University of Udine

Department of Computer Science, Mathematics and Physics
PhD Course in Computer Science, Mathematics and Physics

Dissertation

TEMPORAL LOGIC SPECIFICATIONS: EXPRESSIVENESS, SATISFIABILITY AND REALIZABILITY

Candidate
Luca Geatti

Supervisor
Alessandro Cimatti

Cosupervisors
Angelo Montanari
Stefano Tonetta

Cycle XXXIV

A ducj i fruts dal mont,
ch'a no dismetin mai di cori pai cjamps,
ancje intal mieç dal temporâl.

SUMMARY

The main topic of this thesis concerns temporal logics, with particular attention to their expressive power and to the satisfiability and realizability problems.

Temporal logics are nowadays a well-established formalism for expressing properties about sequences. Their connection with first- and second-order logic, automata and formal verification makes temporal logics not only a powerful theoretical framework, but also a valuable tool in practical scenarios (*e.g.*, for the specification of concurrent systems). On the theoretical side, one typical problem when dealing with a temporal logic is to characterize exactly its expressive power, that is to give the set of all and only the properties that it can formalize. On a more practical side, there are two important problems that are considered when using temporal logics as specification languages: (i) the satisfiability problem, that is finding whether a given formula admits at least one model; and (ii) the realizability problem, namely to find whether a given formula is implementable. In formal verification, satisfiability can be used as a sanity check for detecting vacuous specifications (*i.e.*, valid or unsatisfiable formulas), while realizability can be used to check the existence of correct-by-construction implementations (and their consequent syn-

thesis).

In this thesis, we try to stay in the intersection between the theoretical and the practical sides of temporal logics, by accompanying the theoretical results (whenever possible) with algorithms, implementations and experimental evaluations.

Theory We introduce three fragments of Linear Temporal Logic with Past (LTL+P) and study their expressive power: (i) for the first fragment (called $\mathsf{LTL}_{\mathsf{EBR}}$+P), we prove that it is *expressively complete* with respect to the (semantically) safety fragment of LTL+P (a safety property is a property in which a violation is irremediable); (ii) the second fragment (called $\mathsf{LTL}_{\mathsf{EBR}}$) is obtained from $\mathsf{LTL}_{\mathsf{EBR}}$+P by removing past operators; we prove that $\mathsf{LTL}_{\mathsf{EBR}}$ is *strictly less expressive* than full $\mathsf{LTL}_{\mathsf{EBR}}$+P; (iii) the third fragment is called GR-EBR and it is an extension of $\mathsf{LTL}_{\mathsf{EBR}}$+P for expressing properties beyond the safety fragment; we compare its expressive power with the Temporal Hierarchy of Manna and Pnueli. In addition we propose a first-order *syntactical* characterization (called Safety-FO) that captures the *semantically safety* fragment of the first-order logic of one successor: this result can be considered as the version of Kamp's Theorem for safety properties.

Problems and Algorithms We consider the satisfiability problem of LTL+P and the realizability problem from $\mathsf{LTL}_{\mathsf{EBR}}$+P and GR-EBR specifications. Particular attention is devoted to the use of *symbolic algorithms* instead of classical explicit-state ones. We implement all the algorithms that we propose and we compare them with competitor tools. From the outcomes of the experimental evaluations, it is often evident that our symbolic techniques can solve instances of sizes that are prohibitive for the other tools based on an explicit-state representation.

Last but not least, we consider an industrially relevant problem in the context of real-time requirements (*i.e.*, properties expressing not only the ordering between events but also the amount of time elapsed between two events). We define and formalize the *compatibility problem of timed requirements*, give symbolic algorithms for this problem, and implement and evaluate the proposed procedure.

SOMMARIO

L'argomento principale di questa tesi riguarda le logiche temporali, con particolare attenzione alla loro potenza espressiva e ai problemi di soddisfacibilità e realizzabilità.

Le logiche temporali sono oggigiorno un formalismo ben consolidato per esprimere proprietà su sequenze. La loro connessione con logiche al primo e second'ordine, con gli automi e con la verifica formale ha reso le logiche temporali non solo un potente framework teorico, ma anche un prezioso strumento pratico (per esempio per la specifica di sistemi concorrenti). Sul lato teorico, uno dei tipici problemi quando si lavora con una logica temporale è di caratterizzare esattamente la sua potenza espressiva, cioè di dare l'insieme di tutte e sole le proprietà che essa è in grado di formalizzare. Su un lato più pratico, ci sono due importanti problemi che vengono considerati quando si usano le logiche temporali come linguaggi di specifica: (i) il problema di soddisfacibilità, cioè stabilire se la formula data ammette almeno un modello; e (ii) il problema di realizzabilità, cioè stabilire se la formula data è implementabile. In verifica formale, la soddisfacibilità può essere usata come un controllo per individuare specifiche vacue (cioè formule valide o insoddisfacibili), mentre la realizzabilità può essere usata per controllare l'esistenza

di implementazioni corrette per costruzione (e la loro conseguente sintesi).

In questa tesi, cerchiamo di porci nell'intersezione tra il lato teorico e quello pratico delle logiche temporali, accompagnando i risultati teorici (quando possibile) con algoritmi, implementazioni e valutazioni sperimentali.

Teoria Introduciamo tre frammenti della Logica Temporale Lineare con Passato (LTL+P) e studiamo la loro potenza espressiva: (i) per il primo frammento (chiamato $\mathsf{LTL}_{\mathsf{EBR}}$+P) dimostriamo che è *espressivamente completo* rispetto al frammento safety semantico di LTL+P (una proprietà si dice essere di safety se ogni sua violazione è irrimediabile); (ii) il secondo frammento (chiamato $\mathsf{LTL}_{\mathsf{EBR}}$) è ottenuto da $\mathsf{LTL}_{\mathsf{EBR}}$+P rimuovendo gli operatori al passato; dimostriamo che $\mathsf{LTL}_{\mathsf{EBR}}$ è *strettamente meno espressivo* di $\mathsf{LTL}_{\mathsf{EBR}}$+P; (iii) il terzo frammento è chiamato GR-EBR ed è un'estensione di $\mathsf{LTL}_{\mathsf{EBR}}$+P per esprimere proprietà che vanno oltre al frammento safety; confrontiamo la sua potenza espressiva con la Temporal Hierarchy di Manna e Pnueli. Inoltre proponiamo una caratterizzazione sintattica al prim'ordine (chiamata Safety-FO) che cattura il frammento safety semantico della logica al prim'ordine con un successore: questo risultato può essere considerato come la versione del Teorema di Kamp per proprietà safety.

Problemi e Algoritmi Consideriamo il problema di soddisfacibilità per LTL+P ed il problema di realizzabilità per specifiche di $\mathsf{LTL}_{\mathsf{EBR}}$+P e GR-EBR. Particolare attenzione è rivolta all'uso di *algoritmi simbolici* al posto di quelli classici espliciti. Implementiamo tutti gli algoritmi che abbiamo proposto e li confrontiamo con gli altri tool concorrenti. Dai risultati delle valutazioni sperimentali, è spesso evidente che le nostre tecniche simboliche riescono a risolvere istanze di dimensioni che sono proibitive per gli altri tool basati su una rappresentazione esplicita.

Ultimo, ma non per importanza, consideriamo un problema di rilevanza industriale nel contesto di requisiti real-time (cioè proprietà che esprimono non solo l'ordine tra gli eventi ma anche la quantità di tempo passata tra i due). Definiamo e formalizziamo il problema di *compatibilità di requisiti temporizzati*, diamo degli algoritmi simbolici per questo problema, e implementiamo e valutiamo la procedura proposta.

ACKNOWLEDGEMENTS

You know that you are surrounded by beautiful people when even your acknowledgements' page is too long. Let's start.

The biggest thank goes to my family (Gabriella, Marco, Vilma, Armando, Gloria, Carlo, Marco, Andrea, e ducj chei atris ca nus cjalin già di la su) that has been a point of stability during all my journeys around Europe, making me understanding what a real *home* is, and (surprisingly) helping me also with some aspects of my research.

I feel honored to have the friends that I have. Thanks a lot to Sara Di Lorenzo, Nicola Gigante, Matteo 'Fire' Simonetti, Tommaso Gobbo, Marco Bortolato, Marco 'Tank' D'Agostini, Elisa, Riccardo Marinato (for all the bottles of wine that he brought with him in train), Elia Calligaris, Marta Serafini and Michele Collevati. I'm eternally grateful to Tank for making me starting my working life as a mason when I was eighteen (it opened unimaginable doors), and good luck to Marta and Michele for their PhD.

Without Nicola Gigante, I'd have quit my PhD two months before its start. I have literally no words to explain my gratitude for its patience and support. Those who know him know also that this is a little acknowledgement for a great man.

I am enormously grateful to my supervisors, Alessandro Cimatti, Angelo Montanari and Stefano Tonetta for all the things that I learned in these 3 years. Their outstanding capacities in their work, their speed of thought, the rigorous approach to work and their creativity have always left me amazed.

I wish also to thank my PhD colleagues Francesco Fabiano and Enrico Magnago for the moments of mutual help that are really precious to me.

با تشکر فراوان از هانی بیرامی دوست وفادار در این 3 سال.
در همه چیز موفق باشید، دوست من!

Forgetting the one who started me to mathematics would be a crime. Thanks to my highschool professor Luca Marinatto, who, first of all, is a metalhead like me (and this would be enough), and he has started in me the real passion in maths (now more than 11 years ago). Thank you! I am grateful also to Riccardo Giannitrapani (I know that he always goes to great lengths to remember the faces of all his students) for the passion and the poetry that makes for his students.

Thanks a lot to my friend Jack for all the unforgettable moments that we spent together: we spoke a lot without even saying a word.

I am very grateful to Dennis for all the PdV that each month he never forgets to send me. Thank you for such valuable advices. I'll continue to take you as an example.

Finally, I would like to express my gratitude to my hometown. Running through your orange and green fields of wheat has been and will always remain the biggest and purest form of freedom I'll ever know.

Mandi!

October 26th, 2021
Girlan/Cornaiano

Luca Geatti

CONTENTS

I Introduction **1**

1 Introduction **3**

1.1 A Brief History of Logic and Computer Science . . . 4

1.2 The Theory of Linear Orders 7

1.2.1 Sequential Calculus 8

1.2.2 Satisfiability, Model Checking and Realizability 10

1.3 Linear Temporal Logic 14

1.4 The Classical Algorithms 17

1.4.1 Algorithms for LTL+P Satisfiability 17

1.4.2 Algorithms for LTL+P Realizability 19

1.4.3 Symbolic Algorithms 23

1.5 Contributions 30

1.5.1 Theory 30

1.5.2 Problems and Algorithms 31

1.6 Organization of the thesis 37

1.6.1 Publications 39

II Theory **41**

2 Background **43**
2.1 Sequential Calculus 44
2.1.1 Definition of S1S 44
2.1.2 Properties of S1S 47
2.1.3 The Safety, co-Safety and Liveness classes . . 47
2.1.4 A Notation for Syntax and Semantics 49
2.2 Automata over finite and infinite words 50
2.2.1 Automata over Finite Words 50
2.2.2 Properties of Automata over Finite Words . . 52
2.2.3 Automata Over Infinite Words 53
2.3 Regular and ω-Regular Expressions 57
2.3.1 Regular Expressions 58
2.3.2 ω-Regular Expressions 59
2.3.3 Characterization of safety and co-safety classes 61
2.4 Linear Temporal Logics 61
2.4.1 Syntax 62
2.4.2 Semantics 63
2.4.3 Properties 66
2.4.4 Safety and co-Safety fragments of LTL 70
2.4.5 The Temporal Hierarchy 74

3 The $\mathsf{LTL}_{\mathsf{EBR}}$+P, $\mathsf{LTL}_{\mathsf{EBR}}$ and GR-EBR logics **81**
3.1 The $\mathsf{LTL}_{\mathsf{EBR}}$+P logic 82
3.1.1 Definition and Normal Form 82
3.1.2 Examples 84
3.1.3 Expressive Power 86
3.2 The $\mathsf{LTL}_{\mathsf{EBR}}$ logic 88
3.2.1 Definition and Normal Form 88
3.2.2 Expressive power 89
3.3 The GR-EBR logic 98
3.3.1 Definition 98
3.3.2 Example 98
3.3.3 Expressive Power 99
3.4 Conclusions 102

4 A first-order logic characterisation of safety and co-safety languages 105
4.1 Safety-FO and coSafety-FO 106
4.1.1 Notations 107
4.1.2 Discussion on the two fragments 108
4.1.3 Lemmas 108
4.1.4 Main Theorem 112
4.2 Safety-FO captures the safety fragment of LTL 122
4.3 Conclusions 125

III Problems and Algorithms 131

5 Background 133
5.1 LTL Satisfiability 134
5.1.1 Complexity 134
5.1.2 The automata-theoretic algorithm 135
5.1.3 The two-pass and graph-shaped tableau system 137
5.1.4 The one-pass and tree-shaped tableau system 140
5.2 Model checking 143
5.2.1 Symbolic Kripke Structures 145
5.2.2 Symbolic Automata 147
5.2.3 The SMV modeling language 151
5.2.4 Binary Decision Diagrams, SAT and SMT . . 154
5.2.5 Bounded Model Checking 157
5.2.6 K-Liveness 160
5.2.7 From LTL satisfiability to LTL model checking 162
5.3 LTL Realizability and synthesis 164
5.3.1 Definition of the problem 164
5.3.2 Decidability and Complexity 165
5.3.3 The classical approach 167
5.3.4 The classical approach to safety synthesis . . 170
5.3.5 The Safraless approach 173
5.3.6 Bounded Synthesis 175
5.3.7 The parity games approach 176
5.3.8 GR(1) realizability 177
5.3.9 Reactive Synthesis vs Parameter Synthesis . 180

6 Satisfiability of LTL+P specifications **183**
6.1 Related work 185
6.2 The SAT encoding of Reynolds' tableau 186
6.3 Extensions for finite traces and models extraction . . 196
6.3.1 Extension for LTL under finite traces 196
6.3.2 Extraction of models 198
6.4 Experimental evaluation 199
6.4.1 LTL over infinite traces 199
6.4.2 LTL+P 207
6.4.3 LTL over finite traces 208
6.5 Conclusions 209

7 Realizability of $\mathsf{LTL_{EBR}}$+P specifications **211**
7.1 Overview 212
7.2 From $\mathsf{LTL_{EBR}}$+P to deterministic SSA 213
7.2.1 From $\mathsf{LTL_{EBR}}$+P to Pastified-$\mathsf{LTL_{EBR}}$+P . . . 215
7.2.2 From Pastified-$\mathsf{LTL_{EBR}}$+P to Normal-$\mathsf{LTL_{EBR}}$+P 218
7.2.3 From Normal-$\mathsf{LTL_{EBR}}$+P to deterministic SSA 222
7.3 An optimal algorithm for $\mathsf{LTL_{EBR}}$+P realizability . . 228
7.3.1 Solving the game on the deterministic SSA . 228
7.3.2 Complexity of $\mathsf{LTL_{EBR}}$+P 229
7.4 Experimental Evaluation 235
7.4.1 Normalized Benchmarks 242
7.5 Conclusions 244

8 Realizability of GR-EBR specifications **245**
8.1 Overview 246
8.1.1 Contributions 247
8.1.2 Related work 248
8.2 A Framework of Safety Reductions for Realizability . 250
8.2.1 A sound but not complete safety reduction . 250
8.2.2 Definition of safety reduction 251
8.2.3 Link between realizability and model checking 251
8.2.4 Formalization of Bounded Synthesis 255
8.3 A Safety Reduction for GR-EBR 256
8.3.1 The automaton with the GR(1) condition . . 258
8.3.2 Degeneralization 261
8.3.3 Reduction to Safety for R(1) objectives . . . 263
8.4 Experimental Evaluation 266
8.4.1 Description of the competitor tools 266

8.4.2 Description of the benchmarks set 267
8.4.3 Discussion of the results 268
8.5 Conclusions 270

9 Compatibility Check of Timing Requirements 273
9.1 Introduction 274
9.1.1 Related Work 276
9.1.2 Outline 277
9.2 Problem Statement 277
9.2.1 Domain formalization 277
9.2.2 NP-hardness 280
9.3 Verification 281
9.3.1 Reduction to Model Checking 281
9.3.2 Encoding into SMT 285
9.3.3 Optimization of the Encoding 289
9.4 Parameter Synthesis 290
9.5 Experimental Evaluation 291
9.6 Conclusions 293

10 Conclusions 299

IV Appendix 327

A Proofs 329
A.1 Proofs of Chapter 7 329
A.2 Formalization into Timed Interface Automata (Chapter 9) 349
A.3 Quantifier-free encoding for requirements with only finite bounds (Chapter 9) 354
A.3.1 Positive Dependencies 354
A.3.2 Strict Semantics 355
A.3.3 Weak Semantics 355
A.4 Quantifier-free encoding for Convex Dependencies (Chapter 9) 359

Part I

Introduction

CHAPTER

1

INTRODUCTION

Formal logic, also referred to as mathematical logic, offers an unambiguous language for formalizing human and mathematical reasoning. Formal logic stands at the basis of computer science. Their connection is so strong to have brought some of the most renowned researchers on this areas to state that [111]:

Logic has turned out to be significantly more effective in computer science than it has been in mathematics.

Among the fields of study standing at the intersection between formal logic and computer science, there is the research on *temporal logics* and *automata theory*, whose goal is to study properties of sequences (defined as linear orders) or more general structures, with different formalisms. Over the years, the results obtained in these two fields of study influenced and have been influenced by *formal verification*, the study of formal and mathematical tools for the development of correct systems, ranging from digital circuits to cyber-physical systems. While temporal logics and automata theory can arguably be considered as being more theoretical, formal verification

requires techniques that work well in practice, that is, for example, fast and efficient algorithms.

In this thesis, we investigate and propose some techniques, theorems and algorithms that stand in the intersection between temporal logics, automata theory and formal verification. Particular effort will be devoted to accompany the theoretical results, wherever possible, with experimental evaluations that show the impact of our algorithms in practice.

In this chapter, we want to give an introduction as self-contained as possible to the topics of this thesis.

1.1 A Brief History of Logic and Computer Science

The connection between logic and computer science is so tight that, actually, logic constituted the core for the birth of computer science. This is perfectly summarized by the following quote by Georg Gottlob [109]:

Computer Science is the continuation of Logic by other means.

Going back to the roots of formal logic, and actually to the roots of computer science, it is impossible not to mention Gottfried Leibniz, the creator (together with Isaac Newton) of the infinitesimal calculus. Leibniz, in what Martin Davis defines as the *Leibniz's Dream* [64], aspired to create what he called the *characteristica universalis*, a formal language able to embrace the whole human knowledge, extended with a calculus, called *calculus ratiocinator*, for entaling only true consequences starting from known facts. Leibniz's dream is summarized in the following famouse quote by him:

[...] if controversies were to arise, there would be no more need of disputation between two philosophers than between two calculators. For it would suffice for them to take their pencils in their hands and to sit down at the abacus, and say to each other (and if they so wish also to a friend called to help): Calculemus!

From this quote, it is already clear how Leibniz's dream was about the mechanization of reasoning. Despite being far from developing his *characteristica universalis*, Leibniz made fundamental progress in the *calculus ratiocinator*, which led to what was probably the first

formal language: its main intuition was to use an algebraic notation for logic to denote operations over concepts, just like it is used in arithmetic for denoting operations over natural numbers.

The algebraic notation of Leibniz was then formalized by George Boole, in its book *The laws of Thought* [27], and resulted in what is nowadays called Boolean algebra, or, equivalently, *propositional logic*. One of the most important limitations of Boolean algebra is that it is not able to distinguish, for example, between the statement "there exists a researcher that works on logic" and the statement "all researchers work on logic". This limitation was overcomed by Gottlob Frege, in its *Begriffsschrift* [94], by introducing the *quantification on predicates*. Frege based its logic on the algebraic notation of Boole but introducing also the two symbols $\exists$ and $\forall$ to denote the *existence* and the *totality* of the elements. The resulting formalism is what nowadays is called *first-order logic*.

Frege is also known for being the first advocate of *logicism*, the school of thought according to which arithmetic, and in general the entire mathematics, can be formalized (or better, axiomatized) in logic. In his *Grundgesetze der Arithmetik* [95], he claimed that he had succeeded in proving the reduction from arithmetic to logic. Nonetheless, famous is the letter by Bertrand Russell to Frege stating that some axioms in Frege's proof could be used for deriving a contradiction, thus breaking down the consistency of its theory for an axiomatization of arithmetic. That letter reports what is nowadays called *Russell's Paradox*, stating that if S is defined as the set of sets that do not belong to themselves, then S belongs to itself if and only if S does *not* belong to itself. This paradox undermined the foundations of set theory.

After the flaw in Frege's proof became known among the mathematicians, a large class among them began to ask whether mathematics as known until then was consistent at all, in the sense of whether it was not self-contradictory. Given the apparent difficulties in creating stable foundations for mathematics, many spoke of a crisis of foundations of mathematics.

