigcirc b) \vee (p_e \wedge a_2 \wedge \bigcirc b) \vee (\neg a_1 \wedge \bigcirc^2 c) \vee (\neg p_e \wedge \bigcirc \neg b)$$

[5] Algorithm is proof and correctness beyond the scope of this Master Thesis

The execution of Algorithm 1 calculates

$$\begin{aligned}
&\Pi_1^{\{p_e\}} = \{(p_e \wedge a_1 \wedge \bigcirc b), (p_e \wedge a_2 \wedge \bigcirc b)\} \\
&join(\Pi_1^{\{p_e\}}) = \{(p_e \wedge a_1 \wedge a_2 \wedge \bigcirc b)\} \\
&\Pi_2^{\{p_e\}} = \{(p_e \wedge a_2 \wedge \bigcirc b), (p_e \wedge \neg a_1 \wedge \bigcirc^2 c)\} \\
&join(\Pi_2^{\{p_e\}}) = \{(p_e \wedge a_2 \wedge \neg a_1 \wedge (\bigcirc b \vee \bigcirc^2 c))\} \\
&\Pi_1^{\emptyset} = \{(\neg p_e \wedge \neg a_1 \wedge \bigcirc^2 c), (\neg p_e \wedge \bigcirc \neg b)\} \\
&join(\Pi_1^{\emptyset}) = \{(\neg p_e \wedge \neg a_1 \wedge (\bigcirc^2 c \vee \bigcirc \neg b))\}
\end{aligned}$$

For p_e, the move $join(\Pi_2^{\{p_e\}})$ is weaker than $join(\Pi_1^{\{p_e\}})$. Hence,

$$\mathcal{T} = \{(p_e \wedge a_2 \wedge \neg a_1 \wedge (\bigcirc b \vee \bigcirc^2 c), (\neg p_e \wedge \neg a_1 \wedge (\bigcirc^2 c \vee \bigcirc \neg b))\}$$

Note that $\mathcal{T}$ contains a single (weakest) minimal covering.

5.13 Example. *Let $\mathcal{X} = \{p_e\}$ and* DNF $\equiv (p_e \wedge \neg c_1 \wedge c_2 \wedge \bigcirc s) \vee (\neg c1 \wedge \bigcirc b)$ *(see Example 4.4). Algorithm 1 executes two iterations corresponding to $\{p_e\}, \emptyset$ and calculates the following sets.*

$$\begin{aligned}
&\Pi_1^{\{p_e\}} = \{(p_e \wedge \neg c_1 \wedge c_2 \wedge \bigcirc s) \vee (p_e \wedge \neg c1 \wedge \bigcirc b)\} \\
&join(\Pi_1^{\{p_e\}}) = \{(p_e \wedge \neg c_1 \wedge c_2 \wedge (\bigcirc s \vee \bigcirc b))\} \\
&\Pi_1^{\emptyset} = \{((p_e \wedge \neg c1 \wedge \bigcirc b)\} \\
&join(\Pi_1^{\emptyset}) = \{(\neg p_e \wedge \neg c1 \wedge \bigcirc b)\}
\end{aligned}$$

Returning the following TNF*,*

$$\mathcal{T} = \{(p_e \wedge \neg c_1 \wedge c_2 \wedge (\bigcirc s \vee \bigcirc b)), (\neg p_e \wedge \neg c1 \wedge \bigcirc b))\}$$

5.14 Example. *Referring to Running Example 4.5, where $\mathcal{X} = \{r1_e, r2_e\}$, $\mathcal{Y} = \{g1, g2\}$, the DNF is as follows.*

$$\begin{aligned}
\mathrm{DNF}(\psi) \quad \equiv \quad & (r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2) \vee \\
& (r1_e \wedge \neg r2_e \wedge \neg g2 \wedge \bigcirc \Diamond_{[0,2]}(g1)) \vee \\
& (r2_e \wedge \neg g1 \wedge \bigcirc \Diamond_{[0,2]}(g2) \wedge \bigcirc \Diamond_{[0,2]}(g1) \wedge) \vee \\
& (r1_e \wedge g1 \wedge \neg g2 \wedge \bigcirc \Diamond_{[0,2]}(g2)) \vee \\
& (r1_e \wedge \neg g1 \wedge g2 \wedge \bigcirc \Diamond_{[0,2]}(g1)) \vee \\
& (\neg r1_e \wedge \neg r2_e \wedge \neg g2 \wedge \neg \bigcirc g2) \vee \\
& (\neg r1_e \wedge r2_e \wedge \neg g2 \wedge \bigcirc \Diamond_{[0,2]}(g2)) \vee \\
& (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2) \vee \\
& (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg \bigcirc g2)
\end{aligned}$$

The execution of Algorithm 1 calculates

$$\begin{aligned}
\Pi_1^{\{r1_e,r2_e\}} &= \{(r1_e \wedge r2_e \wedge \neg g1 \wedge \bigcirc\Diamond_{[0,2]}(g2) \wedge \bigcirc\Diamond_{[0,2]}(g1)\wedge), \\
&\quad (r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \bigcirc\Diamond_{[0,2]}(g1))\} \\
join(\Pi_1^{\{r1_e,r2_e\}}) &= \{(r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \Diamond_{[0,2]}(g1))\}^{6}
\end{aligned}$$

$$\begin{aligned}
\Pi_2^{\{r1_e,r2_e\}} &= \{(r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g2))\} \\
join(\Pi_2^{\{r1_e,r2_e\}}) &= \{(r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g2))\}
\end{aligned}$$

$$\begin{aligned}
\Pi_1^{\{r1_e\}} &= \{(r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2), (r1_e \wedge \neg r2_e \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g1)), \\
&\quad (r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g2))\} \\
join(\Pi_1^{\{r1_e\}}) &= \{(r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2)\}^{7}
\end{aligned}$$

$$\begin{aligned}
\Pi_2^{\{r1_e\}} &= \{(r1_e \wedge \neg r2_e \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g1)), \\
&\quad (r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g2))\} \\
join(\Pi_2^{\{r1_e\}}) &= \{(r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2 \wedge (\bigcirc\Diamond_{[0,2]}(g1) \vee \bigcirc\Diamond_{[0,2]}(g2))\}
\end{aligned}$$

$$\begin{aligned}
\Pi_1^{\{r2_e\}} &= \{(\neg r1_e \wedge r2_e \wedge \neg g1 \wedge \bigcirc\Diamond_{[0,2]}(g2) \wedge \bigcirc\Diamond_{[0,2]}(g1)), \\
&\quad (\neg r1_e \wedge r2_e \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g2))\} \\
join(\Pi_1^{\{r2_e\}}) &= \{(\neg r1_e \wedge r2_e \wedge \neg g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g2))\}^{8}
\end{aligned}$$

$$\begin{aligned}
\Pi_2^{\{r2_e\}} &= \{(\neg r1_e \wedge r2_e \wedge \neg g1 \wedge \bigcirc\Diamond_{[0,2]}(g2) \wedge \bigcirc\Diamond_{[0,2]}(g1)), \\
&\quad (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2)\} \\
join(\Pi_2^{\{r2_e\}}) &= \{(\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2)\}^{9}
\end{aligned}$$

$$\begin{aligned}
\Pi_1^{\emptyset} &= \{(\neg r1_e \wedge \neg r2_e \wedge \neg g2 \wedge \neg\bigcirc g2), (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg\bigcirc g2)\} \\
join(\Pi_1^{\emptyset}) &= \{(\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg\bigcirc g2)\}
\end{aligned}$$

In addition, the move $join(\Pi_1^{\{r1_e\}})$ is weaker than $join(\Pi_2^{\{r1_e\}})$ and the move $join(\Pi_2^{\{r2_e\}})$ is weaker than $join(\Pi_1^{\{r2_e\}})$. Hence, the following TNF is returned

$$\begin{aligned}
\mathcal{T} &= \{r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \Diamond_{[0,2]}(g1)), \\
&\quad (r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc\Diamond_{[0,2]}(g2)), \\
&\quad (r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2), \\
&\quad (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2), \\
&\quad (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg\bigcirc g2)\}
\end{aligned}$$

[2] $\{\bigcirc\Diamond_{[0,2]} g1\} \lessdot \{\bigcirc\Diamond_{[0,2]} g1, \bigcirc\Diamond_{[0,2]} g2\}$

[3] $\{\bigcirc True\} \lessdot \{\bigcirc\Diamond_{[0,2]} g2\}$

[4] $\{\bigcirc\Diamond_{[0,2]} g2\} \lessdot \{\bigcirc\Diamond_{[0,2]} g2, \bigcirc\Diamond_{[0,2]} g1\}$

[5] $\{\bigcirc True\} \lessdot \{\bigcirc\Diamond_{[0,2]} g2, \bigcirc\Diamond_{[0,2]} g1\}$

which also contains the weakest minimal covering:

$$\begin{aligned} C1 \;=\; & \{r1_e \wedge r2_e \wedge \neg g1 \wedge g2 \wedge \Diamond_{[0,2]}(g1)), \\ & (r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2), \\ & (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2), \\ & (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg \bigcirc g2)\} \end{aligned}$$

$$\begin{aligned} C2 \;=\; & \{(r1_e \wedge r2_e \wedge g1 \wedge \neg g2 \wedge \bigcirc \Diamond_{[0,2]}(g2)), \\ & (r1_e \wedge \neg r2_e \wedge g1 \wedge \neg g2), \\ & (\neg r1_e \wedge r2_e \wedge \neg g1 \wedge g2), \\ & (\neg r1_e \wedge \neg r2_e \wedge \neg g1 \wedge \neg g2 \wedge \neg \bigcirc g2)\} \end{aligned}$$

5.4 Tableau rules

In this section, we introduce the tableau rules along with the concepts, notations and properties related with the sets of formulas.