In 1900, David Hilbert, one of the most influential mathematicians of modern times, proposed 23 problems to the community. In the preface, the motivations for the introduction of those problems were clear: showing that *any problem is solvable*. Hilbert's belief was really that for any problem it is always possible to prove, in a finite number of unambiguous steps (with what nowadays is called

an *algorithm*), its truth or its untruth. Hilbert's motto was the following:

We must know
We will know

The coherence of arithmetics stood in the second place of Hilbert's list. Later on, in 1928, along with his students John von Neumann and Wilhelm Ackermann, Hilbert proposed two main problems about logic, that would help to solve the coherence of arithmetics. The first one was about proving the *completeness* of first-order logic, that is proving whether for any (semantically) true sentence expressible in first-order logic, there exists a finite proof of its truth starting from the axioms and using deduction rules (syntactical truth). The second problem is called *Decision Problem* (or *Entscheidungsproblem*), and asks whether for any sentence written in the language of first-order logic, there exists a finite proof of its truth or its untruth starting from the axioms and using deduction rules. In other words, the decision problem asks to find whether, for any sentence, either its truth or its untruth is provable.

In 1929, in his PhD thesis [110], Kurt Gödel proved the completeness of first-order logic, thus giving a positive answer to the first of Hilbert's questions of 1928. But the real revolution started in 1931, with the proof by the same Kurt Gödel that there exists *unprovable* statements in *all* the logic systems powerful enough to express Peano arithmetics. This is also known as the *first incompleteness theorem* [108]. Using this result, Gödel proved also that the consistency of arithmetic, which is the second among Hilbert's problems, is *unprovable* within the theory of arithmetic itself. This results is known as *second incompleteness theorem* [108]. The two incompleteness theorems are commonly recognized as the end of Hilbert's program of giving consistent logical foundation of the entire mathematics.

The *Entscheidungsproblem*, the second of the 1928's problems by Hilbert, required a mechanization of the calculus: given a sentence, the problem asks to find a proof of its truth or untruth that is so simple that it can be executed by a machine. This challenge pushed a group of logicians to investigate the real nature of computation, in order to formally answer to the question *what is really a machine?* Among those researchers, Alonzo Church and Alan Turing proposed two different models of computation, nowadays called λ-*calculus* [42]

and *Turing Machines* [194], respectively, that with very simple steps are able to carry out complex tasks. This is generally recognized as the birth of computer science. Despite being very different, the two formalisms were proved to be equivalent. Most importantly, Turing machines are so general that there exists a type of them, called *Universal Turing Machines*, that can solve themselves all possible problems that any other Turing machine can solve. Given this generality, and also the difficulty in finding more powerful formalisms, it is generally believed that each problem that can be solved by a machine, can be solved by a Turing machine (or equivalently by λ-calculus) as well. This is known as *Church-Turing thesis*.

In the same 1936's paper introducing his new machines [194], Turing also proved that there exist undecidable problems, that is problems that cannot be solved by any Turing machine. The *Halting Problem* asks to find whether, given a Turing machine and a string as input, that Turing machine terminates when reading that input. Turing proved that the halting problem is undecidable, that is, there does not exists any Turing machine that solves this problem. This is closely related to the first incompleteness theorem by Gödel, but it extends it in some sense by formalizing the notion of computation and by proving that there exist statements whose truth or untruth cannot be derived by *any* machine (under the Church-Turing thesis). Interestingly, Gödel and Turing used the same technique for their proof of the first incompleteness theorem and the undecidability of the halting problem, respectively. The proof uses an argument called *diagonalization*, which was first used by Georg Cantor for proving that the set of all sets of natural numbers is greater than the set of natural numbers, and, consequently, that there exist infinitely many transfinite numbers.

1.2 The Theory of Linear Orders

In the previous section, we have seen that, by the first incompleteness theorem of Gödel and the negative answer of the Entscheidungsproblem by Turing, first-order logic in undecidable. This result gave rise to a fruitful line of research focusing on finding *decidable fragments* of first-order logic, that is logics that are weaker than the full first-order logic but for which there is an algorithm for deciding the truth or the untruth of each sentence of the logic. It turned out that, if one

restricts the *theory* of the logic, that is the domain of the interpretation and the interpretation of all symbols except from the variables, one can obtained decidable fragments also in the second-order logic, that is the logic obtained from the first-order one by allowing the Frege's quantifiers $\exists$ and $\forall$ to be applied also to n-ary relations, and not only to variables. One of the most known examples, which revealed central for the birth of formal verification, is the theory of *Sequential Calculus*.

1.2.1 Sequential Calculus

Sequential Calculus [44, 34] is a logical formalism for specifying properties of sequences (or, equivalently, linear orders). The first use of Sequential Calculus can be traced back to Church [44], who suggested to use it for specifying properties of sequential circuits (hence its name). In particular, he proposed to specify the relation *over time* $A(\mathsf{i}, \mathsf{o})$ between the *input* i and the *output* o of a sequential circuit by means of a formula $\phi(\mathsf{i}, \mathsf{o})$ of Sequential Calculus.

Time is so the crucial common point for modeling sequential circuits with Sequential Calculus: the behaviors over time of the formers are modeled by the latter by means of a finite or infinite linear order. Formally, this corresponds to fixing the *theory* of the logic: in Sequential Calculus, the domain is fixed to be $\mathbb{N}$ (the set of natural numbers), terms can be constructed only with the functional symbol $+1$ (interpreted as the successor function) starting from the constant symbol 0 (interpreted as the constant 0), and formulas can either compare two terms with the symbol $<$ (interpreted as the *strictly less* relation), or be more complex formulas using Boolean connectives, first-order quantifiers (like $\exists x$ and $\forall x$) or *monadic* quantifiers (like $\exists X$ and $\forall X$, where X denotes a set). For this reason, Sequential Calculus is often referred to as the *Monadic Second-order Theory of One Successor*, or S1S, for short.

Reactive Systems and Infinite Time

Circuits and, in general, cyber-physical systems are not closed-loop systems, but ruther open-loop systems, in the sense that they continuously maintain an interaction with an environment and their actions influence and are influenced by the actions of the environment: systems of this type fall under the name of *reactive systems*.

A reactive system is a system that is expected to maintain an ongoing interaction with the environment, rather than to produce a value on termination. Some reactive sytems are not even expected to terminate: this is the case, for example, of operating systems or processes controlling nuclear plants [141]. For this reason, typically the exact duration of the working cycle of these type of systems is unknown or too long to be precisely estimated. Therefore, the common assumption is to consider the duration of their working time to be really *infinite*. It follows that a good formalism for modeling reactive systems must have the possibility to specify requirements that constrain the behavior of the system even for an infinite amount of time. As we will see, Sequential Calculus is one of these, and this is the reason why it is usually interpreted over *infinite linear orders*, rather than on finite ones. With Sequential Calculus is thus possible not only to formalize properties that constrain a finite interval of the behavior of a circuit, like for example:

In the first three states, the output bit of the circuit must be low.

but also properties that constrain the behavior of the circuit for an *infinite* amout of time, like for instance the following requirement:

For each request in input, a grant will always be output in the future.

When Sequential Calculus is interpreted over *finite* linear orders, we talk about *Weak Sequential Calculus* (WS1S, for short).

Properties as Languages

Let us take a property written in natural language. The same property can be represented either by a formula of Sequential Calculus expressing it (in this case, we talk about *intentional* definition) or by the set of linear orders that satisfy it, *i.e.*, its models (*extensional* definition). We call the set of the models of a formula the *language* of the formula. In our setting, a language is therefore a *set* of (infinite or finite) linear orders. From now on, we will refer with ω-regular (resp. regular) languages to the set of languages expressible by means of a formula in S1S interpreted over infinite (resp. finite) linear orders. From now on, this definition holds not only for formulas of Sequential Calculus, but also for formulas of any other logic that we will see.

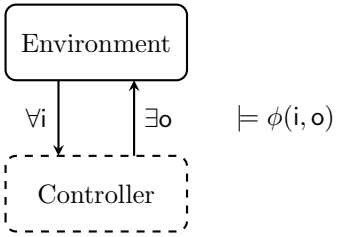


Figure 1.1: Visual representation of the realizability and reactive synthesis problems.

1.2.2 Satisfiability, Model Checking and Realizability

The groundbreaking connection drawn by Church between digital circuits and *logic* will turn out to be the basis for the birth of an entire field of research, that nowadays falls under the name of *formal verification*. In fact, Church's intuition, along with many others, opened the way for the algorithmic study (and its application in practical scenarios) of three fundamental problems over logical formulas: (i) satisfiability and validity; (ii) validity over structures, that nowadays goes under the name of *model checking*; and (iii) realizability and synthesis.

Realizability and *reactive synthesis* (or simply *synthesis*) refer to the process (and the related set of techniques) of synthesizing a sequential circuit starting from a logical specification $\phi(\mathsf{i}, \mathsf{o})$ (for example of Sequential Calculus). The synthesized circuit must choose the value of its output signals o in such a way that the specification $\phi(\mathsf{i}, \mathsf{o})$ is satisfied by all infinite computations of the circuit and for any choice of the input signals i, which are *not* under the control of the circuit. In fact, realizability and synthesis are usually modelled as two-player games between the Environment player, who tries to violate the specification $\phi(\mathsf{i}, \mathsf{o})$, and the Controller player, who tries to fulfill it by seeing only the history of Environment's moves made so far. A visual representation is depicted in Fig. 1.1. In some sense, the synthesized circuit must be *correct-by-construction*.

Realizability is the decision problem of finding whether such a circuit exists, while the synthesis problem requires to actually produce such a circuit in the cases in which it exists. Notably, these two problems were the original motivations of Church for the use of

Sequential Calculus as a specification language [44].

Satisfiability and *Validity* are the problems of establishing whether a formula is true in at least one structure and in all structures, respectively. As observed by Büchi in [34], the *satisfiability* and the *validity* problems are of crucial importance in formal verification, since they

> *[...] provide a method for deciding whether or not the input-output relation* $A(\mathsf{i}, \mathsf{o})$ *of a circuit satisfies a condition* $\phi(\mathsf{i}, \mathsf{o})$ *stated in Sequential Calculus.*

In fact, suppose that the behavior of a given circuit $A(\mathsf{i}, \mathsf{o})$ can be modeled by a formula $\phi_A(\mathsf{i}, \mathsf{o})$ of Sequential Calculus. The task of checking whether all computations of $A(\mathsf{i}, \mathsf{o})$ are compliant with a property $\phi(\mathsf{i}, \mathsf{o})$ of Sequential Calculus corresponds to the *validity problem* of a formula of type

$$\phi_A(\mathsf{i}, \mathsf{o}) \rightarrow \phi(\mathsf{i}, \mathsf{o})$$

In turn, since for any formula ϕ it holds that ϕ is valid if and only if $\neg\phi$ (its negation) is *not satisfiable*, tipically the validity problem is tackled by solving the corresponding *satisfiability* problem.

The *model checking* problem is very connected to validity. In fact, it takes as input not only a specification ϕ, but also a model of a circuit, in form of a state machine M [115], and checks whether *all* computations of the state machine are compliant with the specification, written in symbols as:

$$M \models \phi$$

For this reason, model checking is also known as *validity over structures*. We will formally define these three problems later in this thesis.

Decidability and Complexity

In his two seminal papers of 1960 [34, 35], Büchi showed that the satisfiability problem (and thus also the validity problem) of Sequential Calculus [34] (under both finite and infinite linear orders interpretation) is decidable. It follows that also the model checking problem is decidable. For proving this result, Büchi showed a translation from every formula of Sequential Calculus to an automaton recognizing

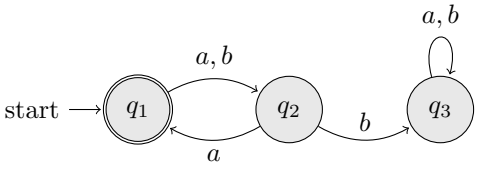


Figure 1.2: Büchi automaton accepting the words with an a in all even positions.

the same language of infinite words (we will use the term *"infinite word"* as a synonym of infinite linear order), and *vice versa*, thus proving an exact correspondence between Sequential Calculus and Büchi automata. An automaton is made of a set of states, some of which are *initial*, some are *final* and others are neither initial nor final. A *transition relation* specifies, from a given state and a letter, which states are reached. The distinguishing feature of Büchi automata is that they read and accept *infinite* words. An example of an automaton is depicted in Fig. 1.2. The automata used by Büchi, nowadays known as Büchi automata, have a particular type of acceptance condition: for each accepted word, starting from an initial state, the automaton has to visit a final state *infinitely many times*, when reading that word. The automaton in Fig. 1.2 accepts the language of words over the alphabet $\{a, b\}$ containg an a in all even positions. Interestingly, for proving the decidability of S1S [34], Büchi used automata over *infinite words*, while for proving the decidability of WS1S [35], he used automata over *finite words*, called Nondeterministic Finite Automata (NFA, for short) and Deterministic Finite Automata (DFA, for short).

Büchi also focused on realizability from Sequential Calculus specifications [36], showing the decidability of this problem as well. Although being decidable, these three problems for specifications of Sequential Calculus have very high complexities. Both satisfiability, model-checking and realizability are *nonelementary* [185, 36], meaning that the time spent by any Turing machine for solving any of these problems can be described only by an exponential function of unbounded height.

Safety, co-Safety and Liveness properties

An important class of ω-languages is the class of properties that express the fact that "*something bad never happens*", like a deadlock, or a simultaneous access to a critical section. For this reason, properties in this class are called *safety properties*, and the class is called *safety class*. A precise definition of this class will be given later on in this thesis.

Dual to the safety class, there is the class of *co-safety* languages, comprising all and only the properties expressing the fact that "*something good will eventually happen*", like termination of a program. The *safety* and the *co-safety* class are important in formal verification because, if it happens that the property falls under one of these two classes, than there are efficient, dedicated methods for solving the model checking, the satisfiability and the realizability problems. Intuitively, this is due to the fact that we can reduce reasoning on infinite linear orders to reasoning on *finite* ones, thus simplifying a lot the algorithms.

Another fundamental class is called the *liveness* class, and is made of the properties requiring a condition *"to happen infinitely many times"*, like for instance the assigment of a grant to a request at any time. As we will see, with the *liveness* class and its complement, it is possible to create a hierarchy of properties of increasing expressive power, called *temporal hierarchy* [140].

Real-time Properties

In some scenarios, the interaction between the environment and the systems must satisfy quantitative requirements about the time on which some events occur. Therefore it is important to model not only the ordering between the events of a system, which can be done with formulas of S1S, but also the exact time at which they occur or the precise time interval between two occurrences, expressed in some time measure. These properties are said to be *real-time properties* or, equivalently, *timed properties*. An example of real-time requirement is the following:

For each request in input, a grant will always be output in the future in at least 0.8 seconds and at most 3.0 seconds after the request.

Different formalisms have been introduced in the literature for expressing real-time properties. Among them, arguably, one of the most foundational ones is $\mathsf{S1S_t}$ [114]. This theory is interpreted no more on simple linear orders but instead on *timed linear orders*, in which each state is labeled by a time, which can belong either to the set of natural numbers $\mathbb{N}$ (in this case we talk about *discrete time*), or to the set of real numbers $\mathbb{R}$ (*continuous time*), or to other more complex domains, like $(\mathbb{R}, \mathbb{N})$, *i.e.*, the set of pairs of real and natural numbers (called *super dense time*). Accordingly, formulas of $\mathsf{S1S_t}$ can use the $<$ symbol to not only compare two state of a linear order but also their times.

In the part of this thesis dedicated to theory, we will introduce some fragments of temporal logics (some of them belonging to the *safety* class), which are in turn fragments of $\mathsf{S1S}$, and we will study their expressive power. In the part dedicated to algorithms, we will: (i) investigate new techniques for the satisfiability problem of specifications of Linear Temporal Logics, a fragment of $\mathsf{S1S}$; (ii) introduce new algorithms for solving the realizability of specifications written in the fragments that we introduced; and (iii) introduce a formalism for modeling a certain type of real-time properties, and study a new problem motivated by practical scenarios, called the *compatibility checking problem*, that asks whether, given some *local* real-time properties (each modeling a subsystem), they can in some sense be put together in order to accomplish a *global* real-time requirement (of the whole system). Before going to the contributions of this thesis, it is important to introduce temporal logics.

1.3 Linear Temporal Logic

In this section, we take a look at another formalism for defining properties of sequences, called *Linear Temporal Logic with Past* ($\mathsf{LTL{+}P}$, for short). $\mathsf{LTL{+}P}$ was introduced by Pnueli [159] in the late seventies as a way for expressing properties of computations of computer programs and, consequently, for verifying whether those computations are compliant with properties formalized in the language of $\mathsf{LTL{+}P}$.

Syntax

Differently from the Sequential Calculus formalism, which is a second-order logic, LTL+P is a *modal* logic. In addition to the classical Boolean operators, it features a particular set of operators for moving to the right (*future temporal operators*) or to the left (*past temporal operators*) from any point of a linear order. Its syntax is given by the following context-free grammar:

$$\left.\begin{array}{l} \phi := p \mid \end{array}\right\} \quad \text{propositional atoms}$$

$$\left.\begin{array}{l} \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \\ \neg\phi \mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \leftrightarrow \phi_2 \end{array}\right\} \quad \text{propositional connectives}$$

$$\left.\begin{array}{l} \mathsf{X}\phi_1 \mid \phi_1 \,\mathsf{U}\, \phi_2 \mid \phi_1 \,\mathsf{R}\, \phi_2 \mid \\ \mathsf{F}\phi_1 \mid \mathsf{G}\phi_1 \end{array}\right\} \quad \text{future temporal operators}$$

$$\left.\begin{array}{l} \mathsf{Y}\phi_1 \mid \phi_1 \,\mathsf{S}\, \phi_2 \mid \phi_1 \,\mathsf{T}\, \phi_2 \mid \\ \mathsf{P}\phi_1 \mid \mathsf{H}\phi_1 \mid \mathsf{Z}\phi_1 \end{array}\right\} \quad \text{past temporal operators}$$

where $p \in \Sigma$ is a proposition letter, and Σ is a finite alphabet. We briefly give the name of the future temporal operators, which are the most important in this section: (i) X is called *next* or *tomorrow*; (ii) U is the *until* operator; (iii) R is the *release* operator; and (iv) F and G are called *eventually* and *globally*, respectively. We call LTL the logic of LTL+P devoid of past temporal operators. LTL+P is interpreted over (typically infinite) state sequences, that is linear orders in which each state is labeled by the set of proposition letters that hold in that state. For example, the two requirements written in natural language in the previous section can be formalized in LTL (with only future operators) as follows. The requirement *"in the first three states, the output bit of the circuit must be low"* can be formalized by the LTL formula

$$\neg\mathsf{out} \wedge \mathsf{X}\neg\mathsf{out} \wedge \mathsf{XX}\neg\mathsf{out}$$

while the requirement *"for each request in input, a grant will always be output in the future"* can be specified by the LTL formula

$$\mathsf{G}(\mathsf{request} \rightarrow \mathsf{F}(\mathsf{grant}))$$

Expressive Power and Complexity

The choice of the operators of LTL+P proved to be carefully designed: LTL+P is equivalent to the first-order fragment of the Sequential Calculus [120] (S1S[FO], for short), that is S1S in which all quantifiers are applied only to classical variables and not to monadic ones. It follows that LTL+P is *strictly less expressive* than S1S. For example, the language of words over the alphabet $\{a, b\}$ containing the letter a in all even positions can *not* be expressed by any LTL+P formula [199], but it can be recognized by the Büchi automaton depicted in Fig. 1.2, and thus it can be formalized in S1S. Moreover, past temporal operators are not necessary for the expressive power of LTL+P: the logics LTL+P and LTL have the same expressive power [97], although some specifications of LTL can be formalized in an exponentially more succinct way using LTL+P [143].

As we will see in the next chapters, while the complexity of the satisfiability, the model checking and the realizability problems for S1S and first-order S1S specification is nonelementary [185], the complexities for LTL+P are lower: (i) satifiability and model checking for LTL+P specifications are PSPACE-complete [179]; (ii) realizability from LTL+P specifications if 2EXPTIME-complete [160, 164].

Real-time Extensions

Several extensions of LTL have been proposed in the literature for expressing also real-time properties, similarly to how S1S has been extended to $\mathsf{S1S_t}$. One is the logic of TPTL, which stands for *Timed Propositional Temporal Logic* [114]. TPTL extends LTL with the *freeze quantifier* "x.", which freezes the time of the state in which the formula is interpreted into a variable, say x; that variable can than be used in contraints of type $x \leq y + c$ or $y \leq x + c$, for some variable y and contant c. TPTL is interpreted over timed state sequences, *i.e.*, state sequences in which each state is labeled by a time value.

Another real-time extension of LTL is *Metric Temporal Logic*, MTL for short [125]. This logic extends the operators of LTL to be time-bounded: for example, the *eventually* operator F is replaced by $\mathsf{F}^{[a,b]}$, where $a, b \in \mathbb{R} \cup \infty$. For example, the classical time-bounded response propery *"for each request in input, a grant will always be output in the future in at most 3.0 seconds after the request"* can be

formalized by means of the following formula:

$$\mathsf{G}(\mathsf{request} \rightarrow \mathsf{F}^{[0.0,3.0]}(\mathsf{grant}))$$

Of course, also MTL is interpreted over timed state sequences.

Despite having different mechanisms for expressing real-time properties (TPTL has the freeze quantifier, while MTL has time-bounded operators), both TPTL and MTL are expressively complete with respect of the first-order fragment of $\mathsf{S1S}_\mathsf{t}$, just like LTL is expressively complete with respect to S1S. This concludes our brief overview of Linear Temporal Logic for this chapter.