First of all, Always Rules (Figure 5.2) provides a non-deterministic procedure of analyzing the minimal $\mathcal{X}$-coverings in the TNF$(\Phi \wedge \psi)$. The rule $(\Box\&)$ is the only rule that produces AND-successors for splitting the moves of each minimal $\mathcal{X}$-covering.

$$(\Box \mathit{False})\quad \frac{\Phi, \Box\psi}{\mathit{False}, \Box\psi} \qquad \text{if } \tau \text{ is not an } \mathcal{X}\text{-covering}$$

$$(\Box\|)\quad \frac{\Phi, \Box\psi}{\bigvee_{i\in J_1} \pi_i, \Box\psi \ \mid \cdots \mid \ \bigvee_{i\in J_m} \pi_i, \Box\psi} \qquad \text{if } J_1, \ldots, J_m \text{ is the collection of all minimal } \mathcal{X}\text{-covering of } \tau$$

$$(\Box\&)\quad \frac{\bigvee_{i\in I} \pi_i, \Box\psi}{\pi_1, \bigcirc\Box\psi \ \ \& \ \ldots \ \ \& \ \pi_n, \bigcirc\Box\psi} \qquad \text{if } I \text{ is a minimal } \mathcal{X}\text{-covering}$$

Figure 5.2: Always Rules (where τ denotes TNF$(\Phi \wedge \psi)$)

Then, we introduce the set of rules that are use in the decomposition of formulas into its constituents in the usual way that tableau methods perform it with the so-called saturation. In our method, decomposition of formulas inside the conjunction (or sets) connected by the operator $\ddot{\vee}$ just performs an unfolding formula. The Saturation Rules in Figure 5.3 are used to saturate classical connectives $\wedge$ and $\vee$ (including $\ddot{\vee}$) and temporal operators $\Diamond_I$ and $\Box_I$.

$$(\vee)\ \frac{\Phi, \beta \vee \gamma}{\Phi, \beta \ \mid \ \Phi, \gamma} \qquad\qquad (\ddot{\vee}\,\vee)\ \frac{\Phi, (\eta \wedge (\beta \vee \gamma))\ddot{\vee}\delta}{\Phi, (\eta \wedge \beta)\ddot{\vee}(\eta \wedge \gamma)\ddot{\vee}\delta}$$

$$(\wedge)\ \frac{\Phi, \beta \wedge \gamma}{\Phi, \beta, \gamma}$$

$$(\Diamond<)\ \frac{\Phi,\ \Diamond_{[n,m]}\beta}{\Phi,\ \bigcirc^n\beta \ \mid \ \Phi,\ \bigcirc\Diamond_{[n,m-1]}\beta} \quad \text{if } n < m$$

$$(\ddot{\vee}\Diamond<)\ \frac{\Phi,\ (\eta \wedge \Diamond_{[n,m]}\beta)\ddot{\vee}\delta}{\Phi,\ (\eta \wedge \bigcirc^n\beta)\ddot{\vee}(\eta \wedge \bigcirc\Diamond_{[n,m-1]}\beta\}\ddot{\vee}\delta} \quad \text{if } n < m$$

$$(\Diamond=)\ \frac{\Phi,\ \Diamond_{[n,n]}\beta}{\Phi,\ \bigcirc^n\beta} \qquad\qquad (\ddot{\vee}\Diamond=)\ \frac{\Phi,\ (\eta \wedge \Diamond_{[n,n]}\beta)\ddot{\vee}\delta}{\Phi,\ (\eta \wedge \bigcirc^n\beta)\ddot{\vee}\delta}$$

$$(\Box<)\ \frac{\Phi,\ \Box_{[n,m]}\beta}{\Phi,\ \bigcirc^n\beta,\ \bigcirc\Box_{[n,m-1]}\beta} \quad \text{if } n < m \qquad\qquad (\Box=)\ \frac{\Phi,\ \Box_{[n,n]}\beta}{\Phi,\ \bigcirc^n\beta}$$

$$(\ddot{\vee}\Box<)\ \frac{\Phi,\ (\eta \wedge \Box_{[n,m]}\beta)\ddot{\vee}\delta}{\Phi,\ (\eta \wedge \bigcirc^n\beta \wedge \bigcirc\Box_{[n,m-1]}\beta)\ddot{\vee}\delta} \quad \text{if } n < m$$

$$(\ddot{\vee}\Box=)\ \frac{\Phi,\ (\eta \wedge \Box_{[n,n]}\beta)\ddot{\vee}\delta}{\Phi,\ (\eta \wedge \bigcirc^n\beta)\ddot{\vee}\delta}$$

Figure 5.3: Saturation Rules

Before introducing next-state rule, we need to define when a set of formulas is elementary and the notation of $\eta^{\downarrow}$, where η is strict-future formula.

5.18 Definition (Elementary set of formulas)**.** *A set of formulas Φ is elementary if it consists of a set of literals and one elementary strict-future formula.*

5.19 Definition (Down-arrow formulas)**.** *For any set Φ of next-formulas, $\Phi^{\downarrow} = \{\beta \mid \bigcirc\beta \in \Phi\}$. Given an elementary strict-future formula $\eta = \ddot{\bigvee}_{i=1}^{n} \bigwedge_{j=1}^{m} \bigcirc\beta_{i,j}$, the formula $\eta^{\downarrow}$ is defined to be $\ddot{\bigvee}_{i=1}^{n} \bigwedge_{j=1}^{m} \beta_{i,j}$.*

5.15 Example. *Consider the strict-future formula $\eta = \bigcirc a \,\ddot{\vee}\, \bigcirc\Diamond_{[1,1]}a \,\ddot{\vee}\, (\bigcirc b \wedge \bigcirc\Box_{[1,2]}b)$, then $\eta^{\downarrow} = a \ddot{\vee}\, \Diamond_{[1,1]}a \ddot{\vee} (b \wedge \Box_{[1,2]}b)$.*

Finally, Next-state Rule (Figure 5.4) is applied whenever the target set of formulas is elementary and, consequently, no saturation rules can be applied. This rule allows us to jump from one state to the next one.

$$(\bigcirc) \quad \frac{\Phi, \eta, \bigcirc\Box\psi}{\eta^{\downarrow}, \Box\psi} \quad \text{if } \Phi \cup \{\eta\} \text{ is elementary and } \eta \text{ is strict-future.}$$

Figure 5.4: Next-state Rule

Note that, if there is not an strict-future formula η, the successor of the above rule $(\bigcirc)$ is just $\Box\psi$.

5.5 A Tableau Algorithm for Realizability

In this section we present a non-deterministic algorithm (see Algorithm 2) for deciding whether a given safety specification is realizable or not. Algorithm 2 constructs a completed tableau that analyzes the minimal $\mathcal{X}$-coverings produced by the moves of the input safety specification TNF at the successive states of the game. For deciding realizability of a safety specification $\varphi = \alpha \wedge \Box\psi$, the initial call $\mathtt{Tab}(\varphi)$ is really $\mathtt{Tab}(\{\alpha\} \cup \{\Box\psi\})$.

Algorithm 2 can be seen as a safety game where Eve strategy is represented by $\chi = \Box\psi$ (line 3) and Sally strategy by $\chi = \bigcirc\Box\psi$ (line 24). As we mentioned before, tableau nodes consist of two types of successors, AND-successors and OR-successors. While OR-successors are generated by Sally with saturation rules (line 27) and by Eve with the selection of a minimal covering (line 11), AND-successors are only generated by Eve (line 17) with the moves of a specific minimal covering. The result is returned in the boolean variable is_open. If is_open is *False*, Eve wins, whereas if Sally wins, is_open is *True*.

5.20 Definition (open/closed branch)**.** *A branch b of a tableau is a finite sequence of nodes $n_0, \ldots, n_k$ such that n_0 is the root and $(n_i, n_{i+1}) \in R$ for all $0 \leq i < k-1$.*

- *If n_k is a successful leaf, we say that b is a open branch.*
- *If n_k is a failure leaf, we say that b is a closed branch.*

Recursive calls (lines 7, 13, 19 and 23) and the notion of open and closed tableau, are related with AND-successors, for which we introduce the notion of bunch.

5.21 Definition. *Given a set of branches H of a completed tableau, we say that H is a* bunch *if and only if for every $b \in H$ and every AND-node $n \in b$, and every n' that is an $(\Box\&)$-successor of n, there is $b' \in H$ such that $n' \in b'$. A completed tableau is* open *if and only if it contains at least one* bunch *such that all its branches are successful. Otherwise, when all possible bunches of a completed tableau contains a failure branch, the tableau is* closed.

Algorithm 2: Tab$(\Phi \cup \{\chi\})$ returns *is_open*: Boolean

1 **if** Φ *is inconsistent* **then**
2   *is_open* $:=$ *False*
3 **else if** $\chi = \Box\psi$ **then**
4   **if** $\Phi_0 \lessdot \Phi$ *for some* Φ_0 *in the branch of* Φ **then**
5   *is_open* $:=$ *True*
6   **else if** TNF$(\Phi \wedge \psi)$ *is not an* $\mathcal{X}$*-covering* **then**
7         $is\_open :=$ Tab$(\{$*False*$, \Box\psi\})$;
8   **else if** TNF$(\Phi \wedge \psi)$ *is a non-minimal* $\mathcal{X}$*-covering* **then**
9   Let $J_1, \ldots, J_m$ be all the minimal $\mathcal{X}$-coverings of TNF$(\Phi \wedge \psi)$;
10         $i := 0$; *is_open* $:=$ *False* ;
11   **while** $\neg$*is_open* $\wedge\ i < m$ **do**
12             $i := i + 1$;
13   *is_open* $:=$ Tab$(J_i \cup \{\Box\psi\})$;
14   **end**
15   **else**   // TNF$(\Phi \wedge \psi) = \bigvee_{i=1}^{n} \pi_i$ is a minimal $\mathcal{X}$-covering
16         $i := 0$; *is_open* $:=$ *True* ;
17   **while** *is_open* $\wedge\ i < n$ **do**
18             $i := i + 1$;
19   *is_open* $:=$ Tab$(\{\pi_i, \bigcirc\Box\psi\})$;
20   **end**
21   **end**
22 **else if** $\Phi = \Lambda \cup \{\eta\}$ *is elementary* (η *is strict-future*) **then**
23   *is_open* $:=$ Tab$(\{\eta^{\downarrow}, \Box\psi\})$;
24 **else**
25     $\rho :=$ *select_saturation_rule*(Φ);
26   Let $1 \leq k \leq 2$ and $\Phi_1, \ldots, \Phi_k$ the set of all ρ-children;
27   *is_open* $:=$ Tab$(\Phi_1 \cup \{\bigcirc\Box\psi\})$;
28   **if** $k = 2\ \wedge \neg$*is_open* **then** *is_open* $:=$ Tab$(\Phi_2 \cup \{\bigcirc\Box\psi\})$;
29 **end**

Algorithm 2 continuous looks for bunches of successful branches as follows:

- First, according to rule $(\Box\|)$, a recursive call is invoke for each minimal $\mathcal{X}$-covering. If any of these calls return *is_open* $:= True$, as node successors are OR-successors, the iteration is finished.

- Next, the construction of the tableau for a specific minimal covering J_k, by rule $(\Box\&)$ and according to lines from 15 to 20, produces a recursive call for each move π_i in

J_k. In addition, $(\Box\&)$ rule generates AND-successors, therefore, all moves π_i should return *is_open* $:= True$ to obtain truth for J_k.

- Then, lines 2 and 5 represent two types of terminal nodes which do not produce any recursive calls because no rules can be applied. Note that line 7 produces a recursive call that immediately returns failure.
- Finally, line 22 and 23 perform the application of $(\bigcirc)$ to change to the next state, and lines from 24 to 29 the application of saturation rules.

5.6 Examples

In this section, we present some representative examples that illustrate how our tableau method works.

5.16 Example. *Given $\Box(\bigcirc p_e \leftrightarrow \bigcirc s)$ safety specification, the following figure shows an open tableau construction.*

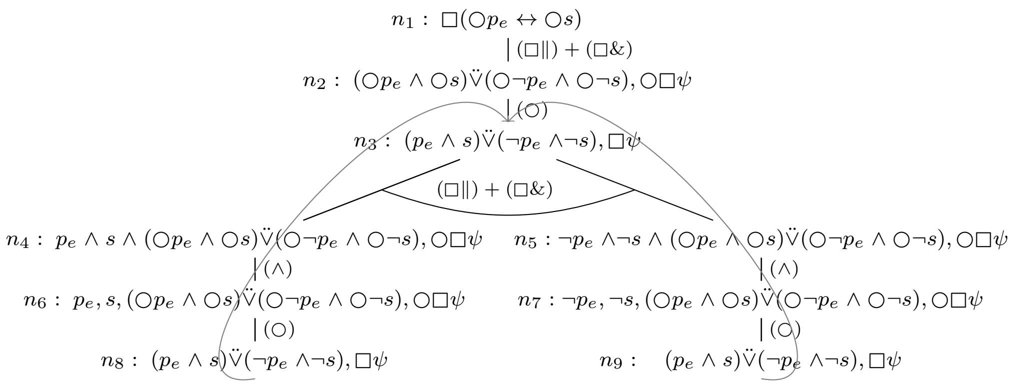


Figure 5.5: Open tableau for $\Box(\bigcirc p_e \leftrightarrow \bigcirc s)$.

First, we calculate the equivalent TNF of the safety specification that generates one minimal coverings and, as there are no environment variables taking part in the current state, only a single AND-successors is generated, n_2.

$$TNF(\bigcirc p_e \leftrightarrow \bigcirc s) \equiv (\bigcirc p_e \wedge \bigcirc s)\ddot{\vee}(\bigcirc\neg p_e \wedge \bigcirc\neg s)$$

Then, no more rules can be applied and we change to the next state. Now, there is an environment variable taking part in the present, p_e, so after calculating the TNF$((p_e \wedge s)\ddot{\vee}(\neg p_e \wedge\neg s) \wedge \psi)$, we generate the one and the only following minimal covering:

$$C_1 : (p_e \wedge s \wedge (\bigcirc p_e \wedge \bigcirc s)\ddot{\vee}(\bigcirc\neg p_e \wedge \bigcirc\neg s))\ \&$$
$$(\neg p_e \wedge\neg s \wedge (\bigcirc p_e \wedge \bigcirc s)\ddot{\vee}(\bigcirc\neg p_e \wedge \bigcirc\neg s))$$

Afterwards, $p_e \wedge s \wedge (\bigcirc p_e \wedge \bigcirc s)\ddot{\vee}(\bigcirc\neg p_e \wedge \bigcirc\neg s)$ move change to the next state resulting in an open branch between n_8 and n_3. Finally, n_5 node will follow the same strategy as n_4 due to the fact that both have the same strict-future and, therefore, same n_4 strategy will open n_5.

5.17 Example. *Given* $\Box(p_e \wedge s \wedge \bigcirc s) \vee (\neg s \wedge \bigcirc^2 s) \vee (\neg p_e \wedge \neg s \wedge \bigcirc^3 s)$ *safety specification we conclude by the construction of the following closed tableau that is not a realizable specification.*

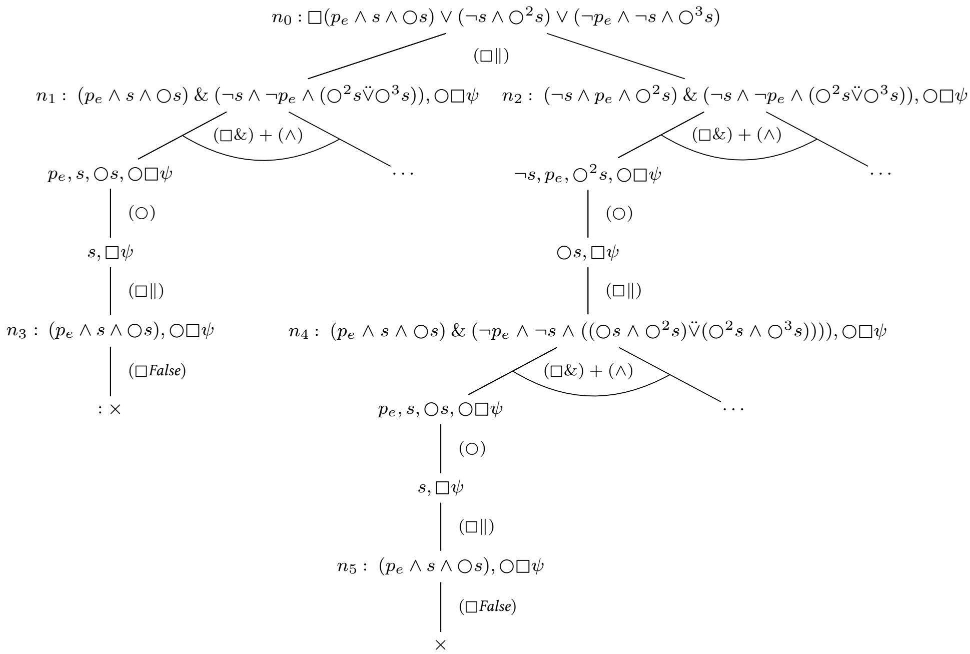


Figure 5.6: Closed tableau for $\Box(p_e \wedge s \wedge \bigcirc s) \vee (\neg s \wedge \bigcirc^2 s) \vee (\neg p_e \wedge \neg s \wedge \bigcirc^3 s)$

The construction of the tableau starts by calculating TNF$((p_e \wedge s \wedge \bigcirc s) \vee (\neg s \wedge \bigcirc^2 s) \vee (\neg p_e \wedge \neg s \wedge \bigcirc^3 s) \wedge \psi)$ *and resulting in the following two minimal coverings:*

- $n_1 : (p_e \wedge s \wedge \bigcirc s) \,\&\, (\neg s \wedge \neg p_e \wedge (\bigcirc^2 s \vee \bigcirc^3 s)))$
- $n_2 : (\neg s \wedge p_e \wedge \bigcirc^2 s) \,\&\, (\neg s \wedge \neg p_e \wedge (\bigcirc^2 s \vee \bigcirc^3 s)$

Both minimal covering failed turning into closed branches due to the fact that TNF$(s \wedge \psi)$ $\equiv (p_e \wedge s \wedge \bigcirc s)$ *is not a* $\mathcal{X}$*-covering. In addition, node* n_3 *causes to fail node* n_1 *because is AND-successor, in the same way that node* n_5 *provoke the failure of* n_4 *and* n_2 *nodes. Consequently, as all the minimal coverings of the safety specification fails, its a closed tableau an a not realizable specification. Referring to the safety games, closed branches represent a winning strategy for the environment.*

5.18 Example. *Let* $\square(p_e \wedge s \wedge \square_{[1,10]}t \wedge \bigcirc s) \vee (\neg p_e \wedge s \wedge \Diamond_{[1,10]}t \wedge \bigcirc^2 s) \vee (\neg s \wedge \square_{[1,10]}\neg s)$ *be the safety specification, the following tableau shows a winning strategy for the system and, therefore, an open tableau.*

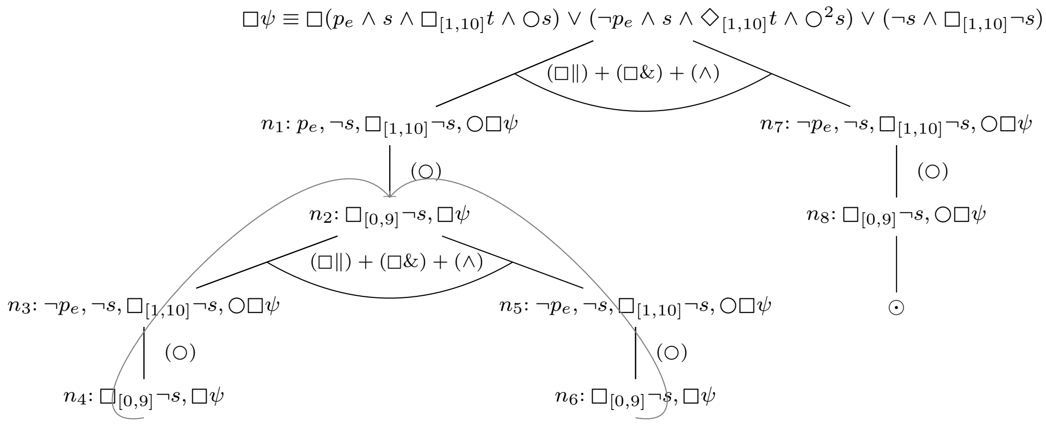


Figure 5.7: Open tableau for $\square((p_e \wedge s \wedge \square_{[1,10]}t \wedge \bigcirc s) \vee (\neg p_e \wedge s \wedge \Diamond_{[1,10]}t \wedge \bigcirc^2 s) \vee (\neg s \wedge \square_{[1,10]}\neg s))$.