A precondition for the description of the algorithms proposed in this thesis is an overview of the classical approaches for solving the satisfiability and the realizability problems for LTL specifications.

1.4 The Classical Algorithms

In this section, we take an overview of classical techniques and algorithms for solving the satisfiability and the realizability problems of specifications belonging to LTL+P or to known fragments of LTL+P.

1.4.1 Algorithms for LTL+P Satisfiability

Previously, we have already seen that *satisfiability* is the problem of checking whether a formula admits at least one model, and that it is among the first theoretical questions that are answered for a particular logic. For LTL+P formulas, the problem is PSPACE-complete [179], meaning that any Turing machine solving the problem requires, in the worst case, a polynomial amount of space and exponential time with respect to the size of the input.

Satisfiability and Formal Verification

The satisfiability problem of LTL+P specifications has an important application to formal verification. Over the years, *model checking*, *i.e.*, the problem of deciding whether a system model satisfies a given specification for all its computations, has become very popular and effective in the verification of large systems [60]. However, the specification must be written with care: checking the model against a valid formula (*i.e.*, a formula which is trivially true in all structures)

is useless at the least and it could be severely harmful at worst. In fact, model checking against valid formulas always returns a positive answer that may convince the designers of the safety of the system, while instead some severe bugs may be present in it. More generally, the satisfiability of LTL+P formulas plays a central role in *property-based design* [156] and requirement analysis [23].

Classical Algorithms

Among the different kinds of techniques that has been proposed for satisfiability, *tableau methods* were one of the first to be investigated [62]. Originally devised for propositional logic [14], and then adapted to many other logics [62], tableau methods usually represent the easiest-to-understand and the easiest-to-design decision procedures for the satisfiability of a logic. LTL+P is no exception.

For the case of LTL+P, most of the tableau systems are *graph-shaped* and *two-pass*, meaning that a graph structure is first created (first pass), representing the set of all candidate models, and then traversed (second pass) looking for a correct model (if any) or discarding all the wrong candidate models, proving in this way the unsatisfiability of the formula. This is the case for most known tableau systems for LTL+P [141, 135, 200]. Incremental variants of this systems that build only the portions of the graph that are actually needed for the search have been proposed as well [122].

Another successful approach for solving LTL+P satisfiability is to reduce the problem to LTL+P model checking [165, 166]. This tecnique works as follows. Firstly a graph structure, very similar to the one described above, is built for the *negation* of the initial formula. In a second step, by means of a model checking algorithm, all paths of this structure are checked against a property witnessing the existence of a path that visits some given states infinitely many times. If model checking returns a positive result, this means that the negation of the formula is valid, and thus the initial formula is *unsatisfiable*. Otherwise, that is, if a counter example is found in the graph structure, it means that the negation of the formula is not valid, that is the initial formula is *satisfiable*.

The one-pass and tree-shaped tableau

Going back to propositional logic, its usual tableau system is *tree-shaped*, meaning that a tree structure is created for the search of

a model [180]: branches are created for assigning different values (true or false) to a propositional atom, and complete branches are total assigments to the set of propositional atoms of the formula. Differently from the propositional case, models of LTL+P formulas are *infinite* state sequences, and thus the graph structure of common tableau systems for LTL+P derives from the fact that an infinite path can be represented by a finite path ending with a loop-back to one of its previous states.

Recently, there has been some interest in proposing tree-shaped tableau systems, like the one for propositional logic, for LTL+P as well. The first tableau of this type was proposed by Schwendimann [172], which described a procedure that creates a tree representing the candidate models and uses some conditions to terminate even without building the full tree. We will call this last feature *one-pass*, as opposed to the classical tableau described above which requires two passes, one for building the graph and the other for traversing it.

The second of such tree-shaped tableau systems was proposed by Reynolds [163]. Like Schwendimann's tableau, it is tree-shaped and one-pass. However, it has the distinctive feature of producing a tree whose branches are totally *independent* from the others. This is different from the system by Schwendimann, which requires to keep track of multiple branches in the search for a model. This feature of Reynolds' tableaux has been possible thanks to the introduction of a new rule, called *prune* rule, which recognizes candidate models that are doing redundant work but are not fulfulling all the necessary requests for being a model. The smaller size of the tree with regards to the full graph structure of previous methods, and its simple rule-based tree search mechanism, led to an efficient implementation [12], a simple parallel version of the algorithm [145], and modular extensions to more expressive logics [100, 106]. As we shall see, we will give a new algorithm for solving LTL+P satisfiability based on Reynold's tableau system, exploiting, among other things, the fact that each branch is independent from the others.

1.4.2 Algorithms for LTL+P Realizability

In this part, we take a brief look on classical algorithms for solving realizability from LTL+P specifications. We recall that LTL+P realizability is the problem of establishing, given an LTL+P formula

over a set of controllable and uncontrollable variables, whether there exists a controller (*i.e.*, a strategy) choosing the value of the controllable variables in such a way that, no matter what value the environment chooses for the uncontrollable variables, the LTL+P formula is satisfied by all computations of the controller. The *reactive synthesis* problem refers to the problem of really building such a controller, if it exists. LTL+P realizability is 2EXPTIME-complete, meaning that any Turing machine for the problem having an input tape of length n terminates with the correct answer after 2^{2^n} steps, in the worst case.

Realizability and Formal Verification

Realizability has a crucial importance in model-based design and formal verification. We have already seen that the original motivation of Church for the synthesis problem was to allow for correct-by-construction systems [44]. The advantages deriving from such an approach to design are clear: the designers are freed from the logic control aspects of the system and from its implementation details, and they can instead direct all the effort on the quality of the specification. This is why synthesis and realizability are always referred to as the culmination of declarative programming [201]. Nowadays, this problem has been used in a variety of practical scenarios. One of the most famous is the synthesis of a controller starting from a complete set of requirements for the IBM AMBA bus [25, 24].

Even when the real construction of the system is skipped, that is when we are talking about *realizability* of a specification, the problem is crucial in property-based design and requirement analysis [21, 23]. In fact, before taking the design as the object of the verification, one should verify the set of specifications, for example asking whether it is consistent in the first place (satifiability) or whether it is *implementable* (realizability). This helps producing high-quality requirements, that can be used in later stages of the design process, not only for the verification of designs but also for communicating intents between different designers teams. High-quality specifications have proved to be fundamental, since industrial data revealed that nearly 50% of bugs was due to flaws in requirements and about 80% of the effort for rework can be traced back to requirement defects [21, 130].

Classical Algorithms

We take a look now to classical approaches for solving LTL+P realizability. The classical approach is usually identified with the one proposed by Pnueli and Rosner [160].

Given an LTL+P formula ϕ, the classical algorithm constructs in the first place the Büchi automaton recognizing the same language of ϕ. In general, this automaton is nondeterministic, meaning that there is the possibility that from a state there are two edges labeled with the same letter leading to two different states. The algorithm then performs a *determinization* of the automaton. Since Büchi automata are not closed under determinization, the resulting deterministic automaton has another type of acceptance condition, called Rabin condition, on which we will not go into the details. This determinization is typically done using *Safra's algorithm* [167]. Moreover, since a strategy (*i.e.*, what we are looking for) is a *tree* rather than a string, the automaton is built in a way to no longer reading words but rather trees.

The deterministic automaton is then viewed as an arena for a two-player game between the Controller player, who tries to build a winning strategy, and the Environment player, who tries to violate the specification. This last step consists in checking the *emptiness* of the Rabin automaton, *i.e.*, checking whether its language of trees is or is not empty: if it is empty, then the specification is unrealizable, otherwise there exists a tree in the language which is a strategy implementing the original specification, and thus the original formula is realizable. We remark that determinism is crucial for the game solving process; in fact, all the algorithms for checking the emptiness of a Rabin automaton work with a deterministic representation of the automaton.

Safraless approaches and Bounded Synthesis

When reactive synthesis started to become a practical task, it was soon clear that the bottleneck of the previous approach is Safra's algorithm, that is the determinization of a Büchi automaton into a Rabin automaton. The bottleneck is due to at least these three reasons: (i) it is arguably very complex: it deals with tree structures whose nodes are again trees; (ii) it is difficult to implement, witness the fact that its first implementation [118] has seen light more than twenty years after the original paper by Safra [167]; (iii) it is

definetely not amenable to optimizations.

One of the pioneering works devoted to avoiding Safra's construction is the work done by Kupferman and Vardi introducing the so-called *Safraless decision procedures* [128]. In that paper, they propose a framework for circumventing the use of Safra's algorithm. For example, in the context of realizability of LTL+P specifications, they first build the corresponding Büchi automaton (like in the classical approach), but, instead determinizing it into a Rabin one, they build the equivalent Universal co-Büchi automaton. Without entering into the details, universal co-Büchi automata are duals of Büchi automata, meaning that a language is recognized by an automaton of the former type if and only if its complement language is recognized by an automaton of the latter type. This means that, like a word is accepted by a Büchi automaton when it visits infinitely many times *an accepting state* while reading that word, a Universal co-Büchi automaton accepts a word when it visits only *finitely* many times *all rejecting states*.

Most importantly, the number of visits to a rejecting state can be bounded by an exponential function in the size of the automaton. This clever observation lead Kupferman and Vardi to observe that, by bounding this number with some *constant*, one obtains an automaton over *finite words*, *i.e.*, an NFA, that can be made deterministic by means of the classical *subset construction* [115], a much simpler and efficient algorithm than Safra's one. This implies that the emptiness problem of any Universal co-Büchi automaton can be approximated by a sequence of emptiness checks to each Deterministic Finite Automaton (DFA) obtained by bounding the visits to rejecting states and by applying subset construction. This approximation will eventually lead to a complete emptiness check. This is an example of Safraless decision procedure, which solves LTL+P realizability while in fact avoiding Safra's determinization algorithm

Bounded Synthesis [92] is another example of Safraless approach. It was introduced in 2007 by Schewe and Finkbeiner as a way of using the Safraless approach to reactive synthesis for distributed architectures, which is undecidable in the general case. As Safraless synthesis, it builds the Universal co-Büchi automaton corresponding to the starting LTL+P formula. However, instead of bounding directly the number of visits to rejecting states and building the sequence of DFAs obtained in that way, bounded synthesis encodes, by means of a constraint system, the existence of a strategy forcing the

automaton to visit n times its rejecting states, for each value of n between 0 and a computable uppperbound. Bounded synthesis has revealed to be very effective in practice, thanks in particular to the many different possible encodings for it, for example using Boolean formulas, or quantified Boolean formulas, or even dependency quantified Boolean formulas [88].

Realizability of Safety-LTL

Like Safraless techniques, there is another case when the reduction to the finite words case leads to fast and efficient algorithms. The temporal logic of Safety-LTL is defined as the logic obtained from LTL (thus with only future temporal operators) by avoding existential temporal operators, that is the *until* U and the *eventually* F [201]. It is known that Safety-LTL can express only *safety properties* [178]. Interestingly, it holds also the converse: each safety property definable in LTL is definable in Safety-LTL as well [40].

In [201], Zhu *et al.* study the reactive synthesis problem from Safety-LTL specifications. By definition of safety property, for each Safety-LTL formula, there exists an NFA (thus an automaton over *finite* words) recognizing the same language of the *negation* of the formula, and it can be then made deterministic by the classical subset construction. In [201], the determinization is performed by an external tool, MONA [113], which is very optimized for performing manipulation of automata. The game is then solved over the DFA by playing the classical reachability game [68]: if Controller can prevent the game to ever reach a final state, then there exists a winning strategy and the original Safety-LTL formula is realizable, otherwise, that is in the case the Environment player can force the game to reach a finale state of the automaton, then the specification is unrealizable. Overall, the result is an efficient algorithm for solving realizability and reactive synthesis from Safety-LTL specifications.

1.4.3 Symbolic Algorithms

Before 1993, there was a major problem in almost all the fields related to formal verification, and in particular in model checking. The size of the graphs, being them either systems' models or automata, was limited by the memory of the computers used to represent them. This was mainly due to the fact that the graphs were represented

explicitly, meaning that each node corresponded to a location of the memory and edges were represented as pointers. The verification of *concurrent* systems posed to the formal verification community the challenging problem of verying models with more than 10^{20} states. In fact, the number of states of a model corresponding to the synchronous product of n submodules is exponential in n, in the worst case: this is known as the *state-space explosion problem* [60]. Some techniques, like *partial order reduction* [155, 103], were proposed, but they were not sufficient to deal with very large systems in the general case.

Symbolic representation

The 1993 year knew a revolution in model checking, in the particular case, and in formal verification, in the general case, called *Symbolic Model Checking* [37, 146]. The necessity of verifying concurrent systems of growing size called for a new way for representing huge graphs, no more in an explicit way, but rather *symbolically*, by means of Boolean formulas.

The intuition behind symbolic model checking is to succinctly represent *sets of states* by means of Boolean formulas. In particular, the symbolic representation makes each state of a graph be in correspondence with an assignment of a set of Boolean variables, and it exploits the fact that there are 2^m possible assignments to m variables in order to ensure succinctness. Therefore, it should be clear that, given a graph with n states, it suffices to use $\log_2(n)$ variables for representing all states.

As states of a graph are encoded by means of Boolean variables, the symbolic representation encodes the set of edges of a graph with a Boolean formula. Consider a graph $G = (S, E)$ with n states and let $V = \{v_1, \ldots, v_m\}$ be the corresponding set of Boolean variables, with $m = \log_2(n)$. We define V' as the set $\{v'_1, \ldots, v'_m\}$. For each edge $e = (s_1, s_2)$ of the graph, we will use v for denoting the value of the variable v in state s_1, while v' denotes the same variable but for state s_2. The set of edges E is encoded into a Boolean formula ϕ_E over the set of variables $V \cup V'$. Each model of ϕ_E is an assignment to the variables $\{v_1, \ldots, v_m, v'_1, \ldots, v'_m\}$ that in turn encodes an edge of E. It is clear that, using Boolean formulas, a graph can be encoded logarithmically. Consider for example the simple graph depicted in Fig. 1.3, modeling a modulo-4 counter. For sake of clarity, the nodes

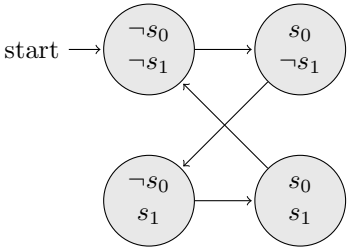


Figure 1.3: A modulo-4 counter.

of the graph contain also the value for the 2 variables s_0 and s_1 that are necessary for representing its states: since there are only 4 states, it is sufficient to use $\log_2(4) = 2$ Boolean variables. The graph can be succinctly represented by the following Boolean formula:

$$(s'_0 \leftrightarrow \neg s_0) \wedge (s'_1 \leftrightarrow ((s_0 \wedge \neg s_1) \vee (\neg s_0 \wedge s_1)))$$

Similarly, its initial state can be represented by the following Boolean formula:

$$\neg s_0 \wedge \neg s_1$$

Binary Decision Diagrams

Binary Decision Diagrams (BDDs, for short) are a canonical representation of Boolean formulas [33, 3]. They represent in a very compact way (logarithmic in the average case) all the models of a Boolean formula by means of a tree structure (very similar in some sense to the classical tableau for propositional logic) and by using a set of clever operations for reducing its number of edges and nodes, obtaining in the end a graph representing the set of models of the starting Boolean formula. By using BDDs, symbolic model checking can successfully verify systems with more than 10^{120} states, which is a hardly imaginable number, considered that the number of atoms in the universe is estimated to be around 10^{80}.

SAT-solvers

The use of propositional logic for representing graphs and the recognition that a wide range of combinatorial problems can be formalized

by means of it gave new life to effective procedures for checking the *satisfiability* of Boolean formulas [20], called the *SAT problem*. SAT was the first problem to be proved being NP-complete [61, 121]. Nevertheless, despite no polynomial algorithms exist for SAT so far, a lot of efficient techniques have been proposed in the literature that solve the SAT problem very effectively in practice. The literature of this field of research is so vast that it would be impossible to give even a comprehensive summary of all the results obtained in the last decades. We recap here below two of the major breakthroughs.

Starting from the classical tableau system for propositional logic [14, 62], which tooks exponential space (and thus at least exponential time) in the worst case, research moved to more space- and time-efficient algorithms for SAT. One breakthrough was marked by the DPLL algorithm [65], a space-efficient version of the previous algorithm DP [66], which uses *backtrack search*, *resolution methods* and *unit propagation* for an efficient decision procedure. We conclude with *Conflict-Driven Clause Learning* (CDCL), one of the other major breakthroughs in this field [176, 144]. Roughly speaking, when a conflict is found during the exploration of the state space of all possible assignments, this technique learns the partial assignment that led to the contradiction and uses its negation in the rest of the search, in order to shrink the state space.

Bounded Model Checking

We have previously seen that a graph can be represented logarithmically by means of Boolean formulas, and that, in turn, Boolean formulas can be stored succinctly using BDDs. We have also seen that this succinct representation is used by some algorithms of symbolic model checking in order to tackle the state-space explosion problem. However, a known drawback of BDDs is that there exist cases of Boolean formulas for which the corresponding BDD occupies an exponential amount of memory with respect to the size of the formula, making in fact useless the use of binary decision diagrams.

In order to overcome the limitations of BDDs, the technique of *Bounded Model Checking* (BMC) was introduced in 1999 [17, 16]. The main rationale of BMC consists in laveraging the great progress that SAT-solvers have had in the last decades. Let us recall the main concepts underlying BMC. First of all, BMC considers the negation $\neg\phi$ of the formula ϕ we want to model check, and it looks for a

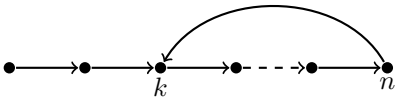


Figure 1.4: A finite path representing an infinite one.

path in the system's model satisfying $\neg\phi$: if it exists, then it returns *false* along with the counterexample, otherwise it returns *true*. BMC encodes the previous problem into a Boolean formula. In order to do that: (i) it considers paths of incremental length, say n, ranging from 0 to the diameter of the graph; (ii) starting from the Boolean formula encoding the transition relation of the system's model, it encodes the existence of a path of length n satisfying $\neg\phi$ into a Boolean formula; (iii) it uses a SAT solver to check the satisfiability of that formula: if it is satisfiable, then a counterexample (of length n) for ϕ has been found; otherwise it increments n and starts again.

Despite considering only finite paths, BMC can still work for full LTL, which is interpreted over infinite state sequence. In fact, a finite path can still represent an infinite one if it contains a loop-back from its final state to one of its previous state. For example, Fig. 5.3 shows a finite path representing the infinite path that coincides with the finite one in the first n states, and then loops forever between the state n and the state k.

If the formula ϕ is true in the system's model, a counterexample does not exists and the BMC algorithm continues to increment the length of paths until reaching the *diameter* of the graph, witnessing the exploration of *all paths*. This is referred to as the *completeness* of the algorithm. However, the computation of the diameter of a graph starting from a symbolic representation of the latter can be cumbersome in the general case, since it requires to solve the satisfiability of a quantified first-order formula. For this reason, BMC is usually used as a *bug-finder*, that is for looking for counterexamples only, rather than for *proving* that a property holds in a system. In some sense, *completeness* is sacrificed for sake of *efficiency*.

Symbolic LTL+P Satisfiability

Symbolic algorithms have been proposed also for the satisfiability of LTL+P. Since LTL+P satisfiability can be reduced to LTL+P model

checking, all algorithms for symbolic model checking can be used for satisfiability as well, and they proved to be quiet effective.

An interesting use of SAT-solvers is shown by Li *et al.* [134]. The SAT-solver is used as a generator of all the models of a propositional formula. These models are then used as states of an explicitly represented tableau which is subject to a series of heuristics for state elimination.

Symbolic LTL+P Realizability

Bounded synthesis [92] is an example of a (semi-)symbolic technique for LTL+P realizability. In fact in bounded synthesis, given the Universal co-Büchi automaton for the starting LTL+P formula ϕ, and a number n, the existence of a strategy that forces the game to visit the rejecting states at most n times is encoded into a constraint system. In the original paper [92], the encoding is done in first-order logic modulo finite integer arithmetic (with uninterpreted functions). Other encodings have been proposed for bounded synthesis, for example using: (i) binary decision diagrams [81]; (ii) Boolean formulas, quantified Boolean formulas, and dependency quantified Boolean formulas [88]. Even if the existence of a strategy is symbolically encoded into a constraint system, the Universal co-Büchi automaton is always represented explicitly. Since there is an exponential blow-up in passing from LTL+P formulas to equivalent Universal co-Büchi automata, there are cases in which the latter is prohibitively large to be stored on memory explicitly. To the best of our knowledge, there are not present in the literature techniques for constructing a symbolic Universal co-Büchi automata directly from an LTL+P specification.

We already mentioned that Safety-LTL is the fragment of LTL (with only future operators) devoid of the *until* and the *eventually* operators, and that the languages definable in Safety-LTL are exactly the safety properties definable in LTL+P. We have also seen that, since it is a safety fragment, for each Safety-LTL formula ϕ it is possible to construct a DFA (an automaton over *finite* words) recognizing the complement language, that is the language of $\neg\phi$. In [201], the authors build such an automaton using a semi-symbolic representation; starting from the negation of the formula, they use the MONA tool [113] for building the equivalent DFA whose nodes are explicitly represented but, for each pair of nodes, the set of edges between such

two nodes and their labels is stored using a BDD, thus symbolically. Usually, this type of representation is called *semi-symbolic*. Nevertheless, since the DFA corresponding to a Safety-LTL formula can be in general doubly exponential in the size of the formula, there are cases in which the DFA can't be generated due to its size and due to the fact that states are still explicitly represented.

Satisfiability Modulo Theory and Real-time Properties

Previously we have seen that sometimes, for a temporal specification language, it is important not only to be able to express the order between two events, for example *"each request is eventually followed by a response"*, but also the real time elapsed between two events, like for instance *"each request is eventually followed by a response in at most 3.0 time units"*, or the exact time at which an event occurs, like *"at time 5.0, the system must start up"*.

Satisfiability Modulo Theory (SMT, for short) is the problem of checking the satisfiability of formulas that are like Boolean formulas except from the fact that their atoms need not to be Boolean, but they can belong to an arbitrary theory. A non-exhaustive list of famous theories used in SMT is the following: (i) Linear Real Arithmetic (LRA); (ii) Linear Integer Arithmetic (LIA); (iii) Equality and Uninterpreted Functions (EUF); (iv) Bit vectors (BV); (v) Arrays (A). Usually, given a theory TH, the set of SMT formulas belonging to that theory is denoted as SMT(TH). An example of formula in SMT(LRA) is the following:

$$(x \leq 5.1) \rightarrow ((z + y > 7.2) \wedge (z - x \leq 10))$$

Similarly to the strong connection between discrete systems and Boolean logic that we have previously seen, there is also a strong connection between real-time systems and SMT formulas, in particular using the LRA theory. Therefore, a real-time system can be symbolically encoded using SMT(LRA) formulas for representing: (i) its discrete states (this is the Boolean part of SMT); and (ii) its real-time constraints (this is the LRA part). A consequence of this connection is that almost all symbolic algorithms that use SAT-solvers for solving model checking of discrete systems, including BMC, can be extended almost without effort for model checking real-time systems by using SMT-solvers under the theory of LRA instead of SAT-solvers.

1.5 Contributions

The contributions of this thesis can be partitioned into the following parts:

1. *Theory*: we introduce fragments of LTL+P and fragments of S1S, we characterize their expressive power, and we compare them with related formalisms already present in the literature.

2. *Problems and Algorithms*: we propose algorithms for the satisfiability problem from LTL+P specifications and algorithms for the realizability problem from specifications belonging to the fragments of LTL+P that we introduced in the previous part. In addition, at the end of this part, we define the compatibility problem from real-time specifications and show a symbolic algorithm for solving it.

In the following, we briefly recap the main contributions in each category.

1.5.1 Theory

The contributions related to this part consist in the introduction of three fragments of LTL+P and one fragment of S1S, the study of their expressive power and the comparison with related formalisms.

As a first contribution, we introduce the logic of *Extended Bounded Response* LTL+P ($\mathsf{LTL}_{\mathsf{EBR}}$+P, for short) as a particular fragment of LTL+P whose syntax is organized in layers. As we will see in the *Problems and Algorithms part*, this organization allows for a fully symbolic compilation of the formulas of this fragment into language-equivalent *deterministic symbolic* automata, that can be used for instance as *arenas* for a two-player game for solving realizability, without the need of additional determinization algorithms. In this part, we prove that, despite having a syntax which is arguably more difficult than other logics, $\mathsf{LTL}_{\mathsf{EBR}}$+P is *epressively complete* with respect to the safety fragment of LTL+P, meaning that:

a safety property can be defined in LTL+P *if and only if it can be defined in* $\mathsf{LTL}_{\mathsf{EBR}}$+P

In the proof of this result, a crucial role is played by the *past operators* of $\mathsf{LTL}_{\mathsf{EBR}}$+P. We compare the expressive power of $\mathsf{LTL}_{\mathsf{EBR}}$+P

with that of $\mathsf{LTL}_{\mathsf{EBR}}$, that is the logic defined as $\mathsf{LTL}_{\mathsf{EBR}}$+P devoid of past operators. Informally, we ask whether past operators are really necessary. We give a *positive* answer to this question, proving that $\mathsf{LTL}_{\mathsf{EBR}}$ is *strictly less expressive* than $\mathsf{LTL}_{\mathsf{EBR}}$+P. This is very interesting in our opinion, since it proves that past temporal operators, despite not being important for the expressive power of LTL+P (since LTL and LTL+P share the same expressiveness [97]), can play a crucial role for the expressive power of *fragments* of LTL+P, like for instance $\mathsf{LTL}_{\mathsf{EBR}}$+P.

As a second contribution of the *Theory part*, we introduce the logic of *Generalized Reactivity(1)* $\mathsf{LTL}_{\mathsf{EBR}}$+P (GR-EBR, for short), defined as an extension of $\mathsf{LTL}_{\mathsf{EBR}}$+P that goes beyond the safety fragment. In fact, in addition to safety properties, GR-EBR is able to express also: (i) assumptions and guarantees, in form of a logical implication between $\mathsf{LTL}_{\mathsf{EBR}}$+P formulas; (ii) recurrence formulas in both assumptions and guarantees, that is formulas of type $\mathsf{GF}\alpha$, expressing that a pure past formula α holds infinitely many times. We prove that the expressive power of GR-EBR stands between the expressive power of the Reactivity(1) and Generalized Reactivity(1) classes of the *Temporal Hierarchy* decribed in [140]. However, the exact expressive power of GR-EBR is not clear yet.

The third and last contribution of this part consists in the definition of a fragment of S1S[FO] (the first-order fragment of S1S) that captures exactly the safety fragment of LTL, meaning that any property definable in the first fragment is also definable in the latter, and *vice versa*. It follows that the fragment is expressively equivalent to $\mathsf{LTL}_{\mathsf{EBR}}$+P as well. This result joins Kamp's theorem that S1S[FO] and LTL are expressively equivalent, and it provides a direct, compact, and self-contained proof that any safety language definable in LTL is definable in Safety-LTL as well [40] (that is in LTL with only the X, G and the R operators), which seems not to be very much known, as the problem was presented as open as lately as 2021 [201, 71].[1]

1.5.2 Problems and Algorithms

In this part, (i) we give a symbolic encoding of the one-pass and tree-shaped tableau system for LTL+P; (ii) we give symbolic algo-

[1] As a matter of fact, we discovered about Chang *et al.* [40] after proving the same result in a different way.

rithms for solving realizability from $\mathsf{LTL}_{\mathsf{EBR}}$+P and GR-EBR specifications; (iii) we define the compatibility problem of real-time specifications and give symbolic algorithms for it.

A SAT-based encoding of the one-pass and tree-shaped tableau for LTL+P

In the context of LTL+P satisfiability, we introduce a symbolic SAT-based encoding of the one-pass and tree-shaped tableau system for LTL+P [163, 106]. The tableau tree is symbolically built in a breadth-first way, by means of Boolean formulas that encode all the tableau branches up to a given depth k, which is increased at every step. A set of Boolean formulas is then used for encoding the *termination rules* of the tableau system, *i.e.*, the rules allowing the tableau to terminate either by finding a model of the formula, or by proving its unsatisfiability. A SAT-solver is used for checking the satisfiability of all those Boolean formulas.

This breadth-first iterative deepening approach has been exploited in the past by bounded satisfiability checking and bounded model checking algorithms [17, 56, 112]. However, in contrast to those techniques, which require to compute the diameter of a graph or of a tableau in order to terminate even for unsatisfiable instances (*completeness*), the encoding that we propose guarantees completeness and termination both for satisfiable and unsatisfiable instances, without the need to precompute unsatisfiability thresholds, thanks to the encoding of the *prune rule* of the original tableau system.

The proposed procedure has been implemented in a tool called BLACK[2], which stands for Bounded Ltl sAtisfiability ChecKer. BLACK has been designed and implemented following the principles of speed, flexibility and reliability. For example,

- (*speed*) we use state-of-the-art SAT-solvers as backends for checking the satisfiability of Boolean formulas;
- (*flexibility*) BLACK contains a layer of abstraction that allows the seamless integration of new SAT-solvers; moreover, it can be extended very easily to support satisfiability of different logics, for example LTL interpreted over finite models;
- (*reliability*) its code coverage by unit tests is 100%.

[2] BLACK can be downloaded from `https://github.com/black-sat/black`

We run some experimental evaluation between BLACK and other tools. The results show that BLACK is competitive with them for many classes of formulas in the benchmarks' set.

Symbolic algorithms for $\mathsf{LTL_{EBR}}$+P and GR-EBR realizability

Previously we have seen that the realizability problem from LTL+P specifications is 2EXPTIME-complete and that, for the vast majority of cases, an explicit or a semi-symbolic automaton is built. When such an automaton is too big to be stored in memory, those techniques fail in solving the problem. A successful line of research focused on finding *fragments* of LTL+P for which there exists an efficient realizability problem. In this context, we propose symbolic and efficient algorithms for the realizability problem from $\mathsf{LTL_{EBR}}$+P and GR-EBR specifications.

The algorithm for $\mathsf{LTL_{EBR}}$+P realizability compiles any $\mathsf{LTL_{EBR}}$+P formula into a language-equivalent symbolic and deterministic safety automaton. Since it is deterministic, any algorithm for safety synthesis [26, 117] can be used as black-box for solving the two-player game played over the arena represented by the automaton. In addition, we show that this algorithm runs in singly exponential time and that it is *optimal*: in fact, we prove that $\mathsf{LTL_{EBR}}$+P realizability is EXPTIME-complete.

We implemented this algorithm in a prototype tool called EBR-LTL-SYNTH[3] and we conducted some experimental evaluation. The results show that our symbolic algorithm can in fact avoid an exponential blow-up in time which is common to all other tools. Some of these tools also fail at producing the explicit-state automaton due to space limits, while our algorithm successfully produce a symbolic automaton which can be stored in memory. This shows the advantages of symbolic algorithms for realizability in constrast to standard explicit-state ones.

We propose a fully symbolic algorithm for solving realizability starting from GR-EBR specifications, thus allowing the realizability of properties beyond the safety fragment. Since GR-EBR is syntactically an extension of $\mathsf{LTL_{EBR}}$+P, the algorithm first builds the symbolic automata for the $\mathsf{LTL_{EBR}}$+P parts, using the techniques we proposed for the $\mathsf{LTL_{EBR}}$+P logic. Then it approximates the

[3] EBR-LTL-SYNTH can be downloaded from `http://users.dimi.uniud.it/~luca.geatti/tools/ebrltlsynth.html`

language of the starting formula by performing a *safety reduction*, in order to reduce formulas of type $\mathsf{GF}\alpha$ (that are not safety) to safety properties. The completeness of the procedure is proved by introducing a general framework for guaranteeing completeness of arbitrary fragments of LTL and then by instantiating it for the case of GR-EBR. Interestingly, our framework proves that if a reduction is complete for the model checking problem, then it is also complete for the realizability problem. Moreover, the framework can easily be used for proving the completeness of bounded synthesis [92], a well-known approach for LTL realizability.

We implemented this procedure in a prototype tool called GRACE[4]. Our experimental evaluation witnesses the importance of using symbolic techniques with respect to explicit-state ones also in this case.

Compatibility checking of real-time requirements

In early phases of the workflow of model-based design, system engineers typically have only a set of requirements for the system they want to design: this is at the core of model-based design, since, in the first phases, requirements do not require any implementation, and any error in the requirements costs very few compared to similar errors in implementations.

Complex cyber-physical systems often have initialization procedures that require them to reach a target phase in a given real-time interval. In order for the system as a whole to reach its target, all of its subsystems have to reach their targets as well, possibly going through a number of intermediate phases, each within their own time interval. Moreover, subsystems are tipically dependent to each other: the entering of a component into phase i may be dependent to another component entering its phase j. For instance, the system shown in Fig. 1.5 is made of three subsystems/components, each one with its own set of phases, time intervals and dependencies. For example, component C must transition from phase Off to phase Normal in at least 2 and at most 3 times units, and, in doing that, it has to make sure that component E is in phase Normal.

As already mentioned, this type of requirements arise very often when designing initialization procedures of complex cyber-physical systems. Take into consideration for example a power station; the

[4] GRACE can be downloaded from `http://users.dimi.uniud.it/~luca.geatti/tools/grace.html`

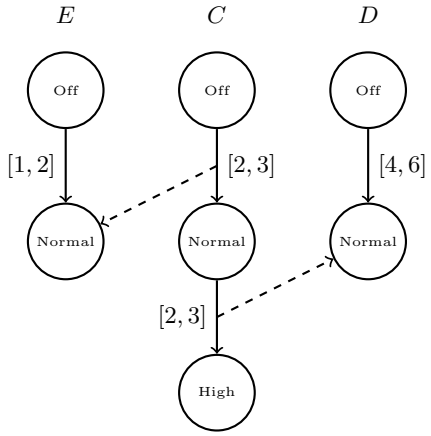


Figure 1.5: An example of a system made of three subsystems, each one with its own phases, time intervals and dependencies.

global system may be made of a power-generation component, one component for the cooling and one for the fuel supply. Suppose that the power-generation subsystem has a time interval in which it must transition from low-power mode to high-power mode, but, in order to do that, it has to make sure that the cooling and the fuel supply components are in their high-output mode as well. In turn, the cooling and the fuel supply components themselves have a time interval in which they must transition from low-output mode to high-output mode.

For such systems, engineers typically write a set of *real-time requirements*. In particular, we have a *global requirement* (or *system requirement*) asking the global system to reach the end of its initialization procedure in at least a and at most b time units, and a set of *local requirements*, one for each subsystem/component, asking each component i to reach its own target in at least a_i and b_i time units by (i) going through their sequence of phases; and (ii) fulfilling the dependencies among the other subsystems/components. This situation is depicted in Fig. 1.6.

For complex systems, local requirements are typically outsourced to different companies (sometimes, those companies have not the possibility to communicate with each other). Each company can implement the requirement assigned to it in different ways, according

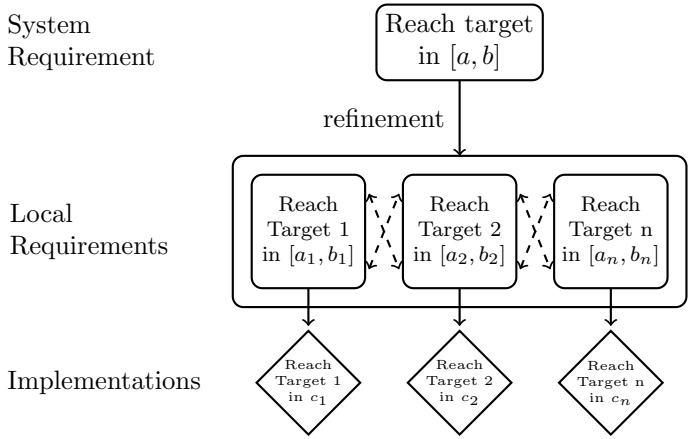


Figure 1.6: Compatibility checking problem.

that the implementation is compliant to the requirement. So, in the general case, there are several possible implementations for a single local requirement.

However, there may be *integration errors*, that is implementations such that, even though being compliant with the corresponding local requirement, when *composed* together, they form a system that does not satisfy the global requirement. In the example above, it may be the case that the company assigned to the implementation of the power-supply component implements it in such a way that the transition from low-output mode to high-output is performed too early with respect to the time at which the implementation of the cooling subsystem (which the power-supply component depends on) performs its own transition from low to high, even though both implementations comply with their own local requirements. This constitutes an *integration error*. Therefore, given a global real-time requirement and a set of local real-time requirements, we want to check if there are not integration errors among them, *i.e.*, checking whether it is possible to choose *any* possible implementation of each local requirement in such a way to guarantee the fulfillment of the global requirement when composing together such implementations. If this is the case, we say that the set of requirements is *compatible*. We call this problem *compatibility checking*. This is the main problem we investigate in this part.

Sometimes, it happens that there is *no* way of implementing the set of local requirements without avoiding integration errors. In some sense, the set of local requirements is *inconsistent* with respect to the global requirement. We also investigate the problem of checking if this is the case, and we call this problem the *consistency checking* problem.

Finally, if the set of local requirements is consistent but not compatible, this means that the bounds on the time intervals inside the requirements are not strict enough to avoid integration errors for any implementation. Therefore, we study also the problem of synthesizing the set of *compatible refinements* of the original local requirements, that is the requirements obtained by shrinking the original bounds in order to avoid integration errors. We call this problem *synthesis of compatible refinements*.

We propose the use of symbolic techniques for the *compatibility*, the *consistency* and the *compatible refinements' synthesis* problems of real-time requirements. Our main technique is based on a direct encoding into SMT(LRA). We implemented the algorithm into a prototype tool called TRICKER[5]. We compared it with an encoding into model checking of real-time properties. The results of the experimental evaluation reveal better performance of the encoding into SMT(LRA) compared to the reduction to model checking.

1.6 Organization of the thesis

The rest of the thesis is organized in two parts, the first one dedicated to *Theory* and the other one to *Problems and Algorithms*.

The *Theory part* is structured as follows. We start with Chapter 2, which gives the necessary background for this part. In particular, it offers an overview of four intrinsically different formalisms for expressing properties of sequences, that have been thoroughly studied since the dawn of formal languages and automata theory. These formalisms are: (i) the first- and second-order theories of 1 successor; (ii) automata over finite and infinite words; (iii) regular and ω-regular expressions; (iv) linear temporal logics. For all of them, we give the basic definitions and the fundamental theorems, comparing also different formalisms to each other. Particular attention

[5] TRICKER can be downloaded from `http://users.dimi.uniud.it/~luca.geatti/tools/tricker.html`

is devoted also to the classes of *safety* and *co-safety* properties.

We continue with Chapter 3, where we introduce three new fragments of Linear Temporal Logic with Past (LTL+P) that will be the object of our study also in the part dedicated to algorithms. In order of appearance:

- In Section 3.1, we introduce the logic of *Extended Bounded Response* LTL+P ($\mathsf{LTL}_{\mathsf{EBR}}$+P). We give its syntax and semantics, a normal form and some meaningful examples that can be formalized within its language. We then prove that $\mathsf{LTL}_{\mathsf{EBR}}$+P is *expressively complete* with respect to the safety fragment of LTL+P.

- We define the $\mathsf{LTL}_{\mathsf{EBR}}$ logic as $\mathsf{LTL}_{\mathsf{EBR}}$+P devoid of past operators in Section 3.2; the main result of this section is a proof that $\mathsf{LTL}_{\mathsf{EBR}}$ is *strictly less expressive* than $\mathsf{LTL}_{\mathsf{EBR}}$+P, thus proving the importance of past operators in the syntax of $\mathsf{LTL}_{\mathsf{EBR}}$+P.

- Section 3.3 introduces the logic of GR-EBR (*Generalized Reativity(1)* $\mathsf{LTL}_{\mathsf{EBR}}$+P) as an extension of $\mathsf{LTL}_{\mathsf{EBR}}$+P able to express properties beyond the safety fragment. We show an example on how GR-EBR can be used to formalized a simple arbiter and we characterize its expressive power by comparing it with the *temporal hierarchy* [140].

We conclude the *Theory part* with Chapter 4, in which we introduce a novel fragment of the first-order theory of 1 successor and prove that this fragments captures exactly the set of safety properties definable in LTL. As a by-product, we obtain a new proof of the fact that Safety-LTL is expressively complete with respect to the safety fragment of LTL. This concludes the *Theory part*.

In the *Problems and Algorithms part* we describe our symbolic algorithms for solving the *satisfiability*, the *realizability* and the *compatibility* problems. We start with Chapter 5, where we give the necessary background for this part. For example, we describe in details the classical tableau system for LTL+P satisfiability [141, 135, 200] and also the one-pass and tree-shaped tableau system [163, 106]. We also recap the classical methods for solving LTL+P realizability.

In Chapter 6, we propose a SAT-based encoding of the one-pass and tree-shaped tableau system for LTL+P [163, 106]. We describe the details of the encoding and of the choices underlying the imple-

mentation. Finally we show the results of the experimental evaluation.

We continue with Chapter 7, where we focus on realizability of $\mathsf{LTL_{EBR}}$+P specifications. First of all, we introduce the algorithm for constructing a deterministic symbolic automaton corresponding to a formula of $\mathsf{LTL_{EBR}}$+P. The algorithm is made of a sequence of nontrivial steps: for each of them, we prove its correctness and give its worst-case complexity. We give also the theoretical complexity of the satisfiability and the realizability problems of $\mathsf{LTL_{EBR}}$+P specifications: from these results, we derive also the *optimality* of our algorithm. We conclude Chapter 7 showing the performance of our implementation with respect to state-of-the-art competitors.

In Chapter 8, we show how to extend the previous realizability algorithm from $\mathsf{LTL_{EBR}}$+P to GR-EBR specifications. We also introduce a framework for deriving *sound and complete* safety reductions in the context of LTL+P realizability. We then instantiate that framework for the specific case of the GR-EBR logic: this allows us to prove soundness and completeness of our algorithm. Finally, also in this case, we show the outcomes of the experimental evaluation.

We study the *compatibility* problem of real-time requirements in Chapter 9. We give the formal definition of the problem and, in addition, we show how it can be formalized in terms of *timed interface automata* [69, 67]. We then give the encoding of the problem (i) into SMT(LRA), and (ii) into model checking of real-time properties. Finally, we show the experimental results of our implementation.