Initially, we calculate the TNF(ψ) that results in four minimal covering:

$$C_1 \ : (p_e \wedge \neg s \wedge \square_{[1,10]}\neg s) \ \& \ (\neg p_e \wedge \neg s \wedge \square_{[1,10]}\neg s)$$

$$C_2 \ : (p_e \wedge \neg s \wedge \square_{[1,10]}\neg s) \ \& \ (\neg p_e \wedge s \wedge \square_{[1,10]}t \wedge \bigcirc^2 s)$$

$$C_3 \ : (p_e \wedge s \wedge \square_{[1,10]}t \wedge \bigcirc s) \ \& \ (\neg p_e \wedge \neg s \wedge \square_{[1,10]}\neg s)$$

$$C_4 \ : (p_e \wedge s \wedge \square_{[1,10]}t \wedge \bigcirc s) \ \& \ (\neg p_e \wedge s \wedge \square_{[1,10]}t \wedge \bigcirc^2 s)$$

As the minimal covering C_1 has less conjunctions as well as the same futures in both environment valuation moves, we select it. Afterwards, we jump to the next state obtaining a single minimal covering:

$$(\neg p_e \wedge \neg s \wedge \square_{[1,10]}\neg s) \ \& \ (p_e \wedge \neg s \wedge \square_{[1,10]}\neg s)$$

Both moves generate an open branch when moving to the next state. At this point, n_2 has been successful after detecting cycles between $n_2 - n_4$ and $n_2 - n_6$ but, we still have to check node n_7. However, n_1 and n_7 have the same strict-future so both of them will generate the same tableau branches and n_7 will arrives to the success $\square_{[0,9]}\neg s$ node at n_8 that will be will automatically open, ensuring a winning strategy for the system and consequently, an open tableau and a realizable specification.

5.19 Example. *Let* $\Box(p_e \wedge s \wedge \Box_{[1,1000]}t \wedge \bigcirc s) \vee (\neg p_e \wedge s \wedge \Diamond_{[1,1000]}t \wedge \bigcirc^2 s) \vee (\neg s \wedge \Box_{[1,1000]}\neg s)$ *be the safety formula, similar to the previous example but increasing the superior limit of the bounded always interval to 1000. Moreover, the following figure shows a winning strategy for the system and, therefore, an open tableau.*

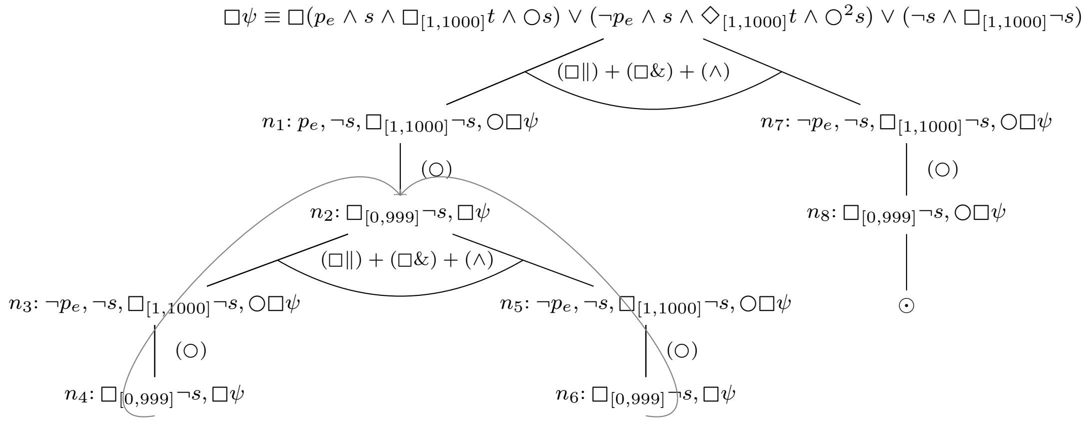


Figure 5.8: Open tableau for $\Box((p_e \wedge s \wedge \Box_{[1,1000]}t \wedge \bigcirc s) \vee (\neg p_e \wedge s \wedge \Diamond_{[1,1000]}t \wedge \bigcirc^2 s) \vee (\neg s \wedge \Box_{[1,1000]}\neg s))$.

Comparing with the previous Example 5.7 , increasing superior limit of the bounded always interval does not affect to tableau construction. Accordingly, we can ensure that if we have a winning strategy for a specification and we increase superior limit of the bounded always interval the resulting tableau will be the same in terms of size.

5.20 Example. *Consider the safety specification,* $a \wedge \Box((a \to c) \wedge (p_e \to \Diamond_{[0,100]}\neg c) \wedge (\neg p_e \to \Diamond_{[0,100]}a))$.

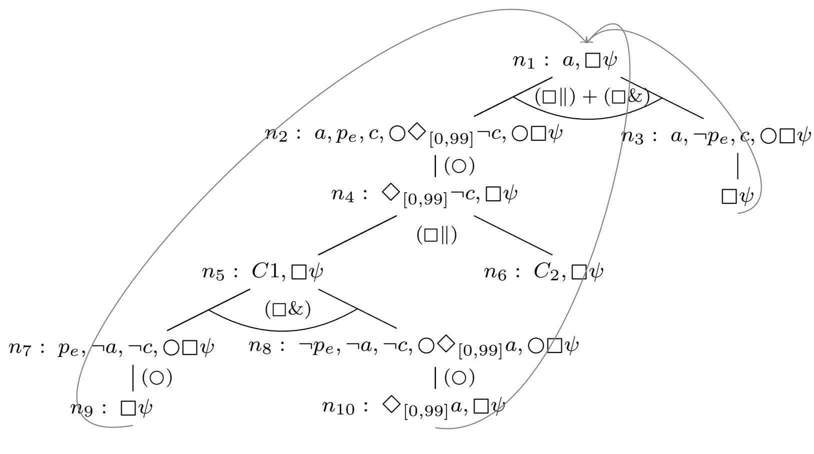


Figure 5.9: Open Tableau for $a \wedge \Box((a \to c) \wedge (p_e \to \Diamond_{[0,100]}\neg c) \wedge (\neg p_e \to \Diamond_{[0,100]}a))$.

Firstly, we calculate the equivalent TNF of the safety specification TNF$(a \wedge \psi)$ *obtaining a single minimal covering* $(p_e \wedge a \wedge c \wedge \bigcirc\Diamond_{[0,99]}\neg c)$ & $(\neg p_e \wedge a \wedge c)$. *Then, we start developing* $(p_e \wedge a \wedge c \wedge \bigcirc\Diamond_{[0,99]}\neg c)$ *move due to the fact that is stronger than* $(\neg p_e \wedge a \wedge c)$ *and since they are AND-successor, we are interested in fulfill the strongest* *moves. Once we jump to the next state, we need to calculate the* TNF($\Diamond_{[0,99]}\neg c \wedge \psi$) *and the corresponding weakest minimal covering,* C_1 *and* C_2*.*

$$\begin{aligned}&\mathrm{TNF}(\Diamond_{[0,99]}\neg c \wedge \psi) \equiv (p_e \wedge c \wedge \bigcirc\Diamond_{[0,98]}\neg c) \vee (p_e \wedge \neg a \wedge \neg c) \vee \\ &(p_e \wedge \neg a \wedge c \wedge \bigcirc\Diamond_{[0,98]}\neg c) \vee (\neg p_e \wedge c \wedge a \wedge \bigcirc\Diamond_{[0,98]}\neg c) \vee \\ &(\neg p_e \wedge c \wedge \neg a \wedge \bigcirc\Diamond_{[0,99]} a \wedge \bigcirc\Diamond_{[0,98]}\neg c) \vee (\neg p_e \wedge \neg a \wedge \neg c \wedge \bigcirc\Diamond_{[0,99]} a)\end{aligned}$$

$$\begin{aligned}C1 &= (p_e \wedge \neg a \wedge \neg c) \vee (\neg p_e \wedge \neg a \wedge \neg c \wedge \bigcirc\Diamond_{[0,99]} a) \\ C2 &= (p_e \wedge \neg a \wedge \neg c) \vee (\neg p_e \wedge c \wedge a \wedge \bigcirc\Diamond_{[0,98]}\neg c)\end{aligned}$$

Afterwards, we develop C_1 *minimal covering that became a successful node by generating an open branch with both* n_9 *and* n_{10} *nodes. At this point, the system already has a winning strategy for* p_e *but it need also a winning strategy for* $\neg p_e$*. Nevertheless,* n_3 *strict-future formula is True so when we change to the next state we will get the weakest possible node,* $\Box\psi$*, that can generate an open branch with any other higher node, in this case with* n_1 *node.*

5.21 Example. *Let* $a \wedge \Box\psi$ *be a safety specification where* $\psi = (a \rightarrow c) \wedge (p_e \rightarrow \bigcirc a) \wedge (\neg p_e \rightarrow \Box_{[2,10]}\neg c)$*. Figure 5.10 is a closed tableau that proves that* $a \wedge \Box\psi$ *is unrealizable.*

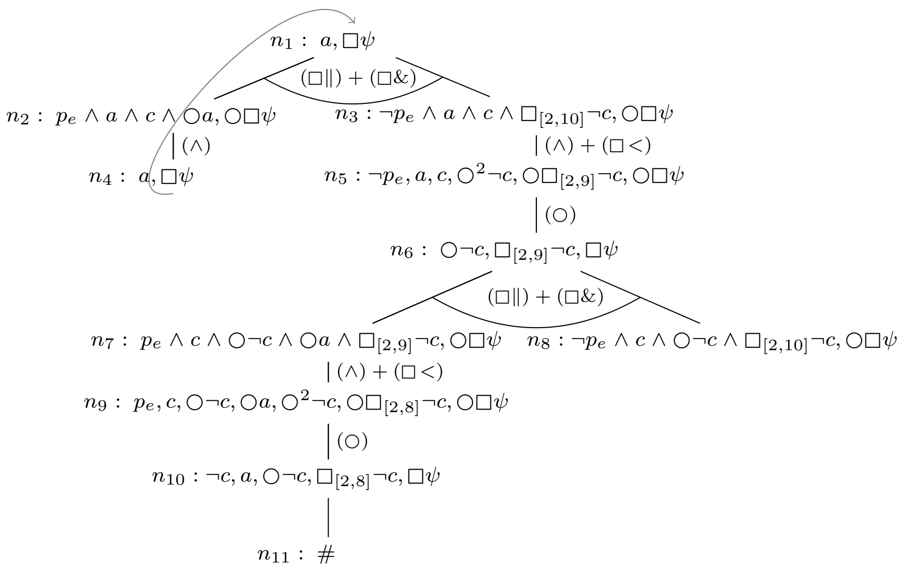


Figure 5.10: Closed tableau for $a \wedge \Box((a \rightarrow c) \wedge (p_e \rightarrow \bigcirc a) \wedge (\neg p_e \rightarrow \Box_{[2,10]}\neg c))$.