We conclude the thesis with Chapter 10, in which we draw some conclusions and point out interesting future directions.

1.6.1 Publications

We give the references to the papers on which the results on this thesis have been published.

The expressive power of $\mathsf{LTL_{EBR}}$+P and $\mathsf{LTL_{EBR}}$ (Sections 3.1 and 3.2) has been published in [47]. We introduced the logic of GR-EBR (Section 3.3) as part of the paper in [45], while the considerations on its expressive power appear for the first time in this thesis.

The work on the BLACK tool and the underlying symbolic algorithm (Chapter 6) have been published in [99, 102]. The symbolic algorithm for $\mathsf{LTL_{EBR}}$+P realizability, its implementation and the

theoretical complexity of the problem (Chapter 7) are reported in the papers in [46]. The symbolic algorithm for GR-EBR realizability, its implementation and the framework for general safety reductions (Chapter 8) are part of the paper in [45]. Finally, the work about compatibility checking of timed requirements (Chapter 9) has been published in [48].

Part II

Theory

CHAPTER

2

BACKGROUND

Our objective in this chapter is to give a background theory for the rest of this part of the thesis. If one wanted to find a common point among all the key concepts that we used in this part, then this would be the notion of *formalism for expressing properties of sequences*, where, with the term *sequences*, we refer to finite or infinite linear orders. One of the most beautiful and elegant features of this field of study is that there is a plethora of intrinsically different formalisms os this type that were proved to be equivalent. In fact, in this chapter we will show the connection between the following formalisms for expressing properties about sequences: (i) Sequential Calculus, both in the second-order (S1S) and in the first-order (S1S[FO]) setting; (ii) Automata over finite and infinite words; (iii) Regular and ω-regular expressions; (iv) Linear Temporal Logics. Each of these formalisms can be interpreted either over finite or infinite sequences. For each of them, we will recap its main properties and the equivalence with the other formalisms, both in the case of finite and infinite sequences. Since the results for infinite sequences are an impressive mathematical achievement and constitute a milestone for the field of

formal languages, particular attention will be devoted to the infinite case.

2.1 Sequential Calculus

Sequential Calculus [44, 34] is a logical formalism for specifying properties of sequences (or, equivalently, linear orders). Formally, Sequential Calculus is the theory of monadic second-order restricted arithmetic, that is the theory with domain $\mathbb{N}$ (the set of natural numbers), the successor function $+1$ and the constant 0, allowing also quantifiers to be applied on sets. The first use of Sequential Calculus can be traced back to Church [44], who suggested to use it for specifying properties of sequential circuits (hence its name). In the following, we will give the definition of Sequential Calculus, showing its fundamental decidability properties, and, in the next sections, its connection with automata theory.

2.1.1 Definition of S1S

Let Σ be a finite set of symbols (or *letters*), called *alphabet*. We define an ω-word (resp. a word) $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ over Σ as any infinite (resp. finite) sequence of elements in Σ. We call Σ^ω (resp. Σ^*, where $*$ is the *Kleene'star*) the set of all the ω-words (resp. words) over Σ. Given an ω-word or a word σ, we call σ_i the i-th element of σ. For some $i, j \in \mathbb{N}$, with $i < j$, with $\sigma_{[i,j]}$ we refer to the interval $\langle \sigma_i, \ldots, \sigma_j \rangle$ of σ. Similarly, for some $i \in \mathbb{N}$, with $\sigma_{[i,\infty)}$ we refer to the suffix of σ starting from index i. For any $\sigma \in \Sigma^*$ and any $\sigma' \in \Sigma^* \cup \Sigma^\omega$, we denote with $\sigma \cdot \sigma'$ (or simply with $\sigma\sigma'$) the concatenation of σ' at the last position of σ. A set of ω-words is called ω-language. With $\overline{\mathcal{L}}$ we denote the *complement language*, defined as $\overline{\mathcal{L}} \coloneqq \{\sigma \in \Sigma^\omega \mid \sigma \notin \mathcal{L}\}$ (the same holds for languages of finite words).

In the context of sequential calculus, an ω-word over Σ is represented as a model-theoretic structure of type $(\mathbb{N}, 0, +1, <, \{Q_a\}_{a \in \Sigma})$, where:

- $\mathbb{N}$ is the domain of the interpretation and is the set of natural numbers,
- 0 is a constant symbol, interpreted as the number 0;

- $+1$ is a function symbol, interpreted as the successor function;
- $<$ is a binary relation symbol, interpreted as the *less than* relation;
- Q_a, for each $a \in \Sigma$, is a unary (or, equivalently, *monadic*) relation symbol, interpreted as the set of positions in which σ contains the letter a, that is, $Q_a := \{i \in \mathbb{N} \mid \sigma_i = a\}$, for each $a \in \Sigma$.

The model-theoretic structure for a *finite* word is a tuple $(D, 0, +1, <, \{Q_a\}_{a\in\Sigma})$, where the domain $D = \{1, \ldots, d\}$ is a *finite* set of natural numbers, and the rest of the symbols is interpreted as in the case for ω-words, except from the successor function $+1$ for the value d, that is defined as $d + 1 = d$.

Sequential calculus is defined as the second-order logic interpreted over structures of type $(\mathbb{N}, 0, +1, <, \{Q_a\}_{a\in\Sigma})$. For this reason, it is often called S1S, which stands for *the monadic Second-order theory of 1 Successor*. From now on, we will use S1S for denoting Sequential Calculus. The term *"second-order"* refers to the fact that quantifiers of this logics are not only applied to one-valued variables $(x, y, z, \ldots)$, hereinafter called classical variables, but, instead, they can be applied also to variables represeting sets $(X, Y, Z, \ldots)$ or n-ary relations. The term *"monadic"* refers to the fact that quantifications are only applied to classical variables and sets variables (hereinafter called *monadic variables*). We give now the formal definition of S1S under the infinite linear order interpretation.

Definition 1 (Syntax of S1S over ω-words [193])**.** *Let Σ be a finite alphabet. The* terms *of* S1S *are built from the constant 0 and the variables $x, y, \ldots$ using the successor function $+1$.* Atomic formulas *are of the form $t < t'$, $t = t'$, $t \in X$ and $t \in Q_a$ (for $a \in \Sigma$), where t and t' are terms, and X is a monadic variable.* S1S-formulas *are built from atomic formulas by means of the Boolean connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ and the quantifiers $\exists$ and $\forall$, on both classical variables and monadic variables. A (classical or monadic) variable x is called* free *if it does not appear in the scope of a quantifier applied to x. A* S1S*-formula ϕ is called a* sentence *if and only if it does not contain any free (classical or monadic) variable.*

Definition 2 (Semantics of S1S over ω-words [193])**.** *Given an ω-word $\sigma \in \Sigma^\omega$ represented by the structure $(\mathbb{N}, 0, +1, <, \{Q_a\}_{a\in\Sigma})$, a*

S1S *formula* $\phi(x_1, \ldots, x_n)$ *with* n *free variables (either classical or monadic), and* n *values* $v_1, \ldots, v_n$ *for the free variables (where either* $v_i \in \mathbb{N}$ *if* x_i *is a classical variable or* $v_i \subseteq \mathbb{N}$ *if* x_i *is monadic), we say that* σ *is a* model *of* ϕ*, written as* $\sigma, v_1, \ldots, v_n \models \phi$*, if and only if* $(\mathbb{N}, 0, +1, <, \{Q_a\}_{a\in\Sigma})$ *satisfies* ϕ*, when* x_i *is replaced with the value* v_i*, for each* $i \in \{1, \ldots, n\}$*. If* ϕ *is a sentence, then we will simply write* $\sigma \models \phi$ *and we define the* language of ϕ *as* $\mathcal{L}(\phi) = \{\sigma \in \Sigma^\omega \mid \sigma \models \phi\}$.

If we interpret an S1S-sentence over *finite* linear orders, then we write $\mathcal{L}^{<\omega}(\phi)$ to denote its language, that is, the set of finite words represented by a structure of type $(D, 0, +1, <, \{Q_a\}_{a\in\Sigma})$, for some finite domain D. We called the resulting formalism *Weak Sequential Calculus*, or equivalently *Weak* S1S. The satisfiability problem for S1S-sentence is the problem of determine, given a sentence ϕ of S1S, whether there exists a structure that satisfy ϕ. Under the infinite (resp. finite) linear order interpretation, the satisfiability problem can be reformulated as $\mathcal{L}(\phi) \stackrel{?}{=} \varnothing$ (resp. $\mathcal{L}^{<\omega}(\phi) \stackrel{?}{=} \varnothing$). The validity problem for S1S-sentence is the problem of determine, given a sentence ϕ of S1S, whether ϕ is satisfied by all the structures of type $(\mathbb{N}, 0, +1, <, \{Q_a\}_{a\in\Sigma})$ (where $\mathbb{N}$ is replace with a finite domain D in case of finite linear orders interpretation). Under the infinite (resp. finite) linear order interpretation, the validity problem can be reformulated as $\mathcal{L}(\phi) \stackrel{?}{=} \Sigma^\omega$ (resp. $\mathcal{L}^{<\omega}(\phi) \stackrel{?}{=} \Sigma^*$).

Note. Since the $<$ relation can be formalized only in terms of the constant 0 and the function $+1$ using the second-order quantifiers, one can restrict ω-words to be structures of type $(\mathbb{N}, 0, +1, \{Q_a\}_{a\in\Sigma})$. This is not true if we restrict S1S to contain only first-order quantifications, that is quantifiers applied only to classical variables. We call S1S[FO] the *first-order fragment of* S1S, obtained by forbidding second-order quantifications. Since the relation $<$ is necessary, S1S[FO] is interpreted over structures of type $(\mathbb{N}, 0, <, =, \{Q_a\}_{a\in\Sigma})$. The $+1$ function can be easily defined by a first-order formula as follows:

$$y = x + 1 \Leftrightarrow (x < y \wedge \neg\exists z \,.\, (x < z < y))$$

2.1.2 Properties of S1S

Let $\mathcal{L}$ be an ω-language. We say that $\mathcal{L}$ is S1S-definable if and only if there exists a S1S-formula ϕ such that $\mathcal{L} = \mathcal{L}(\phi)$. With some abuse of notation, with the symbol S1S, we will also refer to the set of formulas which can be syntactically expressed in the S1S logic.

The two seminal results that magnified the role of the logic S1S in computer science, proved by Büchi in 1960, establish that the satisfiability problem of S1S both on finite [35] and infinite words [34] is decidable. From now on, when the satisfiability of a logic $\mathbb{L}$ is decidable, we simply write that *the logic* $\mathbb{L}$ *is decidable.*

Theorem 1 (Decidability of S1S [34])**.** *The set of* S1S *sentences (over both finite and infinite words) is decidable.*

Büchi first proved the theorem for the case of finite words, and then he generalized the result for the infinite case. The core of both the decidability proofs is a strong correspondence between logical formulas and automata (which will be defined later). In particular, he showed how the satifiability problem of a S1S-formula ϕ over finite (resp. infinite) words can be reduced to the emptiness problem (we will define it later in this section) of an automaton over *finite* (rest. *infinite*) words built starting from ϕ. Since the emptiness problem over automata is decidable, S1S is decidable as well. Finally, it is worth noting that, while in 1960 automata over finite words were already a well-established formalism in computer science, automata on infinite words had not yet been studied. In order to solve the decidability problem of S1S, Büchi [34] introduced a novel type of automata accepting infinite words (from then on called Büchi automata), opening a research field that nowadays (more than 60 years later) is still very fertile.

As will see later, another path of research of fundamental importance in computer science is the one studying *temporal logics* [179], which have strong connections with the S1S logic, and, in particular, with S1S[FO]. As we will see, temporal logics offer an expressively equivalent variant of S1S and S1S[FO], but with lower complexities for the satisfiability and validity problems.

2.1.3 The Safety, co-Safety and Liveness classes

An important class of S1S-definable ω-languages comprises those expressing the fact that something "bad" never happens (like for

instance a deadlock, or a simultaneous access into a critical section by two different process). For this reason, they are called *safety languages* (or *safety properties*).

Definition 3 (Safety language [127])**.** *Let $\mathcal{L} \subseteq \Sigma^\omega$ be a* S1S*-definable ω-language. We say that $\mathcal{L}$ is a* safety language *if and only if for all the words $\sigma \in \Sigma^\omega$ it holds that, if $\sigma \notin \mathcal{L}$, then $\exists i \in \mathbb{N} \,.\, \forall \sigma' \in \Sigma^\omega \,.\, \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}$. We call such a $\sigma_{[0,i]}$ a* bad prefix *of $\mathcal{L}$, and we denote with $BadPref(\mathcal{L})$ the set of all the bad prefixes of $\mathcal{L}$. With* SAFETY*, we denote the set of all the* S1S*-definable safety ω-languages.*

Another important class is the one of *co-safety* languages. Co-safety languages contain the ω-words such that their acceptance can be witnessed by a *finite* prefix.

Definition 4 (Co-Safety language [127, 201])**.** *Let $\mathcal{L} \subseteq \Sigma^\omega$ be a* S1S*-definable ω-language. We say that $\mathcal{L}$ is a* co-safety language *if and only if for all the words $\sigma \in \Sigma^\omega$ it holds that, if $\sigma \in \mathcal{L}$, then $\exists i \in \mathbb{N} \,.\, \forall \sigma' \in \Sigma^\omega \,.\, \sigma_{[0,i]} \cdot \sigma' \in \mathcal{L}$. We call such a $\sigma_{[0,i]}$ a* good prefix *of $\mathcal{L}$, and we denote with $GoodPref(\mathcal{L})$ the set of all the bad prefixes of $\mathcal{L}$. With* coSAFETY*, we denote the set of all the* S1S*-definable co-safety ω-languages.*

Clearly, SAFETY and coSAFETY are dual classes, as stated by the following proposition.

Proposition 1. *Let $\mathcal{L}$ be an ω-language. It holds that:*

1. $\mathcal{L} \in$ SAFETY $\Leftrightarrow \overline{\mathcal{L}} \in$ coSAFETY*;*
2. *If $\mathcal{L} \in$ SAFETY (or $\mathcal{L} \in$ coSAFETY) then $BadPref(\mathcal{L}) = GoodPref(\overline{\mathcal{L}})$.*

Another important class of ω-languages is made of those languages that express the fact that something "good" (like termination of a program) will eventually happen. They are called *liveness languages* (or *liveness properties*). Another way of thinking to liveness languages is that for every prefix of a word, there is always a way to extend it in order to obtain a word in the language (*"a partial execution is never irremediable"* [4]). Ultimately, liveness languages define properties that talk about infinite behaviors, like the property *"there are infinitely many requests of a grant"*.

Definition 5 (Liveness language [178])**.** *Let $\mathcal{L} \subseteq \Sigma^\omega$. We say that $\mathcal{L}$ is a* liveness language *if and only if for all the words $\sigma \in \Sigma^*$ it holds that $\exists \sigma' \in \Sigma^\omega$ such that $\sigma \cdot \sigma' \in \mathcal{L}$. With* LIVENESS*, we denote the set of all the liveness languages.*

An alternative characterization of the LIVENESS class is the following one: an ω-word belongs to a liveness language if and only if the set of its finite prefixes is exactly the set Σ^*, that is the set of *all* the finite words made of elements in Σ.

2.1.4 A Notation for Syntax and Semantics

Let $\mathbb{L}$ be a logic interpreted over infinite or finite linear orders, like, for instance, S1S. From now on, with the symbol $\mathbb{L}$, we will also denote the set of all and only those formulas that syntactically belong to $\mathbb{L}$. In our context, it will be useful to consider not only the formulas that syntactically belong to a logic, but also the *languages* that can be defined in a logic. First of all, we use the following notation to distinguish the interpretation of a logic $\mathbb{L}$ over finite words from the interpretation over infinite words.

Notation 1 (Finite or infinite words interpretation)**.** *Let $\mathbb{L}$ be a logic interpreted over infinite or finite linear orders, and let $\phi \in \mathbb{L}$ be a formula that syntactically belong to this logic. With $\mathcal{L}^{<\omega}(\phi)$ we denote the language of ϕ under the finite linear order interpretation. With $\mathcal{L}(\phi)$ we denote the language of ϕ under the infinite linear order interpretation.*

To distinguish the syntax of a logic (*i.e.*, the set of its formulas) from its semantics (*i.e.*, the set of languages of its formulas), we use the following notation.

Notation 2 (Semantics of a logic over linear orders)**.** *Given a logic $\mathbb{L}$ interpreted over infinite (resp. finite) linear orders, we denote with $[\![\mathbb{L}]\!]$ (resp. $[\![\mathbb{L}]\!]^{<\omega}$) the set of all and only those languages $\mathcal{L}$ over infinite (resp. finite) words for which there exists a formula $\phi \in \mathbb{L}$ (*i.e.*, ϕ syntactically belongs to $\mathbb{L}$) such that $\mathcal{L} = \mathcal{L}(\phi)$ (resp. $\mathcal{L} = \mathcal{L}^{<\omega}(\phi)$). Given a set of languages of finite words $[\![\mathsf{L}]\!]^{<\omega}$, with some abuse of notation, we denote as $[\![\mathsf{L}]\!]^{<\omega} \cdot (2^\Sigma)^\omega$ the set of languages $[\![\mathsf{L}]\!]^{<\omega} \cdot (2^\Sigma)^\omega = \{\mathcal{L} \cdot (2^\Sigma)^\omega \mid \mathcal{L} \in [\![\mathsf{L}]\!]^{<\omega}\}$.*

In this thesis, we will mainly consider logics $\mathbb{L}$ that are (semantical) fragments of S1S, that is $[\![\mathbb{L}]\!] \subseteq [\![\mathsf{S1S}]\!]$ and $[\![\mathbb{L}]\!]^{<\omega} \subseteq [\![\mathsf{S1S}]\!]^{<\omega}$.

Later, we will study which languages can be defined in a logic, and which can not. For example, it often happens that a language is definable only in particular logic $\mathbb{L}$, while it is not in another logic $\mathbb{L}'$. We define the notion of $\mathbb{L}$-definability as follows.

Definition 6 (Definability for a logic over linear orders)**.** *From now on, given a logic $\mathbb{L}$ interpreted over infinite (resp. finite) linear orders and a language $\mathcal{L}$ of infinite (resp. finite) words, we say that $\mathcal{L}$ is* $\mathbb{L}$-definable*, or equivalently* $\mathbb{L}$-expressible*, over infinite (resp. finite) words if and only if $\mathcal{L} \in [\![\mathbb{L}]\!]$ (resp. $\mathcal{L} \in [\![\mathbb{L}]\!]^{<\omega}$).*

2.2 Automata over finite and infinite words

Automata are the formalism that more than the others stands at the basis of computer science and helped to develop its theoretical foundations. In fact, automata can be seen as a weaker form of Turing machines [115], in the sense that the problems solvable (or the *languages* recognized) by automata are solvable by Turing machines as well, but not viceversa.

2.2.1 Automata over Finite Words

The first type of automata to be studied are the ones reading *finite words*, that is words in Σ^*, for a given finite alphabet Σ.

Definition 7 (Finite Automata (NFA and DFA), [115])**.** *A* Nondeterministic Finite Automaton *(*NFA*, for short) is a tuple $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, where: (i) Σ is a finite alphabet; (ii) Q is a finite set of states; (iii) $I \subseteq Q$ is the set of* initial states*; (iv) $\delta \subseteq Q \times \Sigma \times Q$ is the* transition relation*, and (v) $F \subseteq Q$ is the set of* final states*. If δ is a* function *($\delta : Q \times \Sigma \to Q$), then $\mathcal{A}$ is called a* Deterministic Finite Automaton *(*DFA*, for short).*

Given a DFA with transition function $\delta : Q \times \Sigma \to Q$, we define the function $\delta^* : Q \times \Sigma^* \to Q$ as the generalization of δ for sequences of letters in Σ, defined inductively as follows:

$$\delta^*(q, \varepsilon) = q$$
$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

We say that NFAs and DFAs are *automata over finite words*. Each automaton over finite words recognizes a language of finite words $\mathcal{L} \subseteq \Sigma^*$.

Definition 8 (Run and Language of a NFA)**.** *Let $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ be a* NFA*, and let $\sigma = \langle \sigma_0, \sigma_1, \ldots, \sigma_n \rangle \in \Sigma^*$ be a finite word, for some $n \in \mathbb{N}$. A* run *of $\mathcal{A}$ over σ is a finite sequence of states $\tau = \langle q_0, q_1, \ldots, q_{n+1} \rangle$ such that $q_0 \in I$ and $(q_i, \sigma_i, q_{i+1}) \in \delta$, for $i = 0, \ldots, n$. We say that τ is* accepting *if and only if its last state is a final state, that is $q_{n+1} \in F$. A word $\sigma \in \Sigma^*$ is* accepted *by $\mathcal{A}$ if and only if there exist an accepting run of $\mathcal{A}$ over σ. The* language accepted *(or* recognized*) by $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is the set of all and only the words $\sigma \in \Sigma^*$ accepted by $\mathcal{A}$.*

The *product automaton* $\mathcal{A} \times \mathcal{A}'$ between the two NFAs $\mathcal{A}$ and $\mathcal{A}'$ is the automaton that recognizes exactly the intersection between the languages recognized by $\mathcal{A}$ and $\mathcal{A}'$, respectively, *i.e.*, $\mathcal{L}(\mathcal{A} \times \mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$, and it defined as follows.

Definition 9 (Product between NFAs)**.** *Let $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ and $\mathcal{A}' = (\Sigma, Q', I', \delta', F')$ be two* NFAs*. The product automaton $\mathcal{A} \times \mathcal{A}'$ between $\mathcal{A}$ and $\mathcal{A}'$ is the automaton $(\Sigma, Q'', I'', \delta'', F'')$ such that:*

- $Q'' = Q \times Q'$
- $I'' = \{(q, q') \mid q \in I, q' \in I'\}$
- $\delta'' = \{((q, q'), \sigma, (s, s')) \mid (q, \sigma, s) \in \delta, (q', \sigma, s') \in \delta')\}$
- $F'' = \{(q, q') \mid q \in F, q' \in F'\})$

Given a NFA $\mathcal{A}$, the *emptiness problem* is the problem of finding whether $\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \varnothing$. This problem is decidable in polynomial time (with respect to the number of states in the automaton) since it amounts to a reachability problem: $\mathcal{L}(\mathcal{A}) \neq \varnothing$ if and only if the graph representing the structure of the automaton contains a directed path from some initial state of $\mathcal{A}$ to some of its final states.

It is known that the set of languages recognized by some NFA is exactly the set of languages recognized by some DFA [115]. For this reason, we say that NFA and DFA have the same *expressive power*. The proof of this equivalence uses the well known *subset construction* (also known as *powerset construction*) in order to build a DFA $\mathcal{A}'$ starting from a NFA $\mathcal{A}$, in such a way that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Subset

construction consists in introducing a macro-state for each subset of states of $\mathcal{A}$. The states of $\mathcal{A}'$ are exactly those macro-states. A macro-state t is reachable with letter a in $\mathcal{A}'$ from a macro-state s if and only if all the original states in t are reachable with letter a in $\mathcal{A}$ by some state in the macro-state s. The final states of $\mathcal{A}'$ are the macro-states containing a final state of $\mathcal{A}$. In the worst case, subset construction may produce an exponential blowup in the number of states of $\mathcal{A}'$ with respect to the number of state of $\mathcal{A}$. We refer the reader to [115] for more details on the subset construction method.

In [149], McNaughton and Papert define the class of *counter-free* automata over finite words, by restricting the type of words accepted. In particular, the words accepted by a counter-free automaton are all and only the words containing a finite number of consecutive repetitions of a subword.

Definition 10 (Counter-free automata over finite words [149, 170])**.** *Let $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ be a* DFA*. We say that a finite word $w \in (2^{\Sigma})^+$ defines a* nontrivial cycle *in the automaton $\mathcal{A}$ if and only if there exists a state $q \in Q$ such that $\delta^*(q, w) \neq q$ and $\delta^*(q, w^i) = q$, for some $i > 1$, where $w^i = w_{(1)} \cdot \ldots \cdot w_{(i)}$. The automaton $\mathcal{A}$ is said* counter-free *if and ony if it does* not *contain any nontrivial cycle. We denote with* cf-DFA *the class of counter-free* DFAs*.*

Alternatively, we can define counter-free automata in this way. We say that $\mathcal{A}$ is *counter-free* if and only if there does not exists a sequence of states $q_1, q_2, \ldots, q_n \in Q$ (for some $n > 1$) and a word $\sigma \in \Sigma^*$ such that $q_{i+1} \in \delta^*(q_i, \sigma)$ for $i = 1, \ldots, n-1$ and $q_1 \in \delta^*(q_n, \sigma)$. Later in this chapter, we will give the strong relation between counter-free automata and temporal logics.

2.2.2 Properties of Automata over Finite Words

Given a language $\mathcal{L} \subseteq (\Sigma)^*$ of finite words, we define the *reverse language* of $\mathcal{L}$, written $\mathcal{L}^{-1}$, as follows:

$$(\mathcal{L})^{-1} \coloneqq \{\sigma \in (\Sigma)^* \mid \sigma = \langle \sigma_n, \sigma_{n-1}, \ldots, \sigma_0 \rangle \wedge \langle \sigma_0, \sigma_1, \ldots, \sigma_n \rangle \in \mathcal{L}\} \tag{2.1}$$

Both DFA and counter-free automata over finite words are closed under the *reverse* operation $(\cdot)^{-1}$.

Proposition 2 (McNaughton and Paper [149])**.** *Let $\mathcal{L}$ be a language of finite words. $\mathcal{L}$ is recognized by some* DFA *(resp. counter-free automaton) if and only if $(\mathcal{L})^{-1}$ is recognized by some* DFA *(resp.* cf-DFA*).*

Proof. Given an automaton, it suffices to switch its initial states with its final states (and viceversa), and to invert the direction of its transition edges. □

DFAs (and thus also NFAs) have a strong connection with the theory of linear orders, that is S1S. In fact, the seminal result proved independently by Büchi [35] and Elgot [85] establishes that the languages over finite words recognizable by a DFA (or NFA) are exactly those definable in Weak Sequential Calculus, that is S1S interpreted over finite linear orders. McNaughton and Papert proved additionally that, if we restrict the automata to be counter-free, than the languages that they recognize are exactly those definable in the first-order fragment of S1S (over finite linear orders) [149].

Theorem 2 (Büchi [35], Elgot [85])**.** *Let $\mathcal{L}$ be a language of finite words. It holds that $\mathcal{L}$ is recognized by a* NFA *if and only if $\mathcal{L} \in [\![\mathsf{S1S}]\!]^{<\omega}$.*

Theorem 3 (McNaughton and Papert [149])**.** *Let $\mathcal{L}$ be a language of finite words. It holds that $\mathcal{L}$ is recognized by a* cf-NFA *if and only if $\mathcal{L} \in [\![\mathsf{S1S}[\mathsf{FO}]]\!]^{<\omega}$.*

2.2.3 Automata Over Infinite Words

We now consider the case of infinite words (infinite linear orders). For a finite alphabet Σ, we recall that an ω-word $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ over Σ is an element of Σ^ω.

Büchi Automata

One of the first types of automata reading infinite words that were introduced is the class of Büchi automata [34, 193], originally used by Büchi to prove the decidability of the S1S logic over infinite linear orders [34], and thus extending his result in Theorem 2.

Definition 11 (Büchi automata [193])**.** *A* Nondeterministic Büchi Automaton *(*NBA*, for short) is a tuple $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, where:*

(i) Σ *is a finite alphabet; (ii)* Q *is a finite set of states; (iii)* $I \subseteq Q$ *is the set of* initial states*; (iv)* $\delta \subseteq Q \times \Sigma \times Q$ *is the* transition relation*, and (v)* $F \subseteq Q$ *is the set of* final states*. If* δ *is a* function *(*$\delta : Q \times \Sigma \to Q$*), then* $\mathcal{A}$ *is called a* Deterministic Büchi Automaton *(*DBA*, for short).*

A *counter-free* NBA (cf-NBA for short) is defined as an NBA for which we require the counter-free property of Definition 10 [191].

In order for an infinite word to be accepted by a Büchi automata, the word has to induce the automaton to visit *infinitely many times* at least one of its final states. We define the notion of *run* and *language* of a Büchi automaton as follows.

Definition 12 (Run and Language of a NBA)**.** *Let* $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ *be a* NBA*, and let* $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle \in \Sigma^\omega$ *be a* ω*-word over* Σ*. A* run *of* $\mathcal{A}$ *over* σ *is a infinite sequence of states* $\tau = \langle q_0, q_1, \ldots \rangle \in Q^\omega$ *such that* $q_0 \in I$ *and* $(q_i, \sigma_i, q_{i+1}) \in \delta$*, for* $i = 0, \ldots, n$*. We say that* τ *is* accepting *if and only if there exists at least a final state* $q^f \in F$ *such that* $q_i = q^f$ *for* infinitely many $i \in \mathbb{N}$*. A* ω*-word* $\sigma \in \Sigma^\omega$ *is* accepted *by* $\mathcal{A}$ *if and only if there exist an accepting run of* $\mathcal{A}$ *over* σ*. The language* accepted *(or* recognized*) by* $\mathcal{A}$*, denoted as* $\mathcal{L}(\mathcal{A})$*, is the set of all and only the* ω*-words* $\sigma \in \Sigma^\omega$ *accepted by* $\mathcal{A}$*.*

The product between two Büchi automata is defined exactly as in Definition 9.

The *emptiness problem* of a NBA $\mathcal{A}$ consists in determining whether $\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \varnothing$. Similarly for the case of finite words, this problem is decidable in polynomial time (with respect to the size of the automaton) and nondeterministic logarithmic space [196, 168], since $\mathcal{L}(\mathcal{A}) \neq \varnothing$ if and only if the graph representing the structure of the automaton contains a directed path from one initial state of $\mathcal{A}$ to one of its final state, and from such a final state to itself.

Contrary to the case of automata over finite words, the class of languages recognized by NBAs is strictly greater than the class of languages recognized by DBAs [193]. For example, take the language over the alphabet $\Sigma = \{a, b\}$ consisting of all and only the ω-words containing *finitely* many positions in which a holds:

$$\mathcal{L}_{fin} = \{\sigma \subseteq \{a, b\}^\omega \mid \exists^{<\omega} i \in \mathbb{N} \,.\, \sigma_i = a\} \tag{2.2}$$

It can be shown that there not exists any DBA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}_{fin}$ [193].

In [34], Büchi gave the fundamental result that the ω-languages definable in S1S (interpreted over infinite linear orders) are exactly those recognizable by Nondeterministic Büchi Automata. As mentioned before, this result generalizes Theorem 2 to the case of ω-words. Additionally, Thomas [191] proved that if we restrict the Büchi automaton to be *counter-free*, than the languages that we obtain are exactly those definable in S1S[FO] (the first-order fragment of S1S) interpreted over infinite linear orders. This can be considered as a generalization of Theorem 3 to ω-words.

Theorem 4 (Büchi [34])**.** *Let $\mathcal{L}$ be an ω-language. It holds that $\mathcal{L}$ is recognized by a* NBA *if and only if $\mathcal{L} \in [\![\mathsf{S1S}]\!]$.*

Theorem 5 (Thomas [191])**.** *Let $\mathcal{L}$ be an ω-language. It holds that $\mathcal{L}$ is recognized by a* cf-NBA *if and only if $\mathcal{L} \in [\![\mathsf{S1S[FO]}]\!]$.*

The proof of Theorem 4 is based on a language-preserving correspondence between any sentence of S1S to a (nondeterministic) Büchi automaton, and *vice versa*. This means that the satisfiability problem of any S1S-sentence ϕ amounts to the emptiness problem of a NBA built starting from ϕ. Since the emptiness problem of NBAs is decidable, it follows the decidability of S1S-sentences over ω-words (see Theorem 1).

Safety Automata

Theorem 4 establishes a correspondence between S1S-definable languages and Nondeterministic Büchi Automata. In this part, we define another type of automata working on ω-words, called *safety automata*, that corresponds exactly to the SAFETY class (see Definition 3). *Safety automata* have a weaker accepting condition with respect to Büchi's one. In fact, instead of requiring a final state to be visited infinitely often, they require only the *existence* of an infinite run.

Definition 13 (Safety automata [201])**.** *A* Nondeterministic Safety Automaton *(*NSA*, for short) is a tuple $\mathcal{A} = (\Sigma, Q, I, \delta)$, where: (i) Σ is a finite alphabet; (ii) Q is a finite set of states; (iii) $I \subseteq Q$ is the set of* initial states*, and (iv) $\delta \subseteq Q \times \Sigma \times Q$ is the* transition relation*. If δ is a* partial function *($\delta : Q \times \Sigma \to Q$), then $\mathcal{A}$ is called a* Deterministic Safety Automaton *(*DSA*, for short). Let $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle \in \Sigma^\omega$ be a ω-word. A* run *of $\mathcal{A}$ over σ is an* infinite *sequence of states*

$\tau = \langle q_0, q_1, \ldots \rangle \in Q^\omega$ *such that* $q_0 \in I$ *and* $(q_i, \sigma_i, q_{i+1}) \in \delta$*, for* $i = 0, \ldots, n$*. An* ω*-word* σ *is* accepted *by* $\mathcal{A}$ *if and only if there exists a run of* $\mathcal{A}$ *over* σ*. The* language *of* $\mathcal{A}$ *(denoted with* $\mathcal{L}(\mathcal{A})$*) is the set of* ω*-words that are accepted by* $\mathcal{A}$*.*

It is simple to see that the language recognized by any safety automaton is a *safety language* (see Definition 3). In fact, it suffices to observe that the only way for an ω-word to be rejected by a safety automaton is to induce a run that at some point gets stuck, that is, it cannot continue to any state reading the current letter. This implies that each ω-word that is rejected by a safety automaton has a finite witness, or equivalently, it can be rejected by the automaton in a finite number of steps. Since this is exactly the definition of safety language (Definition 3), it follows that the language is safety.

It also holds the converse result, that is, all the S1S-definable safety languages can be defined by a safety automaton. In fact, in [127] Kupferman and Vardi showed that, given a nondeterministic Büchi automaton $\mathcal{A}$, there exists a DFA recognizing $\mathit{BadPref}(\mathcal{L}(\mathcal{A}))$. Since by Theorem 4 the class of S1S-definable languages is equivalent to the class of languages recognizable by a nondeterministic Büchi automaton, it follows the equivalence between the safety fragment of S1S and the class of safety automata.

Theorem 6. *Let* $\mathcal{L}$ *be a language over infinite words. It holds that* $\mathcal{L}$ *is recognized by some* NSA *if and only if* $\mathcal{L} \in$ SAFETY*.*

Duality between Safety Automata and NFAs. Nondeterministic safety automata are duals of NFAs because, for each ω-language $\mathcal{L}$, there exists a NSA $\mathcal{A}$ recognizing $\mathcal{L}$ (*i.e.*, $\mathcal{L}(\mathcal{A}) = \mathcal{L}$) if and only if there exists a NFA $\mathcal{A}'$ accepting the set of its bad-prefixes (*i.e.*, $\mathcal{L}(\mathcal{A}') = \mathit{BadPref}(\mathcal{L}) = \mathit{GoodPref}(\overline{\mathcal{L}})$). The proof of this equivalence relies on a simple translation of NSAs to NFAs, and *vice versa*. Take a NFA $\mathcal{A}' = (\Sigma, Q', I', \delta', F')$. We define the NSA $\mathcal{A} = (\Sigma, Q, I, \delta)$ as follows:

- $Q = Q' \setminus F'$;
- $I = I' \setminus F'$;
- for each $q \in Q$ and for each $a \in \Sigma$, we define $\delta(q, a) = \delta'(q, a)$ if $\delta'(q, a) \subseteq Q$, and $\delta(q, a) = \varnothing$ otherwise.

It is simple to see that $\mathcal{L}(\mathcal{A}') = \mathit{BadPref}(\mathcal{L}(\mathcal{A}))$ [201]. The opposite direction (from DSA to DFA) is specular. Moreover, it is easy to see that if we start with a *deterministic* safety automaton, then the construction produces a *deterministic* finite automaton, and *vice versa*. Finally, since there is an equivalence-preserving translations from NFAs to DFAs (*i.e.*, the subset construction algorithm), the translations from NSAs to NFAs and from DFAs to DSAs can be used to show that, unlike Büchi automata, the nondeterminism does *not* add expressive power to safety automata (*i.e.*, NSAs are equivalent to DSAs). In fact, given an NSA it suffices to: (i) build the equivalent NFA for the set of its bad-prefixes, (ii) determinize it with subset contruction, (iii) build the equivalent DSA for the the DFA obtained in the previous point.

2.3 Regular and ω-Regular Expressions

Regular expressions [115] have been one of the first language-defining formalisms to be studied. They have been introduced in 1951 by Stephen Cole Kleene during his studies on formal languages [123]. One of the most relevant results in the theory of formal languages is that regular expressions are equivalent to finite automata (over finite words) Nevertheless, despite being equivalent formalisms, regular expressions offer a more declarative way to express regular properties than using finite automata. This may be due to the algebraic laws that govern regular expressions, which reseamble the laws of arithmetic.

Regular expressions stay at the core of the foundations of computer science, not only for their role in the study of formal languages, that in turn are the precursors of theoretical computer science, but also for their many uses in practical scenarios. Nowadays, regular expressions are constantly used as a declarative way for specifying the behavior of a *lexer*, that are the very first units in each parser for computer programming languages. Another fundamental use of regular expressions can be found in all those applications that require the search of a pattern inside a text. The well-known Unix utility grep takes in input a pattern in form of a regular expression, and returns all the occurrences of the pattern in a text. It does so by converting the expression into a DFA or NFA and by simulating this automaton on the text being searched [115].

ω-Regular expressions are the extension of regular expression for infinite words. They maintain the algebraic feature of regular expression but work with sets of infinite words. They were introduced by Büchi as a way to give a sort of normal form for the languages recognizable by Büchi automata [34]. Just like regular expressions are equivalent to automata over finite words, we will see that ω-regular expressions have the same expressive power of Büchi automata.

2.3.1 Regular Expressions

A *regular expression* [115] is built starting from the symbols of a finite alphabet Σ, the symbol for the empty word ε, and a set of operations between sets of *finite words*. This operations can be: (i) the union ($\cup$), (ii) the negation ($\neg$), (iii) the concatenation ($\cdot$), and (iv) the *Kleene'star* (*). The language *represented* by a regular expression E is the set of finite words denoted by E, by interpreting the three operations of union, concatenation and Kleene'star as follows (U and V are sets of finite words):

(i) $U \cup V \coloneqq \{\sigma \in \Sigma^* \mid \sigma \in U \vee \sigma \in V\}$

(ii) $\neg U \coloneqq \Sigma^* \setminus U$

(iii) $U \cdot V \coloneqq \{\sigma \in \Sigma^* \mid$
$\sigma = \langle \sigma_0, \ldots, \sigma_m, \sigma_{m+1}, \ldots, \sigma_n \rangle,$
$\langle \sigma_0, \ldots, \sigma_m \rangle \in U$
$\langle \sigma_{m+1}, \ldots, \sigma_n \rangle \in V\}$

(iv) $U^* \coloneqq \bigcup_{i \geq 0} U^i$, where $U^0 \coloneqq U$ and $U^i \coloneqq U^{i-1} \cdot U$.

We refer to [115] for a comprehensive description of regular expressions. A language is called *regular* if and only if there exists a regular expression representing it. We call REG the set of all and only the languages recognized by a regular expression. It is well known that regular expressions have the same expressive power of automata over finite words.

Theorem 7 ([115])**.** *A language $\mathcal{L}$ over finite words is recognized by some* NFA *if and only if* $\mathcal{L} \in$ REG.

As a corollary of Theorem 2, we obtain that a language over finite words is S1S-definable if and only if it stands in REG.

A regular expression is called *star-free* if it is built without the use of the Kleene'star. We call SF the set of all and only the languages recognized by a star-free expression. In [149], McNaughton and Papert prove that star-free expressions have the same expressive power of counter-free automata over finite words (recall Definition 10).

Theorem 8 ([149]). *A language $\mathcal{L}$ over finite words is recognized by a* cf-DFA *if and only if $\mathcal{L} \in$* SF.

As a corollary of Theorem 3, we obtain that a language of finite words is definable in S1S[FO] (*i.e.*, the first-order fragment of S1S) if and only if it stands in SF. Intuitively, this means that *Kleene'star* needs second-order quantifications for being expressed.

Finally, as a corollary of Proposition 2, we obtain that star-free expression are closed under the *reverse* operation $(\cdot)^{-1}$.

Corollary 1. *Star-free expressions are closed under the reverse operation.*

2.3.2 ω-Regular Expressions

The definition of regular expressions has been extended to the case of ω-words [34] into what are called ω-regular expressions. An ω-regular expression is an expression of the form:

$$\bigcup_{i=1}^{n} U_i \cdot (V_i)^{\omega}$$

where $n \in \mathbb{N}$ and $U_i, V_i \in$ REG for $i = 1, \ldots, n$. A language is called *ω-regular* when there exists a ω-regular expression representing it. We call ω-REG the set of all and only the languages of infinite words that are represented by an ω-regular expression. By exploiting the closure properties of Büchi automata, the following theorem can be proved [34].

Theorem 9 (Büchi [34]). *For each language $\mathcal{L}$ of infinite words, it holds that $\mathcal{L} \in \omega$-*REG *if and only if $\mathcal{L}$ is recognized by a* NBA.

As a corollary of Theorem 4, it follows that ω-regular expressions can define exactly the language definable in S1S.

Interestingly, ω-regular languages admit a simple and elegant *normal form*. For any regular (or star-free) language $K \subseteq \Sigma^*$ we

define the operator $\lim(\cdot)$ as follows.

$$\lim(K) \coloneqq \{\sigma \in \Sigma^\omega \mid \exists^\omega i \,.\, \sigma_{[0,i]} \in K\} \tag{2.3}$$

where the symbol $\exists^\omega i$ means that *"there are infinitely many i"*. A Lemma used in McNaughton's Theorem [148, 193], which is a fundamental result for the determinization of Büchi automata into automata with another type of acceptance condition (called Muller automata), provides a normal form for each ω-regular expression.