To start the tableau construction, we have that:

$$\mathrm{TNF}(a \wedge \psi) = (p_e \wedge a \wedge c \wedge \bigcirc a) \vee (\neg p_e \wedge a \wedge c \wedge \Box_{[2,10]}\neg c).$$

The realizability result depends on the success of n_2 *and* n_3 *AND-nodes. Once the success of node* n_2 *is ensured, the tableau goes on with the expansion of node* n_3*. At node* n_6*, we have that* TNF($\bigcirc\neg c \wedge \Box_{[2,9]}\neg c \wedge \Box\psi$) $=$

$$(p_e \wedge c \wedge \bigcirc\neg c \wedge \bigcirc a \wedge \square_{[2,9]}\neg c) \vee (p_e \wedge\neg a \wedge \bigcirc\neg c \wedge \bigcirc a \wedge \square_{[2,9]}\neg c) \vee$$
$$(\neg p_e \wedge c \wedge \bigcirc\neg c \wedge \square_{[2,10]}\neg c) \vee (\neg p_e \wedge\neg a \wedge \bigcirc\neg c \wedge \square_{[2,10]}\neg c)$$

Hence there are 4 possible minimal $\mathcal{X}$-coverings and it is enough to choose any of them to decide that the tableau is closed or open because they have the same strict-future formula. Therefore, the tableau goes on with the AND-nodes n_7 and n_8, which correspond to the following minimal $\mathcal{X}$-covering.

$$(p_e \wedge c \wedge \bigcirc\neg c \wedge \bigcirc a \wedge \bigcirc^2\neg c \wedge \bigcirc\square_{[2,8]}\neg c)\&$$
$$(\neg p_e \wedge c \wedge \bigcirc\neg c \wedge \bigcirc^2\neg c \wedge \bigcirc\square_{[2,9]}\neg c)$$

As the TNF *at node n_{10} is $False$, this node is a failure leaf. This fact completes the tableau, since n_7 and n_8 are AND-siblings.*

5.22 Example. *The next page Figure 5.11 represents an open tableau for the Running Example 4.5, whose construction starts with C_1, the weakest minimal $\mathcal{X}$-covering in* TNF(ψ)*, composed by the labelled nodes n_2, n_3, n_4 and n_5.*

At node n_8, the weakest minimal $\mathcal{X}$-covering in TNF$(\diamond_{[0,2]}\, g_1 \wedge \psi)$ *is C_2, which has the following four moves:*

$$m_1 : (r_1 \wedge\neg r_2 \wedge g_1 \wedge\neg g_2)$$
$$m_2 : (\neg r_1 \wedge r_2 \wedge g_1 \wedge\neg g_2 \wedge \bigcirc\diamond_{[0,2]}\, g_2)$$
$$m_3 : (r_1 \wedge r_2 \wedge g_1 \wedge\neg g_2 \wedge \bigcirc\diamond_{[0,2]}\, g_2)$$
$$m_4 : (\neg r_1 \wedge\neg r_2 \wedge g_1 \wedge\neg g_2 \wedge \bigcirc\neg g_2)$$

Note that, for simplicity, we group m_2 and m_3 in the same node that omits the value of r_1, which is the only difference between both moves. At node n_{10}, C_3, is the weakest minimal $\mathcal{X}$-covering in TNF$(\diamond_{[0,2]}\, g_2 \wedge \psi)$*. It has four moves m'_1, m'_2, m'_3, m'_4 but m'_2 and m'_3 has been grouped. And similarly, at node n_{13}, where* TNF$(\neg g_2 \wedge \psi)$ *provides C_4, with the moves $m''_1, m''_2, m''_3, m''_4$. Note that nodes m'_4, m_4 and n_5 share the same strict-future formula. Hence, to save space, we do not depict the expansion of nodes m_4 and n_5 since it repeats the tableau behind node m'_4. All in all, the completed tableau for the input specification is open.*

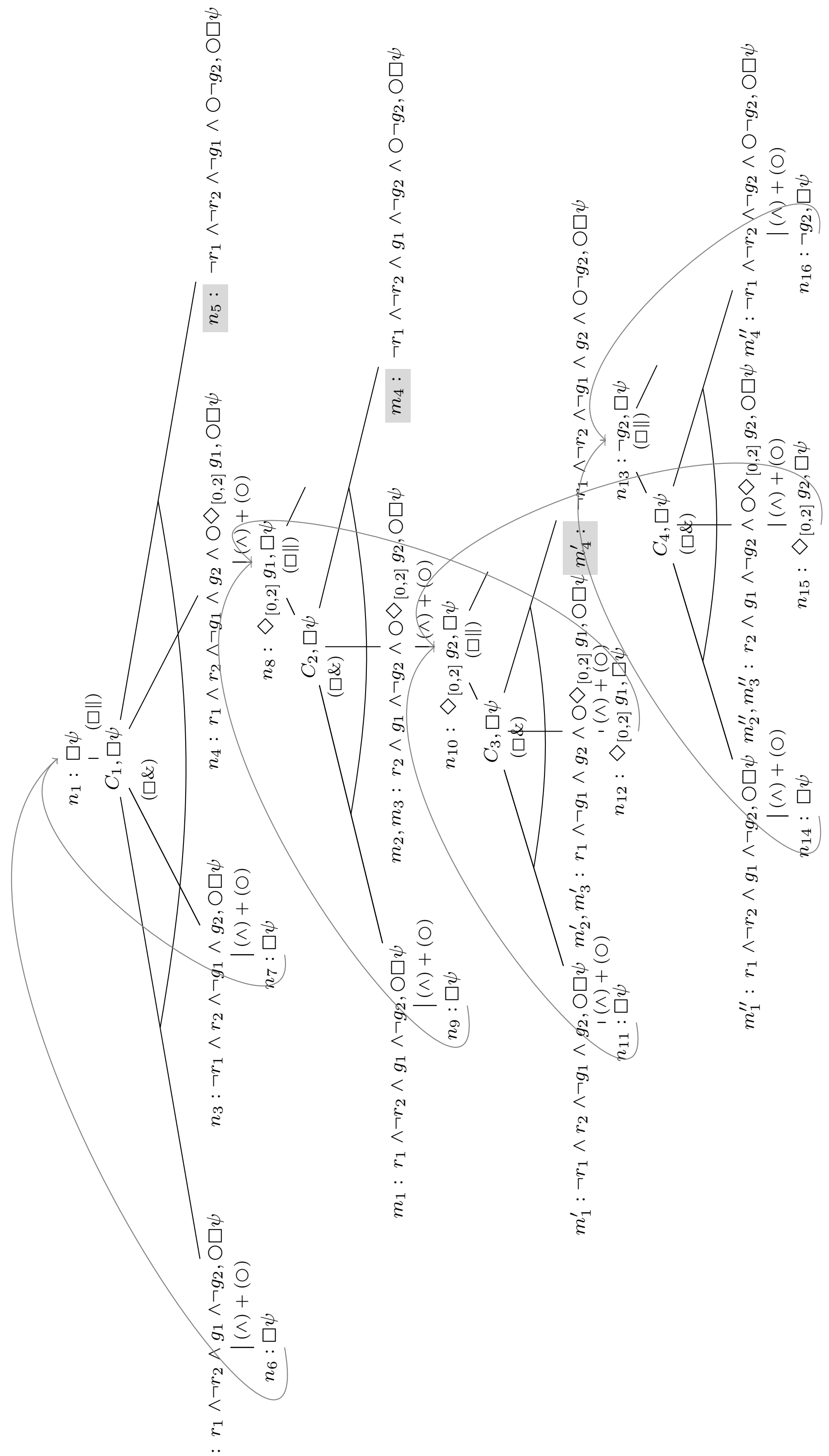


Figure 5.11: Open tableau for $\Box\big((r_1 \rightarrow \Diamond_{[0,3]}\, g_1) \wedge (r_2 \rightarrow \Diamond_{[0,3]}\, g_2) \wedge \neg(g_1 \wedge g_2) \wedge ((\neg r_1 \wedge \neg r_2) \rightarrow \bigcirc\neg g_2)\big)$.

CHAPTER 6

Implementation

This chapter explains in a simplified and superficial way the structure, development and implementation of the application prototype.

6.1 Development tools

Application has been developed using python 3 interpreted high-level programming language. In addition, Git has been used for application version control, GitHub for Git repository hosting service and Visual Studio Code as principal code editor.

The application is hosted in a public repository on GitHub where the implementation can be better understood. Moreover, we will be constantly updating and improving the prototype, so it may not match exactly what we will explain below with the latest version.

https://github.com/AnderEhu/Realizability-Tableau

Benchmarks will be tested on a laptop with the following specifications:

- Operative system: Linux Mint 20.1 Cinnamon 4.8.6
- CPU: Intel© Core™ i5-6300HQ CPU @ 2.30GHz $\times$ 4
- GPU: NVIDIA GM107M [GeForce GTX 950M]
- RAM: 8 GiB
- SSD: Samsung SSD 970 EVO 250GB

6.2 How to run the prototype?

The prototype can be executed using the following command:

```
python3 run_tableau.py benchmark.txt
```

As input parameter you have to specify the benchmark file path you want to run. In addition, benchmark format is divided into three parts; initial formula, safety formula and environment global constraints [1]. However, if there is neither initial formula nor environment constraints it must be indicated by means of True formula.

```
Initial Formula
(temporal formula)

Safety Formula
(temporal formula)
(temporal formula)
(temporal formula)
...

Environment Global Constraints
(temporal formula)
...
```

Referring to Example 4.5, benchmark input file for the application is:

```
Initial Formula
True

Safety Formula
r1_e -> F[0,3]g1
r2_e -> F[0,3]g2
-(g1 & g2)
(-r1_e & -r2_e) -> X-g2

Environment Global Constraints
True
```

The previous benchmark is named as "benchmark6.txt" and is located in "benchmarks/Overleaf/realizable/", so for the execution of the prototype we will run:

```
python3 run_tableau.py benchmarks/Overleaf/realizable/benchmark6.txt
```

[1] It is out of the scope of this project, always represent it with True

6.3 Prototype structure

The project consists of three packages TemporalFormula, TNF and Tableau together with the package for the use of Bica [31]. In addition, most of the functions are documented, and tested through unit tests of the pytest package [32].

```
Realizability Tableaux/
├── benchmarks/
│   ├── Overleaf/...
│   └── Automatic/...
├── Solver/
│   ├── bica.py
│   └── circuit.py
├── Tableau/
│   ├── src/
│   │   ├── automatic_benchmark_generator.py
│   │   ├── minimal_covering.py
│   │   ├── tableau_node.py
│   │   ├── tableau_rules.py
│   │   └── tableau.py
│   └── test/
│       ├── test_tableau_automatic.py
│       └── test_tableau_overleaf.py
├── TemporalFormula/
│   ├── src/
│   │   └── temporal_formula.py
│   └── test/
│       └── test_temporal_formula.py
├── TNF/
│   ├── src/
│   │   ├── inconsistencies.py
│   │   ├── separated_formula.py
│   │   ├── subsumptions.py
│   │   └── tnf.py
│   └── test/
│       ├── test_inconsistencies.py
│       ├── test_separated_formula.py
│       ├── test_subsumptions.py
│       └── test_tnf.py
├── tools.py
└── run_tableau.py
```

6.4 Temporal Formulas

6.4.1 Syntax

For an efficient manipulation of the temporal formulas we represented it as a list of lists. We distinguish two types of operators, the binary operators and the unary operators.

Unary operator syntax	Binary operator syntax
$\neg \equiv$!, -, $\sim$	$\wedge \equiv$ && or &
$\bigcirc^i \equiv \mathrm{X}[i]$ or $\mathrm{X}_0\mathrm{X}_1 ... \mathrm{X}_{i-1}$	$\vee \equiv$ || or |
$\square_{[n,m]} \equiv \mathrm{G}[n,m]$	
$\diamond_{[n,m]} \equiv \mathrm{F}[n,m]$	

Table 6.1: Prototype operator syntax

On the one hand, binary operators operate on two formulas, while unary operators operate on one. Therefore, we represent formulas with binary operator as a list of length 3 where the first element correspond to the binary operator and the second and third elements to the temporal formulas. Whereas the formulas with unary operator are represented as a list of length 2 where the first element match with the unary operator and the second element with the temporal formula. Moreover, we represent system variables as a simple string and environment variable as a string with "_e" at the end

6.1 Example. *Let* `((X[2]s | F[4,7]s) & (-p_e & G[1,10]c))` *be the temporal formula string so the list of lists representation is as follows:*

`["&",["|",["X[2]","s"],["F[4,7]","s"]],["&", ["-","p_e"],["G[1,10]","c"]]]`

6.4.2 Parsing expression grammar

To parse a temporal formula as string to a list of lists we use parsimonious library [33], the fastest arbitrary-lookahead parser.

First, we apply the following grammar to the input temporal formula string:

```
Biconditional = (Conditional "<-->" Biconditional) / Conditional
Conditional = (Disyunction "->" Conditional) / Disyunction
Disyunction = (Conjunction ("||" / "|") Disyunction) / Conjunction
Conjunction = (Literal ("&&" / "&") Conjunction) / Literal
Literal = (Atom) / ((Neg / Bounded_Eventually / Next / Bounded_Always ) Literal)
Atom = True / False / Var / Group
Group = "(" Biconditional ")"
Var = ~r"[a-zA-EH-WY-Z0-9][a-zA-Z0-9_]*"
Next = ~r"X[[0-9]+]" / "X"
Bounded_Eventually = ~r"F[[0-9]+,[0-9]+]"
Bounded_Always = ~r"G[[0-9]+,[0-9]+]"
Neg = "!" / "-" / "~"
True = "TRUE" / "True"
False = "FALSE" / "False"
```

Afterwards, we get an Abstract Syntax Tree that by means of walking through it (with the function `visit` of nodes.NodeVisitor subclass), we create the list of lists representation. Finally, we calculate the equivalent NNF applying the rules seen in Section 2.1.3. Note that optionally futures can be split into strict-future formulas, for example, `[“G[0,4]”, “s”]` splits into `[“&”, “s”, [“G[1,4]”, “s”]]` and `[“F[2,4]”, “s”]` splits into `[“|”, [[“X[2]”, “s”], [“G[3,4]”, “s”]]`

6.5 DNF and Separated Formulas

We calculate with Bica solver, from a temporal formula in an arbitrary form, its equivalent DNF formula as list of separated formulas. Separated formulas are implemented as a dictionary with three keys:

- ’X’: correspond to the set of environment variables
- ’Y’: correspond to the set of system variables
- ’Futures’: correspond to the list of strict-futures sets.

Remark that lists refers to OR-formulas and sets to AND-formulas. For example, `[{’X[1]a’, ’X[1]b’, ’X[1]c’}, {’X[2]a’, ’X[2]b’, ’X[2]c’ }]` is equivalent to `’((X[1]a & X[1]b & X[1]c) | (X[2]a & X[2]b & X[2]c))’`

6.2 Example. *Given a DNF* ≡ `[{’p_e’, ’a’, X[1]a’, ’X[1]b’, ’X[1]c’}, {’p_e’, ’-a’, ’X[2]a’, ’X[2]b’, ’X[2]c’ }]`, *its list of separated formulas representation is as follows:*

- *For* `{’p_e’, ’a’, X[1]a’, ’X[1]b’, ’X[1]c’}` ≡
  `{’X’: {’p_e’}, ’Y’: {’a’}, ’Futures’: [{’X[1]a’, ’X[1]b’, ’X[1]c’}]}`
- *For* `{’p_e’, ’-a’, X[2]a’, ’X[2]b’, ’X[2]c’}` ≡
  `{’X’: {’p_e’}, ’Y’: {’-a’}, ’Futures’: [{’X[2]a’, ’X[2]b’, ’X[2]c’}]}`

6.6 TNF

6.6.1 Data Structure

TNF formula data structure is implemented as a dictionary where the key corresponds to a specific environment valuation and its value is the extension of moves for that environment valuation.

6.3 Example. *Let be the TNF result of Example 5.13:*

$$\mathcal{T} = \{(p_e \wedge a_2 \wedge\neg a_1 \wedge (\bigcirc b \vee \bigcirc^2 c), (\neg p_e \wedge\neg a_1 \wedge (\bigcirc^2 c \vee \bigcirc\neg b))\}$$

we will represent it as follows:

$\mathcal{T}$ ≡ `{’p_e’: [[’a_2’,’-a_1’, [’X[1]b’,’X[2]c’]]], ’-p_e’: [[’-a_1’, [’X[1]b’,’X[2]c’]]]}`

6.6.2 Algorithm

Our TNF algorithm implementation take as input a DNF represented as a list of separated formulas. The equivalent TNF is calculate by joining together all the TNFs calculated in each environment extension of moves.

6.1 Proposition. *Given a DNF formula φ, TNF(φ) is equivalent to the conjunction of TNFs calculated from the extension of moves for each environment valuation.*

It should be pointed out that compatible formulas and join operator has been implemented together for efficiency, i.e. we apply the join operator for each compatible formula as opposed to selecting all compatible formulas and then applying join operator.

We enumerate each formulas with an integer corresponding to its list position and then, we use a pointer 'i' that will go through the formulas adding and removing elements from the following three different stacks:

- literals_s represent a stack of literal sets. Moreover, from now on we will refer to the top of the stack as literals_c.
- futures_s represent a stack of the list of futures set corresponding to Futures_stack. Moreover, from now on we will refer to the top of the stack as futures_c.
- index_s represent a stack that save the list position of the formula that append a new value to the Literals_stack. Moreover, from now on we will refer to the top of the stack as index_c.

The first part of the algorithm is the initialization of the variables. In case the length of formulas is greater than two, we start adding the information of the first formula to the stacks and setting the pointer i to the second formula.

```
i = 1
index_s.append(0)
literals_s.append(formulas[0]['Y'])
futures_s.append(formulas[0]['Futures'])
```

Next, until $i > 0$, in each loop is modified stacks according to three different cases. Note that we denoted the literals and futures of the formula at i position as literal_i and futures_i and $\text{literals}_i \cup \text{literals}_c$ as union_literals.

```
    if inconsistent(union_literals) or union_literals in Skip:
        i++
    if literals_i ≠ literals_c:
        union_futures = union(futures_i, futures_c)
        index_s.append(i)
        literals_s.append(union_literals)
        futures_s.append(union_futures)
        i++
    else:
        append_futures(futures_i, futures_c)
        i++
```

Then, as we increment i, whether all the formulas have already been traversed, we append to TNF the result of the list composed of literal$_c$ and futures$_c$. In addition, we add literal$_c$ set to skip list in order to avoid adding redundant formulas and then, we select a new possible value of i by adding 1 to the value at the top of the index$_s$.

```
if i == length(formulas):
    new_move = [literals_c, futures_c]
    append_tnf(TNF, new_move)
    append_skip(Skip, literals_c)

    i = index_s + 1

    literals_s.pop()
    futures_s.pop()
    index_s.pop()

    i = get_valid_i(i)
```

Finally, after removing the top element of the stacks, we need to ensure that the new value for i points to an index of the formulas. However if there is no formula to deal with, i will be equal to -1 and the algorithm will end returning the current solution of the TNF. The following function get_valid_i validates the pointer i as follows:

```
def get_valid_i(i):
```

Whether i points outside the formulas, i == length(formulas), then i is not a valid index, so there are two cases to deal with. First one is when there is no element to pop from index$_s$, i.e. all formulas have been visited and the algorithm must be end. And the other one, when index$_s$ is not empty, in this case we select as another possible i the value at the top of index$_s$ plus 1, we pop the top of index$_s$, literals$_s$ and futures$_s$ and recursively we call with the new possible value of i to validate it.

```
    if i == length(formulas) then:
        if not index_s then:
            return -1

        else:
            i = Index_c + 1
            literals_s.pop()
            futures_s.pop()
            index_s.pop()
            return get_valid_i(i)
```

If i points to a valid index but its corresponding formula system variables valuations are in the skip list then we increment i in 1. On the other hand, whether index$_s$ is empty then we push i to index$_s$, i formula literals to literals$_s$ and i formula futures to futures$_s$. In both cases we call recursively to validate the new possible i.

```
else:
    if formulas[i]['Y'] in Skip then:
        i++
        return get_valid_i(i)
    elif is_empty(index_s) then:
        index_s.append(i)
        literals_s.append(formulas[i]['Y'])
        futures_s.append(formulas[i]['Futures']
        return get_valid_i(i)
```

If none of the above cases are fulfilled, the pointer i will be a valid index of formulas.

```
    else:
        return i
```

6.6.3 Verification

To verify that the DNF and the resulting TNF are equivalent in terms of environment valuations and strict-futures we need to remove the set of system variables from the moves of both and then apply the logical equivalence seen in Definition 2.3. Consequently, depending on whether our aim is to verify the equivalence of TNF with DNF or not, append_futures and append_tnf functions will change. Both the application of weaker moves and the subsumption of futures will be restricted in the verification in order to preserve it.

On the one hand, to maintain the verification, above-mentioned functions are simple whose unique purpose is to add an element to a list in order to preserve the equivalence.

```
def append_futures(list_futures, futures):
    if futures not in list_futures:
        union_futures.append(futures)
```

```
def append_tnf(tnf, new_move):
    tnf.append(new_move)
```

On the other hand, if equivalence checking is not required, append_futures will apply the order relation according to Definition 5.6 and append_tnf will pursuit weaker moves according to Definition 5.12 .