Theorem 10 (Normal Form Theorem for ω-REG [148, 193])**.** *Any ω-language $\mathcal{L} \in \omega$-REG can be represented in the form:*

$$\bigcup_{i=1}^{n} (\lim(K_i) \cap \neg \lim(K'_i))$$

where $K_i, K'_i \in$ REG, for $i = 1, \ldots, n$.

We remark that the outermost union in the normal form of Theorem 10 derives from the union of all the equivalence's classes of a particular equivalence relation of finite index [148, 193].

Similarly to the case of finite words, we say that an ω-regular expression $\bigcup_{i=1}^{n} U_i \cdot (V_i)^\omega$ is *star-free* if each U_i and each V_i are star-free. We say that an ω-language is *star-free* if and only if there exists a star-free ω-regular expression representing it, and we call ω-SF the set of all and only the star-free ω-languages. Thomas [190] proved that star-free ω-regular expressions and counter-free Büchi automata have the same expressive power, obtaining a generalization of Theorem 8 for finite words.

Theorem 11 ([190])**.** *A ω-language $\mathcal{L}$ is recognized by a* cf-NBA *if and only if $\mathcal{L} \in \omega$-SF.*

As a corollary of Theorem 5, it follows that star-free ω-regular expressions can define exactly the ω-languages definable in S1S[FO].

Thomas [191] proved that Theorem 10 can be specialized for the case of ω-SF languages, obtaining a normal form for the ω-SF class as well.

Theorem 12 (Normal Form Theorem for ω-SF [191, 193])**.** *Any ω-language $\mathcal{L} \in \omega$-SF can be represented in the form:*

$$\bigcup_{i=1}^{n} (\lim(K_i) \cap \neg \lim(K'_i))$$

where $K_i, K'_i \in$ SF, for $i = 1, \ldots, n$.

2.3.3 Characterization of safety and co-safety classes

The classes REG and SF are important also for giving a characterization of the SAFETY (Definition 3) and the coSAFETY (Definition 4) classes. In particular, we can characterize the coSAFETY class by requiring that an ω-language $\mathcal{L}$ is a co-safety language if and only if $\mathcal{L} = K \cdot (\Sigma)^\omega$, where $K \in \mathsf{REG}$ is regular language (over finite words). Following this line, one can also restrict K to be a star-free language (*i.e.*, $K \in \mathsf{SF}$), obtaining two classes, $\mathsf{coSAFETY}^{\mathsf{SF}}$ and $\mathsf{SAFETY}^{\mathsf{SF}}$, which are strictly contained into coSAFETY and SAFETY, respectively.

Definition 14 (Safety and co-safety classes)**.** *We define the following classes of ω-languages:*

- $\mathsf{SAFETY} \coloneqq \{\mathcal{L} \subseteq (\Sigma)^\omega \mid \overline{\mathcal{L}} = K \cdot (\Sigma)^\omega \wedge K \in \mathsf{REG}\}$
- $\mathsf{SAFETY}^{\mathsf{SF}} \coloneqq \{\mathcal{L} \subseteq (\Sigma)^\omega \mid \overline{\mathcal{L}} = K \cdot (\Sigma)^\omega \wedge K \in \mathsf{SF}\}$
- $\mathsf{coSAFETY} \coloneqq \{\mathcal{L} \subseteq (\Sigma)^\omega \mid \mathcal{L} = K \cdot (\Sigma)^\omega \wedge K \in \mathsf{REG}\}$
- $\mathsf{coSAFETY}^{\mathsf{SF}} \coloneqq \{\mathcal{L} \subseteq (\Sigma)^\omega \mid \mathcal{L} = K \cdot (\Sigma)^\omega \wedge K \in \mathsf{SF}\}$

We remark that the definition of the SAFETY class (resp. coSAFETY class) given above is equivalent to the one given in Definition 3 (resp. in Definition 4).

Consider the classes $\mathsf{SAFETY}^{\mathsf{SF}}$ and $\mathsf{coSAFETY}^{\mathsf{SF}}$. It is natural to ask whether the set of languages definable in $\mathsf{SAFETY}^{\mathsf{SF}}$ (resp. $\mathsf{coSAFETY}^{\mathsf{SF}}$) coincides with the set of languages definable in ω-SF that are also safety languages (resp. co-safety languages). In [192], Thomas give a positive answer to this question.

Proposition 3 ([192])**.** *It holds that:*

1. $\mathsf{SAFETY}^{\mathsf{SF}} = \omega\text{-}\mathsf{SF} \cap \mathsf{SAFETY}$
2. $\mathsf{coSAFETY}^{\mathsf{SF}} = \omega\text{-}\mathsf{SF} \cap \mathsf{coSAFETY}$

2.4 Linear Temporal Logics

In the previous sections, we saw some formalisms for expressing properties about sequences, like the first- and second-order logics S1S and S1S[FO], regular and ω-regular expressions, and automata

over finite or infinite words. In this section, we take a look at another formalism for defining properties of sequences, called *Linear Temporal Logic* (LTL, for short). LTL was introduced by Pnueli [159] in the late seventies as a way for expressing properties of computations of computer programs and, consequently, for verifying whether those computations are compliant with properties formalized in the language of LTL. Differently from the S1S and the S1S[FO] formalisms, that are second-order and first-order logics, respectively, LTL is a *modal* logic. Giving a precise definition of modal logic would inevitably result into an incomplete definition. For the sake of this thesis, it suffices to know that LTL is a logic featuring particular operators for moving to the right or to the left from any point of a linear order. The choice of the operators of LTL proved to be carefully designed: LTL is equivalent to the first-order fragment of S1S (*i.e.*, S1S[FO]). However, as we will see in the next chapters, while the complexity of S1S and S1S[FO] is nonelementary [185], meaning that the time spent by any Turing machine for deciding the satisfiability of a formula can be described only by an exponential function of unbounded height, the complexity of LTL is PSPACE-complete [179].

2.4.1 Syntax

Linear Temporal Logic (LTL) is a modal logic interpreted over infinite, discrete linear orders [159, 77]. Syntactically, LTL can be seen as an extension of propositional logic with the addition of the *next* (also called *tomorrow*, $\mathsf{X}\phi$, *i.e.*, at the *next* state ϕ holds) and the *until* ($\phi_1 \,\mathsf{U}\, \phi_2$, *i.e.*, ϕ_2 will eventually hold and ϕ_1 will hold *until* then) temporal operators.

LTL *with Past* extends LTL with the addition of temporal operators able to talk about what happened in the *past* with respect to the current time. LTL+P is obtained from LTL by adding the following *past* temporal operators: (i) the *yesterday* operator ($\mathsf{Y}\phi$, *i.e.*, there exists a *previous* state in which ϕ holds); (ii) the Z operator ($\mathsf{Z}\phi$, *i.e.*, either a previous state does not exists or in the previous state ϕ holds); (iii) and the *since* operator ($\phi_1 \,\mathsf{S}\, \phi_2$, *i.e.*, there was a past state where ϕ_2 held, and ϕ_1 has held *since* then). We will now briefly recall the syntax and semantics of LTL+P, which encompasses that of LTL as well. Formally, given a set Σ of proposition letters, LTL+P

formulas over Σ are generated by the following grammar:

$$\left.\begin{array}{l}\phi := p \mid \end{array}\right\} \quad \text{propositional atoms}$$

$$\left.\begin{array}{l}\phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \\ \neg\phi \mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \leftrightarrow \phi_2\end{array}\right\} \quad \text{propositional connectives}$$

$$\left.\begin{array}{l}\mathsf{X}\phi_1 \mid \phi_1 \,\mathsf{U}\, \phi_2 \mid \phi_1 \,\mathsf{R}\, \phi_2 \mid \\ \mathsf{F}\phi_1 \mid \mathsf{G}\phi_1\end{array}\right\} \quad \text{future temporal operators}$$

$$\left.\begin{array}{l}\mathsf{Y}\phi_1 \mid \phi_1 \,\mathsf{S}\, \phi_2 \mid \phi_1 \,\mathsf{T}\, \phi_2 \mid \\ \mathsf{P}\phi_1 \mid \mathsf{H}\phi_1 \mid \mathsf{Z}\phi_1\end{array}\right\} \quad \text{past temporal operators}$$

where $p \in \Sigma$ and ϕ_1 and ϕ_2 are LTL+P formulas. Most of the Boolean and temporal operators of the language can be defined in terms of a small number of basic ones. As for Boolean operators, we have that: (i) $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$; (ii) $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$; (iii) $\phi_1 \leftrightarrow \phi_2 \equiv (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$. As for the temporal operators, the *release* ($\phi_1 \,\mathsf{R}\, \phi_2 \equiv \neg(\neg\phi_1 \,\mathsf{U}\, \neg\phi_2)$), *eventually* ($\mathsf{F}\phi \equiv \top \,\mathsf{U}\, \phi$), and *always* ($\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$) future operators can all be defined in terms of the *until* operator, while the *triggered* ($\phi_1 \,\mathsf{T}\, \phi_2 \equiv \neg(\neg\phi_1 \,\mathsf{S}\, \neg\phi_2)$), *once* ($\mathsf{P}\phi \equiv \top \,\mathsf{S}\, \phi$), and *historically* ($\mathsf{H}\phi \equiv \neg\mathsf{P}\neg\phi$) past operators can all be defined in terms of the *since* operator. Finally, the *weak yesterday* operator can be defined in terms of the *yesterday* operator using the negation ($\mathsf{Z}\phi \equiv \neg\mathsf{Y}\neg\phi$).

The *until* (U) and the *eventually* (F) are said *existential* temporal operators, while the *release* (R) and the *globally* (R) are said *universal* temporal operators.

We say that a LTL+P formula is *pure past* if and only if all the temporal operators inside the formula are past operators. We call *pure past* LTL+P, written as $\mathsf{LTL+P_P}$, the fragment of LTL+P containing only pure past formulas.

2.4.2 Semantics

Formulas from LTL+P (built starting from an alphabet Σ) are interpreted over *state sequences*. A *state sequence* $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ over Σ is a *finite* or *infinite*, linearly ordered sequence of *states*, where each state σ_i is a set of proposition letters, that is $\sigma_i \in (2^\Sigma)$ for each $i \in \mathbb{N}$. Following the definitions and notations for (ω-)words given in Section 2.1, an infinite (resp. finite) state sequence over the alphabet

Σ is an ω-word (resp. a word) over the alphabet 2^{Σ}. Therefore, we can reuse the basic operations on (ω-)words for state sequences as well. We recall some of those here below. With $|\sigma|$ we will indicate the length of σ: if σ is of infinite length, then $|\sigma| = \omega$; otherwise, if $\sigma = \langle \sigma_0, \ldots, \sigma_n \rangle$, then $|\sigma| = n + 1$. Given two indices $i, j \in \mathbb{Z}$, with $i \leq j$, we denote as $\sigma_{[i,j]}$ the interval $\langle \sigma_i, \ldots, \sigma_j \rangle$ if $i \geq 0$, or the interval $\langle \sigma_0, \ldots, \sigma_j \rangle$ otherwise. Similarly, for some $i \in \mathbb{N}$, with $\sigma_{[i,\infty)}$ we refer to the suffix of σ starting from index i.

Standard Interpretation

The standard interpretation of LTL+P is tipically given on infinite state sequences. Therefore, from now on, unless otherwise stated, with the term *state sequence* we will refer to an *infinite* one. Given a state sequence σ, a position $i \geq 0$, and a LTL+P formula ϕ, we inductively define the *satisfaction* of ϕ by σ at position i, written as $\sigma, i \models \phi$, as follows:

1. $\sigma, i \models p$ iff $p \in \sigma_i$;
2. $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$;
3. $\sigma, i \models \phi_1 \vee \phi_2$ iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$;
4. $\sigma, i \models \phi_1 \wedge \phi_2$ iff $\sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$;
5. $\sigma, i \models \mathsf{X}\phi$ iff $\sigma, i + 1 \models \phi$;
6. $\sigma, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\sigma, i - 1 \models \phi$;
7. $\sigma, i \models \mathsf{Z}\phi$ iff either $i = 0$ or $\sigma, i - 1 \models \phi$;
8. $\sigma, i \models \phi_1 \,\mathsf{U}\, \phi_2$ iff there exists $j \geq i$ such that $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $i \leq k < j$;
9. $\sigma, i \models \phi_1 \,\mathsf{S}\, \phi_2$ iff there exists $j \leq i$ such that $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $j < k \leq i$;
10. $\sigma, i \models \phi_1 \,\mathsf{R}\, \phi_2$ iff either $\sigma, j \models \phi_2$ for all $j \geq i$, or there exists $k \geq i$ such that $\sigma, k \models \phi_1$ and $\sigma, j \models \phi_2$ for all $i \leq j \leq k$;
11. $\sigma, i \models \phi_1 \,\mathsf{T}\, \phi_2$ iff either $\sigma, j \models \phi_2$ for all $0 \leq j \leq i$, or there exists $k \leq i$ such that $\sigma, k \models \phi_1$ and $\sigma, j \models \phi_2$ for all $i \geq j \geq k$

We say that σ *satisfies* ϕ, written as $\sigma \models \phi$, if it satisfies the formula at the first state, *i.e.*, if $\sigma, 0 \models \phi$: in this case, we call σ a *model* of ϕ. We say that two formulas ϕ and ψ are *equivalent*

($\phi \equiv \psi$) if and only if they are satisfied by the same set of state sequences. We say that they are *strongly equivalent* ($\phi \doteq \psi$) when $\sigma, i \models \phi$ if and only if $\sigma, i \models \psi$, for all $\sigma \in (2^{\Sigma})^{\omega}$ and for all $i \in \mathbb{N}$. The *language* of ϕ, written $\mathcal{L}(\phi)$, is defined as the set of models of the formula, that is, $\mathcal{L}(\phi) = \{\sigma \in (2^{\Sigma})^{\omega} \mid \sigma \models \phi\}$. Equivalently, $\mathcal{L}(\phi)$ is the set of the ω-words such that, when considered as infinite state sequences, are models of ϕ. Given a language $\mathcal{L} \subseteq (2^{\Sigma})^{\omega}$ we say that $\mathcal{L}$ is $\mathsf{LTL{+}P}$-*definable* if and only if there exists a formula $\phi \in \mathsf{LTL{+}P}$ such that $\mathcal{L} = \mathcal{L}(\phi)$. A formula of $\mathsf{LTL{+}P}$ is said to be in *Negated Normal Form* (NNF, for short) if and only if all the negations inside the formula are applied only to propositional atoms.

Pure Past $\mathsf{LTL{+}P}$

Consider a *pure past* formula. Since all its temporal operators refer to the past, pure past formulas are typically interpreted at time points different from the starting one (thus differently from the case of $\mathsf{LTL{+}P}$ formulas). In particular, since the past is always bounded, in the sense that each state sequence $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ has always a state without precedessors (*i.e.*, σ_0), each pure past formula is interpreted at the last state of a *finite* state sequence, and the language of the formula is always a language of *finite words*. Let $\phi \in \mathsf{LTL{+}P_P}$. We define the language of ϕ in this way:

$$\mathcal{L}(\phi) \coloneqq \{\sigma \in (2^{\Sigma})^{*} \mid \sigma = \langle \sigma_0, \ldots, \sigma_n \rangle \wedge \sigma, n \models \phi\} \tag{2.4}$$

$\mathsf{LTL{+}P}$ over finite words

We consider now the case of $\mathsf{LTL{+}P}$ interpreted over *finite* state sequences. Its syntax is obtained from the one of $\mathsf{LTL{+}P}$ by adding the *weak next* operator ($\widetilde{\mathsf{X}}$, also called *weak tomorrow*). It is easy to modify the definition of the standard semantics of $\mathsf{LTL{+}P}$ in order for the formulas to be interpreted over *finite* state sequences. Let $\sigma = \langle \sigma_0, \ldots, \sigma_n \rangle \in (2^{\Sigma})^{*}$ be a state sequence of length $n + 1$, and let ϕ be a $\mathsf{LTL{+}P}$-formula. The satisfaction of ϕ over σ, written as $\sigma \models^{<\omega} \phi$, is defined as follows (the cases for the Boolean operators and the past temporal operators are identical to the infinite case, while the F and G modalities easily follow from these ones):

1. $\sigma, i \models^{<\omega} \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i + 1 \models^{<\omega} \phi$;
2. $\sigma, i \models^{<\omega} \widetilde{\mathsf{X}}\phi$ iff $i + 1 < |\sigma|$ implies $\sigma, i + 1 \models^{<\omega} \phi$;

3. $\sigma, i \models^{<\omega} \phi_1 \,\mathsf{U}\, \phi_2$ iff there exists $i \leq j < |\sigma|$ such that $\sigma, j \models^{<\omega} \phi_2$, and $\sigma, k \models^{<\omega} \phi_1$ for all k, with $i \leq k < j$;
4. $\sigma, i \models^{<\omega} \phi_1 \,\mathsf{R}\, \phi_2$ iff either $\sigma, j \models^{<\omega} \phi_2$ for all $i \leq j < |\sigma|$, or there exists $i \leq k < |\sigma|$ such that $\sigma, k \models^{<\omega} \phi_1$ and $\sigma, j \models^{<\omega} \phi_2$ for all $i \leq j \leq k$;

The *weak next* operator can be defined in terms of the negation and the *next* operator ($\widetilde{\mathsf{X}}\phi \equiv \neg\mathsf{X}\neg\phi$). We call $\mathcal{L}^{<\omega}(\phi)$ the set of all and only the *finite* state sequences σ that satisfy ϕ under finite semantics, that is, such that $\sigma \models^{<\omega} \phi$.

2.4.3 Properties

In this part, we recap the main properties of LTL (with only future operators) under both finite and infinite semantics. We pay particular attention to the expressive power of LTL, by comparing it with other formalisms (like S1S or regular expressions). We then show that past operators do *not* add expressive power, and thus LTL+P, although being exponentially more succinct than LTL in the worst case, is expressively equivalent to LTL. We conclude this part by showing some safety and co-safety fragments of LTL.

Expressive Power

Consider the LTL logic, that is LTL+P devoid of past operators. A central theorem, due to Kamp [120] and Gabbay [97], proves that the set of LTL-definable languages is exactly the set of S1S[FO]-definable languages, that is, the languages definable in the *first-order* fragment of S1S. In some sense, this proves that the choice of the temporal modalities in LTL was carefully designed. Interestingly, this holds for the case of finite state sequences as well.

Theorem 13 (Kamp [120], Gabbay [97])**.** *It holds that:*

1. $[\![\mathsf{LTL}]\!] = [\![\mathsf{S1S[FO]}]\!]$
2. $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{S1S[FO]}]\!]^{<\omega}$

The proof of Theorem 13 (that can be found in [120, 97]) is constructive, therefore there is an effective algorithm to turn formulas of the first-order fragment of S1S into LTL-formulas, and *vice versa*.

From Theorems 3, 5, 8 and 11, we obtain the following corollary, establishing the equivalence between LTL, star-free languages, and counter-free automata.

Corollary 2. *Let $\mathcal{L}$ be a ω-language and let $\mathcal{L}^{<\omega}$ be a language (over finite words). It holds that:*

1. $\mathcal{L} \in [\![\mathsf{LTL}]\!] \Leftrightarrow \mathcal{L} \in \omega\text{-}\mathsf{SF} \Leftrightarrow \mathcal{L}$ *is recognized by a* cf-NBA*;*
2. $\mathcal{L}^{<\omega} \in [\![\mathsf{LTL}]\!]^{<\omega} \Leftrightarrow \mathcal{L}^{<\omega} \in \mathsf{SF} \Leftrightarrow \mathcal{L}^{<\omega}$ *is recognized by a* cf-NFA*.*

Extended Linear Temporal Logic

Since LTL is expressively equivalent to the first-order fragment of S1S, not all the S1S properties are expressible in LTL. One example is the language $even(p)$, containing all and only the ω-words such that in at least every even position the proposition atom p holds. In [199], Wolper prove that $even(p)$ is *not* expressible in LTL. Therefore, a natural question is how to extend LTL in order to capture the full expressiveness of S1S. In [199], Wolper answers this question by introducing *Extended Temporal Logic* (ETL, for short), defined as an extension of LTL with operators able to express regular expressions. We refer to [199] for more details on ETL.

Theorem 14 (Wolper [199])**.** *It holds that* $[\![\mathsf{ETL}]\!] = [\![\mathsf{S1S}]\!]$.

A normal form for LTL

Recall from Theorem 15 that the ω-SF class has the following normal form:

$$\bigcup_{i=1}^{n} (\lim(K_i) \cap \neg \lim(K'_i))$$

where $K_i, K'_i \in \mathsf{SF}$, for $i = 1, \ldots, n$. Since the class of LTL-definable languages coincides with the class ω-SF (recall Corollary 2) and since $[\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega} = \mathsf{SF}$, from the Normal Form Theorem for the class ω-SF (Theorem 15) we obtain a normal form for the $[\![\mathsf{LTL}]\!]$ class.

Theorem 15 (Normal Form Theorem for LTL [97, 136])**.** *Any ω-language $\mathcal{L} \in \omega\text{-}\mathsf{SF}$ can be represented in the form:*

$$\bigvee_{i=1}^{n} (\mathsf{GF}(\alpha_i) \wedge \neg \mathsf{GF}(\beta_i))$$

or, equivalently, in the form:

$$\bigwedge_{i=1}^{n} (\mathsf{GF}(\alpha_i) \vee \neg \mathsf{GF}(\beta_i))$$

where $\alpha_i, \beta_i \in \mathsf{LTL}{+}\mathsf{P}_\mathsf{P}$, for $i = 1, \ldots, n$.