6.7 Minimal Covering

Calculating all the minimal coverings is computationally expensive because we need to apply the Cartesian product between the different moves of each environment valuations. For example, given 8 environment variables, we obtain 256 different environment valuations and if each one has only 2 moves, we will need to calculate more minimal covering than atoms on earth (2^{256}), impossible. Therefore, we arise to a problem due to the fact that finding weakest minimal coverings is fundamental for the good development of the tableau algorithm.

To solve this problem we will make two scoring system, one for each move and the other for the environment valuation. Given the list of futures set, we will score each move as follows:

- For each set of futures we will add 1000 to the score because the more sets of futures a move has the weaker it is likely to be.
- The size of each set of futures we will subtract a score equal to the cube of its length. We subtract score because sets represent AND-formulas and this makes the move stronger.
- For each set representing X[1]True we will add the maximum possible score because it is the weakest future.

Furthermore, the score of the environment valuations will be the sum of each scored moves.

6.4 Example. *Given the following TNF:*

```
𝒯 ≡ {
      'p_e': [
                 [{'a_2','-a_1'}, [{'X[1]b', 'X[2]s', 'X[4]s'},{'X[2]c', 'X[3]s'}]],
                 [{'-a_2','-a_1'}, [{'X[5]s', 'X[6]s'},{'X[2]b', 'X[3]b'}]]
              ]
      '-p_e': [
                 [{'-a_1'}, [{'X[1]True'}]]
              ]
    }
```

each move gets the following score:

$$\texttt{[\{'a\_2','-a\_1'\}, }\underbrace{\texttt{[}\underbrace{\texttt{\{'X[1]b', 'X[2]s', 'X[4]s'\}}}_{-2^3},\underbrace{\texttt{\{'X[2]c'\}}}_{-1^3}\texttt{]}}_{2*1000}\texttt{] = 1991}$$

$$\texttt{[\{'-a\_2','-a\_1'\}, }\underbrace{\texttt{[}\underbrace{\texttt{\{'X[5]s', 'X[6]s'\}}}_{-2^2},\underbrace{\texttt{\{'X[2]s', 'X[3]s'\}}}_{-2^2}\texttt{]}}_{2*1000}\texttt{] = 1992}$$

```
[{'-a_1'}, [{'X[1]True'\}}]] = 10^12
```

To calculate minimal covering we will use the Cartesian product of itertools package [34]. In addition, we will use the iterator returned by the function to calculate and obtain minimal coverings dynamically.

6.5 Example. *Given a TNF with one environment variable $\mathcal{X} = \{p_e\}$ and two moves for each environment valuation, $move_1$ and $move_2$ for p_e = True and $move_3$ and $move_4$ for p_e = False, Cartesian product function return the following minimal covering:*

$$[move_1, move_2] \times [move_3, move_4] = [\underbrace{(move_1, move_3)}_{M_1}, \underbrace{(move_1, move_4)}_{M_2}, \underbrace{(move_2, move_3)}_{M_3}, \underbrace{(move_2, move_4)}_{M_4}]$$

As you can see in the example above, minimal coverings are calculated on the basis of the order of the input. Therefore, whether we sort the moves of each environment valuation based on its score in a descending order and we order the input of the Cartesian function so that the first positions are the environment valuations with less score, then, we will obtain the weakest minimal covering dynamically in the first steps of the iterator.

6.6 Example. *Given Example 6.5 we order the Cartesian product input base on the following scores; $move_1$ = -100, $move_2$ = 700, $move_3$ = 800, $move_4$ = 1000 and obtaining $[move_2, move_1] \times [move_4, move_3]$ as input and the output result as:*

$$[\underbrace{(move_4, move_2)}_{M_4}, \underbrace{(move_4, move_1)}_{M_2}, \underbrace{(move_2, move_3)}_{M_3}, \underbrace{(move_2, move_4)}_{M_1}]$$

It is worth noting that if we did not order in Example 6.5 tableau would have started to develop the strongest minimal covering. Although in the previous examples we have calculated all the minimal covering to illustrate the improvement of sorting, but remark that we will only generate a new one when the current is unrealizable, that is, when minimal covering fails.

6.7 Example. *Given the TNF and the scores of Example 6.4 Cartesian product function input will be as follows:*

$$\texttt{[move}_2\texttt{, move}_1\texttt{]} \times \texttt{[move}_3\texttt{]}$$

where:

- *p_e $move_1$* = `['a_2','-a_1', ['X[1]b', 'X[2]s', 'X[4]s','X[2]c', 'X[3]s']]` *(1991 points)*
- *p_e $move_2$* = `['-a_2','-a_1', ['X[5]s', 'X[6]s','X[2]b', 'X[3]b']]` *(1992 points)*
- *-p_e $move_3$* = `['-a_1', ['X[1]True']]` *(10^{12} points)*

6.8 Tableau

6.8.1 Tableau Nodes

Tableau nodes (from now only nodes) are represented by a structure that contains the temporal formula associated to the node, the reference to the predecessor node and the depth of the tableau at which it is located. However, the safety formula is not included in the node structure, since it is the same for all nodes so we keep it frozen as an attribute of the tableau class.

In addition, the nodes contain the following functions:

- `implicate_a_failure_nodes`: if the current node is weaker than a previous node that has failed, automatically is a failure node (see Definition 5.1)
- `is_implicated_by_success_nodes`: if the current node is stronger than any predecessor node that has been successful, automatically is a successful node (see Definition 5.1).
- `has_open_branch`: if the current node is stronger than any previous node, there is an open branch (see Definition 5.20).

6.8.2 Tableau Rules

Saturation rules (see Figure 5.3) will be applied when we parse the formula from a string to the list of lists representation. Furthermore, ($\Box$*False*) always rule (see Figure 5.2) is associated to the minimal covering object by `is_not_X_covering` function and the others, ($\Box||$) and ($\Box\&$) always rules (see Figure 5.2), are directly implemented by loops of the tableau algorithm which will be introduce below. Finally, the next state rule (see Figure 5.4) will be applied to the node's formula by means of the `next` function.

6.8.3 Tableau algorithm

The tableau algorithm is divided into two parts, one related with the environment player moves and the other with the system player moves.

<u>`Environment turn`</u>:

Firstly, the environment player will start first

```
self.initial_node = tableau(self.initial_formula, 1, None)
self.is_open = self.tableau(self.initial_node, ENVIRONMENT_PLAYING, 1)
```

and every time it starts playing will check whether the current branch is a winner or a loser branch. It is a winning branch for the environment whether the current node is weaker or equal than a previously failed node. While it is a loser branch for the environment either if the current node is stronger than a previously successful node or if one predecessor node has a stronger formula.

```
if node.implicate_a_failure_node(self.failure_nodes):
    return False

if node. is_implicated_by_success_node(self.success_nodes):
    return True

if node. has_open_branch():
    return True
```

Next, the environment extracts information about which of its variables are taking part in the present in order to calculate the TNF formula of the conjunction of the safety formula and node formula.

```
formula_env_vars =  get_environment_current_variables(node.formula)
```

```
env_vars_node = self.environment_safety_formula_variables.union(formula_env_vars)
node_tnf = self.calculate_tnf_with_node(node.formula, env_vars_node)
```

Then, the moves and each environment valuation are scored to generate the iterator of the minimal coverings, as we have seen in Section 6.7. When a specific minimal covering is requested, it returns sorted by the environment valuation (from most likely to least likely to generate an open branch).

```
if is_not_X_covering(node_tnf):
   return False

else:
    env_valuations_sorted, minimal_coverings_iterator = sort_minimal_coverings(node_tnf)
```

In addition, the order will be reversed due to the fact that successors are AND-successors and with one of them failing the whole minimal covering will fail and, therefore, the environment player will have won to the system player in this minimal covering. However, the environment will only win ensuring that the current node is a failed node . Whereas the current node will become a successful node if for every move of the minimal covering the environment lose, generating open branches.

```
for minimal_X_covering in minimal_X_coverings_iterator:
    minimal_X_covering.reverse()
    env_valuations_sorted.reverse()
    is_open = False
    for i, environment_move in enumerate(minimal_X_covering):
```

After selecting a move of a specific environment valuation, we check whether the move is consistent in strict-futures formulas to avoid cycles between inconsistent nodes. In the case of a consistent move, the system will start playing, otherwise, the environment will generate another minimal covering.

```
        env_assignment = env_valuations_sorted[i]
        strict_futures_i = delete_inconsistent_sets(environment_move[1])
        if not strict_futures_i: break
        successor_node = TableauNode(environment_move, depth, node)
        is_open = self.tableau(child_node, SYSTEM_PLAYING, depth)
```

At this point, tableau branch can return four different possibilities:

1. True when is a open branch

```
        if is_open is True: continue
```

2. False when is a closed branch

```
        if is_open is False: break
```

3. A positive integer (resp. negative integer) when searching for an open branch it finds a success node (resp. failure node) which is stronger (resp. weaker) than one of its predecessor nodes. In this case, the absolute value indicates the depth where the weaker (resp. stronger) predecessor node is located.

```
            if depth != abs(is_open): return is_open
```

4. When a predecessor node became a success or failure node, the tableau must return to that tableau point. For example, whether it returns -2, the tableau will go back to depth 2 and convert that node into a failure node, whereas, if it returns 1, the tableau will go back to depth 1 and convert that node into a success node

```
            if depth == abs(is_open) and is_open > 0:
                return True

            if depth == abs(is_open) and is_open < 0:
                return False
```

Finally, if a minimal covering is successful for the system, the node that generated it will be added to the success node list, whereas, if all minimal coverings are successful for the environment, the node that generated it will be added to the failure node list.

```
if is_open:
    self.success_nodes.append(node.formula)
    is_success_previous_node = node.success_previous_node()
    if is_success_previous_node:
        return is_success_previous_node

    else:
        return True
else:
    try:
        apply_next_state_rule (minimal_X_coverings_iterator)

    except StopIteration:
        self.failed_nodes.add(node.formula)
        is_failed_previous_node = node.failed_previous_node()
        if is_failed_previous_node:
            return is_failed_previous_node * -1

        else:
            return False
```

`System turn`:

The main purpose of the system is to apply the next state rule to all the strict-future formulas with corresponding saturation rules.

```
formula_after_next = apply_next_state_rule (node.formula)
successor_node = TableauNode(formula_after_next, depth+1, node.previous_node)
is_open = self.tableau(child_node, ENVIRONMENT_PLAYING, depth+1)
return is_open
```

6.9 Automatic benchmark generation

The generation of automatic tests is very important for the verification and testing of the prototype. In addition, through multiple executions, we can detect areas in which the prototype can be improved.

The structure of the test will depend on how many environment variables, system variables and what temporal system interval you want to include in the benchmark.

```
Number of environment variables = 3
Number of system variables = 2
System temporal interval = [1,10]
```

Once these parameters have been set, we establish two different AND-formulas; one for the environment variables and the other for the system formulas.

```
Environment AND-fomula = (p0_e & p1_e & p2_e)
System AND-formula = (G[1,1000](s0) & G[1,1000](s1) )
```

And we create an implication between both, being the conjunction system formulas the logical consequence.

```
Initial Formula
True

Safety Formula
(p0_e & p1_e & p2_e) -> (G[1,10](s0) & G[1,10](s1) )

Environment Global Constraints
True
```

At this point, the above specification is realizable and generates an open tableau but we also need automatic benchmark to test unrealizable specifications. Therefore, we include the negation of one of the temporal system formulas so that the specification becomes unrealizable.

```
Initial Formula
True

Safety Formula
(p0_e & p1_e & p2_e) -> (G[1,10](s0) & G[1,10](s1) )
F[1,10](-s1)

Environment Global Constraints
True
```

6.10 Benchmarking

In the following tables we will show the results of testing both the examples used during the memory (Table 6.2) and some of the automatically generated tests (Table 6.3 and 6.4). Note that n_e refers to number of environment variables and n_s to number of system variables.

File	Corresponding Example	Expected Result	Result	Time(s)
benchmark1.txt	Example 5.5	Open Tableau	Open Tableau	1.06 s
benchmark2.txt	Example 5.6	Closed Tableau	Closed Tableau	2.32 s
benchmark3.txt	Example 5.7	Open Tableau	Open Tableau	1.69 s
benchmark4.txt	Example 5.8	Open Tableau	Open Tableau	1.71 s
benchmark5.txt	Example 5.9	Open Tableau	Open Tableau	1.76 s
benchmark6.txt	Example 5.10	Closed Tableau	Closed Tableau	5.12 s
benchmark7.txt	Example 5.11	Open Tableau	Open Tableau	23.16 s

Table 6.2: Memory examples Benchmarks

File	n_e	n_s	Expected Result	Result	Time
benchmark_1_1_[1,10].txt	1	1	Open Tableau	Open Tableau	1.53 s
benchmark_1_8_[1,10].txt	1	8	Open Tableau	Open Tableau	9.86 s
benchmark_5_3_[1,10].txt	5	3	Open Tableau	Open Tableau	20.36 s
benchmark_5_5_[1,10].txt	5	5	Open Tableau	Open Tableau	46.16 s
benchmark_5_8_[1,10].txt	5	8	Open Tableau	Open Tableau	121.98 s
benchmark_8_1_[1,10].txt	8	1	Open Tableau	Open Tableau	85.9 s
benchmark_8_8_[1,10].txt	8	8	Open Tableau	Open Tableau	1031.55 s

Table 6.3: Realizable Automatic Benchmarks

File	n_e	n_s	Expected Result	Result	Time
benchmark_1_1_[1,1000].txt	1	1	Closed Tableau	Closed Tableau	0.35 s
benchmark_1_8_[1,1000].txt	1	8	Closed Tableau	Closed Tableau	0.44 s
benchmark_5_3_[1,1000].txt	5	3	Closed Tableau	Closed Tableau	0.41 s
benchmark_5_5_[1,1000].txt	5	5	Closed Tableau	Closed Tableau	0.51 s
benchmark_5_8_[1,1000].txt	5	8	Closed Tableau	Closed Tableau	0.44 s
benchmark_8_1_[1,1000].txt	8	1	Closed Tableau	Closed Tableau	0.39 s
benchmark_8_8_[1,1000].txt	8	8	Closed Tableau	Closed Tableau	0.48 s

Table 6.4: Unrealizable Automatic Benchmarks

Referring to unrealizable benchmarks, Table 6.4 shows a good performance when the branch is closed due to an inconsistent node (see Definition 5.1). However, the times obtained in “benchmark2.txt” and “benchmark6.txt” are higher than expected because in the AND-nodes (see Definition 5.1a) the system is not able to choose as first option the successor that fails in the next state and, therefore, the successor that closes the node. Consequently, as it is an AND-node, we develop branches that will be superfluous when the node is closed.

Looking at Table 6.3 and the generated traces, we notice that when a node creates a move m_i without environment variables we can improve the prototype performance because, at this point, we can decide whether the set of formulas in m_i is consistent without using a SAT-solver. Moreover, as our syntax only contains $\bigcirc$ temporal operators we can ensure the satisfiability by checking the absence of inconsistencies.

In conclusion, although the prototype works well there is a lot of work ahead. Testing with new and extensive collections of benchmarks will help us to significantly improve the performance of the prototype.

CHAPTER 7

Conclusions and Future work

We have introduced the first tableau method to decide realizability of a safety specificaiton modelled by a sublanguage of LTL. For that, we have defined a new normal form of temporal formulas (TNF) which precisely capture the information that each player (environment and system) has to reveal at each step. Furthermore, in spite of the fact that the objective of developing a functional prototype for solving LTL realizability and synthesis problem by a tableau algorithm has been achieved and shows promising results, there is still a lot of work ahead.

Our most urgent future work is to experiment with a wide collection of benchmarks in order to improve the performance of the prototype. We want to extend the method to more expressive languages, including the handling of richer propositional languages (like numeric variables and enumerates) by combining realizability tableau rules with tableau reasoning capabilities for these domains. Moreover, we also plan to compare our results with other state-of-art LTL Realizabity and Synthesis tools like AbySynth.

Bibliography

[1] Andreas Nonnengart and Christoph Weidenbach. Chapter 6 - computing small clause normal forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, Handbook of Automated Reasoning, pages 335–367. North-Holland, Amsterdam, 2001. See page 7.

[2] Evert Willem Beth. *Semantic Entailment and Formal Derivability*. Noord-Hollandsche, 1955. See page 8.

[3] Raymond M. Smullyan. *First-Order Logic*. New York [Etc.]Springer-Verlag, 1968. See page 8.

[4] Sat competitions. `http://www.satcompetition.org/`. See page 9.

[5] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020. See page 9.

[6] Sat competition winners on the sc2020 benchmark suite. `https://rise4fun.com/Z3/tutorial/guide`. See page 10.

[7] Yves Crama and Peter L Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge University Press, 2011. See page 10.

[8] Alexey Ignatiev, Alessandro Previti, and Joao Marques-Silva. SAT-Based formula Simplification. In *SAT*, 2015. See pages 10, 34.

[9] Howard Jay Aizenstein. *On the Learnability of Disjunctive Normal Form Formulas and Decision Trees*. PhD thesis, USA, 1993. UMI Order No. GAX93-28958. See page 11.

[10] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer Publishing Company, Incorporated, 3rd edition, 2012. See page 12.

[11] Jose Gaintzarain, Montserrat Hermo, Paqui Lucio, Marisa Navarro, and Fernando Orejas. Dual systems of tableaux and sequents for pltl. *The Journal of Logic and Algebraic Programming*, 78(8):701–722, 2009. See page 12.

[12] David Harel and Amir Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI Series*, pages 477–498. Springer, 1984. See page 14.

[13] Therac-25: un diseño de interacción mortal. `https://lsi2.ugr.es/mvega/docis/aluwork/roddesastres/therac.htm`. See page 15.

[14] Pentium ii math bug? `http://www.rcollins.org/secrets/Dan0411.html`. See page 15.

[15] The worst computer bugs in history: The ariane 5 disaster. `https://www.bugsnag.com/blog/bug-day-ariane-5-disaster`. See page 15.

[16] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. 01 2001. See page 15.

[17] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool

for symbolic model checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification*, pages 359–364, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. See page 15.

[18] Kenneth L. McMillan. Symbolic model checking. In *CAV*, 1993. See page 15.

[19] Formal methods. http://web.stanford.edu/class/cs237b/pdfs/lecture/lecture_15.pdf. See page 15.

[20] Alonzo Church. Logic, arithmetic, and automata. 1962. See page 15.

[21] Reactive synthesis competition (syntcomp). http://www.syntcomp.org. See page 15.

[22] Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin, and Ocan Sankur. Abssynthe: abstract synthesis from succinct safety specifications. In Krishnendu Chatterjee, Rüdiger Ehlers, and Susmit Jha, editors, *Proceedings 3rd Workshop on Synthesis, SYNT 2014, Vienna, Austria, July 23-24, 2014*, volume 157 of *EPTCS*, pages 100–116, 2014. See page 16.

[23] Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, 57(1-2):3–36, 2020. See page 16.

[24] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018. See page 16.

[25] Thibaud Michaud and Maximilien Colange. Reactive synthesis from ltl specification with spot. In *Proceedings Seventh Workshop on Synthesis, SYNT@CAV 2018*, Electronic Proceedings in Theoretical Computer Science, 2018. See page 16.

[26] Barbara Jobstmann and Roderick Bloem. Optimizations for ltl synthesis. In *2006 Formal Methods in Computer Aided Design*, pages 117–124, 2006. See page 19.

[27] Rüdiger Ehlers. Unbeast: Symbolic bounded synthesis. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Part of the Joint European Conferences on Theory and Practice of Software*, TACAS'11/ETAPS'11, page 272–275, Berlin, Heidelberg, 2011. Springer-Verlag. See page 19.

[28] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for ltl synthesis. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification*, pages 652–657, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. See page 19.

[29] Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5–6):519–539, oct 2013. See page 19.

[30] J. Goldsmith, Matthias Hagen, and Martin Mundhenk. Complexity of DNF minimization and isomorphism testing for monotone formulas. *Inf. Comput.*, 206:760–775, 2008. See page 34.

[31] Noel Arteche and Montserrat Hermo. Prime implicant enumeration via QBF solvers. In *International Workshop on Quantified Boolean Formulas and Beyond, at 24th International Conference on Theory and Applications of Satisfiability Testing*, 2021. See page 53.

[32] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. pytest x.y, 2004. See page 53.

[33] Erik Rose. The fastest pure-python peg parser. https://github.com/erikrose/parsimonious, 2012. See page 54.

[34] Guido Van Rossum. *The Python Library Reference, release 3.8.2.* Python Software Foundation, 2020. See page 59.