The role of the past

We consider now the comparison between the expressive power of LTL and $\mathsf{LTL}{+}\mathsf{P}$. Recall from Theorem 13 that $[\![\mathsf{LTL}]\!] = [\![\mathsf{S1S}[\mathsf{FO}]]\!]$ (and the same for the case of finite words), and that there is also an effective translation of any formula of LTL into an equivalent one in $\mathsf{S1S}[\mathsf{FO}]$, and *vice versa*. We can use this translation to prove that $\mathsf{LTL}{+}\mathsf{P}$ is expressively equivalent to LTL. Since $\mathsf{LTL} \subseteq \mathsf{LTL}{+}\mathsf{P}$, obviously it holds that $[\![\mathsf{LTL}]\!] \subseteq [\![\mathsf{LTL}{+}\mathsf{P}]\!]$. In order to prove the other direction ($[\![\mathsf{LTL}{+}\mathsf{P}]\!] \subseteq [\![\mathsf{LTL}]\!]$), it suffices to take any $\mathsf{LTL}{+}\mathsf{P}$-formula, build the equivalent $\mathsf{S1S}[\mathsf{FO}]$-formula (this can be done very simply, by considering the semantics of the operators), and finally use on this formula the procedure for passing from $\mathsf{S1S}[\mathsf{FO}]$ to LTL.

Corollary 3. *It holds that:*

1. $[\![\mathsf{LTL}]\!] = [\![\mathsf{LTL}{+}\mathsf{P}]\!]$
2. $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{LTL}{+}\mathsf{P}]\!]^{<\omega}$

Although the two formalisms are expressively equivalent, it can be proved that $\mathsf{LTL}{+}\mathsf{P}$ is exponentially more succinct than LTL [143].

Proposition 4 (Markey [143])**.** $\mathsf{LTL}{+}\mathsf{P}$ *can be exponentially more succinct than* LTL*.*

Consider now $\mathsf{LTL}{+}\mathsf{P}_\mathsf{P}$, that is *pure past* LTL, and recall that pure past formulas are interpreted at the last state of a *finite* state sequence (Eq. (2.4)). For this reason, the language of any pure past formula is a language of *finite* words. This opens the possibility of a

comparison between $\mathsf{LTL}{+}\mathsf{P_P}$ and LTL interpreted over finite words. In particular, we wonder whether $[\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega} \stackrel{?}{=} [\![\mathsf{LTL}]\!]^{<\omega}$. We will show that there is a positive answer to this question.

Before proving that $[\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega} = [\![\mathsf{LTL}]\!]^{<\omega}$, we show an interesting duality between these two logics. In particular, given the duality of the temporal operators over finite state sequences (X and Y are duals, and U and S are duals), for each language $\mathcal{L}$ of finite words it holds that $\mathcal{L}$ is LTL-definable if and only if $\mathcal{L}^{-1}$ (the *reverse* language of $\mathcal{L}$) is $\mathsf{LTL}{+}\mathsf{P_P}$-definable. Since we didn't find a proof of this result in the literature, we give it here below.

Proposition 5. *Let $\mathcal{L} \in (2^{\Sigma})^*$. It holds that:*

$$\mathcal{L} \in [\![\mathsf{LTL}]\!]^{<\omega} \;\Leftrightarrow\; \mathcal{L}^{-1} \in [\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega}$$

Proof. Consider the left-to-right direction. Let $\mathcal{L} \in [\![\mathsf{LTL}]\!]^{<\omega}$. By definition of $[\![\cdot]\!]^{<\omega}$, there exists a LTL-formula ϕ such that $\mathcal{L}^{<\omega}(\phi) = \mathcal{L}$. We define ϕ' as the formula obtained from ϕ by replacing each operator X with Y, and each operator U with S. Given the duality between the previous pair of operators, it is possible to prove by induction that $(\mathcal{L}^{<\omega}(\phi))^{-1} = \mathcal{L}^{<\omega}(\phi')$, that is $\mathcal{L}^{-1} = \mathcal{L}^{<\omega}(\phi')$. Clearly, $\phi' \in \mathsf{LTL}{+}\mathsf{P_P}$ (*i.e.*, ϕ' syntactically belongs to $\mathsf{LTL}{+}\mathsf{P_P}$), and therefore $\mathcal{L}^{<\omega}(\phi') \in [\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega}$ as well. It follows that $\mathcal{L}^{-1} \in [\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega}$. The right-to-left direction is specular. □

Given this duality between $\mathsf{LTL}{+}\mathsf{P_P}$ and LTL interpreted over finite words, the expressive equivalence between these two logics easily follows from the closure property of star-free expressions under the *reverse* operation (recall Corollary 1), and from the equivalence between LTL and SF (recall Corollary 2). This result is stated in [192, 136] but not proved. Therefore we give here the proof.

Proposition 6 ([192, 136])**.** $[\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega} = [\![\mathsf{LTL}]\!]^{<\omega}$.

Proof. We prove first that $[\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega} \subseteq [\![\mathsf{LTL}]\!]^{<\omega}$.
Let $\mathcal{L} \in [\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega}$. By definition of $[\![\cdot]\!]^{<\omega}$, there exists a pure past formula $\phi \in \mathsf{LTL}{+}\mathsf{P_P}$ such that $\mathcal{L} = \mathcal{L}^{<\omega}(\phi)$. By Proposition 5, there exist a LTL-formula ϕ' such that $\mathcal{L}^{<\omega}(\phi) = (\mathcal{L}^{<\omega}(\phi'))^{-1}$. Since $[\![\mathsf{LTL}]\!]^{<\omega} = \mathsf{SF}$ (Corollary 2) and since SF is closed under the *reverse* operation (Corollary 1), we have also that $(\mathcal{L}^{<\omega}(\phi'))^{-1} \in \mathsf{SF}$. Again by Corollary 2, we have that there exists a LTL-formula ϕ'' such that $\mathcal{L}^{<\omega}(\phi'') = (\mathcal{L}^{<\omega}(\phi'))^{-1}$. It follows that $\mathcal{L} \in [\![\mathsf{LTL}]\!]^{<\omega}$.

Consider now the opposite direction, and take any $\mathcal{L} \in [\![\mathsf{LTL}]\!]^{<\omega}$. By the definition of $[\![\cdot]\!]^{<\omega}$ there exists a formula $\phi \in \mathsf{LTL}$ such that $\mathcal{L} = \mathcal{L}^{<\omega}(\phi)$. By Proposition 5, there exists a $\mathsf{LTL}{+}\mathsf{P_P}$-formula ϕ' such that $\mathcal{L}^{<\omega}(\phi) = (\mathcal{L}^{<\omega}(\phi'))^{-1}$. Since by Corollary 2 $\mathcal{L}^{<\omega}(\phi) \in \mathsf{SF}$, also $(\mathcal{L}^{<\omega}(\phi'))^{-1} \in \mathsf{SF}$. Since SF is closed under the *reverse* operation Corollary 1, it also holds that $\mathcal{L}^{<\omega}(\phi') \in \mathsf{SF}$. By Corollary 2, there exists a LTL-formula ϕ'' such that $\mathcal{L}^{<\omega}(\phi') = \mathcal{L}^{<\omega}(\phi'')$. By applying again Proposition 5, we find that there exists an $\mathsf{LTL}{+}\mathsf{P_P}$-formula ϕ''' such that $(\mathcal{L}^{<\omega}(\phi''))^{-1} = \mathcal{L}^{<\omega}(\phi''')$. It follows that $\mathcal{L} \in [\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega}$. □

The result in Proposition 6 is interesting and definitely not trivial. We can derive the following corollaries.

Corollary 4. *The set* $[\![\mathsf{LTL}]\!]^{<\omega}$ *is closed under the* reverse *operation.*

Corollary 5. $[\![\mathsf{LTL}{+}\mathsf{P_P}]\!]^{<\omega} = \mathsf{SF}$.

2.4.4 Safety and co-Safety fragments of LTL

In this part, we recap some properties of the set of safety and co-safety languages that are definable in LTL.

A formula of $\mathsf{LTL}{+}\mathsf{P}$ is said to be a *safety formula* (resp. *co-safety formula*) if and only if $\mathcal{L}(\phi)$ is a safety language (resp. co-safety language) (recall Definitions 3 and 5).

In Definition 14, we gave the definition of the four classes SAFETY, $\mathsf{SAFETY}^{\mathsf{SF}}$, $\mathsf{coSAFETY}$, and $\mathsf{coSAFETY}^{\mathsf{SF}}$. We recall that SAFETY class is is the class of all the ω-regular languages that are also safety languages, and it can be equivalentely defined as:

$$\mathsf{SAFETY} := \{\mathcal{L} \subseteq (\Sigma)^\omega \mid \overline{\mathcal{L}} = K \cdot (\Sigma)^\omega \wedge K \in \mathsf{REG}\} \tag{2.5}$$

Conversely the $\mathsf{SAFETY}^{\mathsf{SF}}$ class is obtained from SAFETY by restricting the prefixes to belong to the SF class (the class of star-free expressions), and it is exactly the set of all star-free ω-languages that are safety languages (recall Proposition 3), *i.e.*, $\mathsf{SAFETY}^{\mathsf{SF}} = \omega\text{-}\mathsf{SF} \cap \mathsf{SAFETY}$. The same holds for $\mathsf{coSAFETY}$ and $\mathsf{coSAFETY}^{\mathsf{SF}}$ as well.

It is worth noting that it is possible that a formula is both safety and co-safety: one example is the formula $\mathsf{X}p$. Therefore, $\mathsf{SAFETY} \cap \mathsf{coSAFETY} \neq \varnothing$.

Normal Forms

Since in the previous sections we proved that star-free ω-languages are equivalent to $\mathsf{LTL{+}P}$ (recall Corollary 2), and since $\mathsf{SAFETY^{SF}} = \omega\text{-}\mathsf{SF} \cap \mathsf{SAFETY}$, it follows that the $\mathsf{SAFETY^{SF}}$ class comprises exactly the ω-languages that are definable in $\mathsf{LTL{+}P}$ that are safety languages, that is:

$$\mathsf{SAFETY^{SF}} = [\![\mathsf{LTL{+}P}]\!] \cap \mathsf{SAFETY} \tag{2.6}$$

The same holds of course for $\mathsf{coSAFETY^{SF}}$ as well:

$$\mathsf{coSAFETY^{SF}} = [\![\mathsf{LTL{+}P}]\!] \cap \mathsf{coSAFETY} \tag{2.7}$$

Consider now the $\mathsf{SAFETY^{SF}}$ class. By definition, an ω-language $\mathcal{L}$ is in $\mathsf{SAFETY^{SF}}$ if and only if each prefix of each ω-word in $\mathcal{L}$ belongs to a star-free language, or equivalently (by Corollary 5), it is a model of an $\mathsf{LTL{+}P_P}$ formula. Consider the *globally* operator (G) of $\mathsf{LTL{+}P}$. If its argument is a formula of $\mathsf{LTL{+}P_P}$, the globally operator constrains each prefix of each of its models. It follows that the language of such *globally* operators are exactly those that belong to the $\mathsf{SAFETY^{SF}}$ class. From now on, with $\mathsf{G}\,\alpha_{sf}$ we denote the set of formulas of that form, where $\alpha_{sf} \in [\![\mathsf{LTL{+}P_P}]\!]^{<\omega}$ (or, equivalently, $\alpha_{sf} \in \mathsf{SF}$), and as always with $[\![\mathsf{G}\,\alpha_{sf}]\!]$ we refer to the set of ω-languages recognized by a formula of type $\mathsf{G}\,\alpha_{sf}$. It holds that:

$$\mathsf{SAFETY^{SF}} = [\![\mathsf{G}\,\alpha_{sf}]\!] \tag{2.8}$$

Since $\mathsf{SAFETY^{SF}} = \mathsf{LTL{+}P} \cap \mathsf{SAFETY}$, this means that $\mathsf{G}\,\alpha_{sf}$ is a *normal form* for all the safety ω-languages definable in $\mathsf{LTL{+}P}$.

Theorem 16 (Chang *et al.* [40], Thomas [192])**.** *A formula $\phi \in \mathsf{LTL{+}P}$ represents a safety property if and only if there exists a formula $\alpha_{sf}^{\phi} \in \mathsf{LTL{+}P_P}$ such that $\mathsf{G}\alpha_{sf}^{\phi} \equiv \phi$.*

We can generalize the above considerations also to the case where $\alpha_{reg} \in \mathsf{REG}$, that is where the argument of the *globally* operator is a language recognized by a regular expression (not necessarily star-free). We call $\mathsf{G}\,\alpha_{reg}$ the set of formulas of that type where $\alpha_{reg} \in \mathsf{REG}$, and with $[\![\mathsf{G}\,\alpha_{reg}]\!]$ we denote the set of language recognized by a formula of type $\mathsf{G}\,\alpha_{reg}$. By the same considerations above, we have that:

$$\mathsf{SAFETY} = \mathsf{G}\,\alpha_{reg} \tag{2.9}$$

Therefore, we obtain the following *normal form* for every ω-regular safety language.

Theorem 17 (Thomas [192])**.** *A ω-language $\mathcal{L} \in \omega\text{-}\mathsf{REG}$ is a safety property if and only if there exists a formula $\alpha^{\mathcal{L}}_{reg}$ such that $\mathcal{L}^{<\omega}(\alpha^{\mathcal{L}}_{reg}) \in \mathsf{REG}$ and $\mathsf{G}\alpha^{\mathcal{L}}_{reg} \equiv \phi$.*

Of course, in the previous theorem, the formula α^{ϕ}_{reg} cannot be written in $\mathsf{LTL}{+}\mathsf{P}$, since $\mathsf{LTL}{+}\mathsf{P}$ is expressively equivalent to star-free regular expressions. A possibility is to write α^{ϕ}_{reg} with ETL.

The dual considerations hold for $\mathsf{coSAFETY}$ and $\mathsf{coSAFETY}^{\mathsf{SF}}$ as well. In particular, since each language in $\mathsf{coSAFETY}^{\mathsf{SF}}$ comprises only the ω-words containing at least one prefix that has to belong to a star-free language, the *eventually* operator (F) of $\mathsf{LTL}{+}\mathsf{P}$ can be used in combination with formulas of $\mathsf{LTL}{+}\mathsf{P}_{\mathsf{P}}$ in order to give a normal form for the class $\mathsf{coSAFETY}^{\mathsf{SF}}$. We define $\mathsf{F}\,\alpha_{sf}$ (resp $\mathsf{F}\,\alpha_{reg}$) as the set of formulas of that type where $\alpha_{sf} \in \mathsf{SF}$ (resp. $\alpha_{reg} \in \mathsf{REG}$). With $[\![\mathsf{F}\,\alpha_{sf}]\!]$ (resp. $[\![\mathsf{F}\,\alpha_{reg}]\!]$) we refer to the set of ω-languages recognized by a formula of the corresponding type. It holds that:

$$\mathsf{coSAFETY}^{\mathsf{SF}} = [\![\mathsf{F}\,\alpha_{sf}]\!] \tag{2.10}$$

Since $\mathsf{coSAFETY}^{\mathsf{SF}} = \mathsf{LTL}{+}\mathsf{P} \cap \mathsf{coSAFETY}$, we have that $\mathsf{F}\,\alpha_{sf}$ is a *normal form* for all the co-safety ω-languages definable in $\mathsf{LTL}{+}\mathsf{P}$.

Theorem 18 (Chang *et al.* [40], Thomas [192])**.** *A formula $\phi \in \mathsf{LTL}{+}\mathsf{P}$ represents a co-safety property if and only if there exists a formula $\alpha^{\phi}_{sf} \in \mathsf{LTL}{+}\mathsf{P}_{\mathsf{P}}$ such that $\mathsf{F}\alpha^{\phi}_{sf} \equiv \phi$.*

Also the generalization to regular languages holds as well.

Theorem 19 (Thomas [192])**.** *A ω-language $\mathcal{L} \in \omega\text{-}\mathsf{REG}$ is a co-safety property if and only if there exists a formula $\alpha^{\mathcal{L}}_{reg}$ such that $\mathcal{L}^{<\omega}(\alpha^{\mathcal{L}}_{reg}) \in \mathsf{REG}$ such that $\mathsf{F}\alpha^{\mathcal{L}}_{reg} \equiv \phi$.*

Syntactical Fragments

As we will see in the next chapters, a lot of problems in formal verifications, like for instance model checking, satisfiability and realizability, become much easier if the property is a (co-)safety property. Therefore, if the property is expressed by means of a formula of

LTL+P, it is useful knowing whether the formula is a (co-)safety formula, that is whether its language is a (co-)safety language. Unfortunately, Kupferman and Vardi proved that, given an LTL+P-formula, recognizing whether it is (co-)safety is PSPACE-complete [127]. It is a pretty high complexity, given that it is the same complexity of the satisfiability of the same logic. Therefore, a good choice is to express properties by using a *syntactical safety fragment* of LTL (or equivalently a *safety fragment*), that is fragments with which one can express *only* safety properties.

Arguably, one of the most known (and natural) safety fragments of LTL is the one obtained from LTL (with only future temporal operators) by restricting the negation to appear only in front of proposition atoms and by forbidding existential temporal operators, *i.e.*, the *until* and the *eventually*. To the best of our knowledge, this fragment was first studied by Sistla [178], who did not gave it a name. Recently, the name Safety-LTL has become popular for denoting this fragment [201, 105].

Definition 15 (Sistla [178])**.** *The* Safety-LTL *logic is defined as the set of formulas obtained from* LTL *such that, when in* NNF*, do* not *contain any* U *or* F *operator.*

In [178], Sistla prove that Safety-LTL is a *safety fragment* of LTL, that is, each formula $\phi \in$ Safety-LTL is such that $\mathcal{L}(\phi)$ is a safety property. Actually, he prove the following more general result (Chang *et al.* extended it by including also past operators [40]).

Proposition 7 (Sistla [178], Chang *et al.* [40])**.** *Every* pure past *formula of* LTL+P *is a safety property and, if* α *and* β *are safety properties, then so are* $\alpha \wedge \beta$, $\alpha \vee \beta$, $\mathsf{X}\alpha$, $\mathsf{G}\alpha$ *and* $\alpha \,\mathsf{R}\, \beta$.

A very interesting question is whether each safety formula of LTL can be formalized in Safety-LTL as well. In some sense, this would mean that Safety-LTL is *expressively complete* with respect to the set of safety ω-languages definable in LTL. Chang, Manna and Pnueli [40] proved that this is actually the case.

Theorem 20 (Chang *et al.* [40])**.** *It holds that:*

$$[\![\text{Safety-LTL}]\!] = [\![\text{LTL}]\!] \cap \text{SAFETY}$$

All the above definitions and results extend to the case of co-safety properties as well. We define the coSafety-LTL logic as the dual of Safety-LTL.

Definition 16. *The* coSafety-LTL *logic is defined as the set of formulas obtained from* LTL *such that, when in* NNF*, do* not *contain any* R *or* G *operator.*

The coSafety-LTL logic is a co-safety fragment of LTL [178], and it is expressively complete with respect to the set of co-safety ω-languages definable in LTL [40].

Theorem 21 (Chang *et al.* [40])**.** *It holds that:*

$$[\![\mathsf{coSafety\text{-}LTL}]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$$

2.4.5 The Temporal Hierarchy

As we have already seen, two important subclasses of the set of LTL-definable languages are the sets of *safety* and *co-safety* languages. In particular, in the last section, we have seen that a LTL formula is a safety property if and only if it can be expressed in the form $\mathsf{G}\alpha$, where α is a pure-past formula of LTL+P. Similarly, an LTL formula is a co-safety property if and only if it is equivalent to $\mathsf{F}\alpha$, for some pure-past formula α of LTL+P. A natural question is to study the expressive power of the compound classes obtained either by taking Boolean combinations of formulas of type $\mathsf{G}\alpha$ and $\mathsf{F}\alpha$, or by nesting the two operators (G and F) one inside the other. The study of these compound classes brings to a *classification* of *all* the LTL-definable properties into a *hierarchy*, the so-called *Temporal Hierarchy* [140]. We will mainly focus on the work of Manna and Pnueli [140]. As noted by the authors of [140], the importance of such a classification is at least twofold. On the one hand, recognizing whether a property belongs to a class of the hierarchy (*e.g.*, the safety class) typically reduces the computational complexity of problems like satisfiability, realizability and model checking. On the other hand, the classification can be used as a remedy for *underspecification*: by having a sort of checklist partitioned into the classes of the hierarchy, the specifier can try to avoid incomplete specifications by answering to the question:

Is there a property of this class that is relevant to the system I am specifying?

Definition of the Temporal Hierarchy of LTL

We have already seen that safety properties express the fact that "a good thing will eventually happen" and co-safety properties the fact that "a good thing will happen at least one time". We now define two compound classes obtained by nesting the G and F operators.

The first class that we consider is the *liveness class*, also referred to the *recurrence class*, expressing properties of type "a good thing will happen infinitely many times". Manna and Pnuely [140] showed that the any *liveness property* can be represented by a formula of type $\mathsf{GF}\alpha$, where $\alpha \in \mathsf{LTL}{+}\mathsf{P_P}$ is a pure-past formula: this represents a normal form for the liveness class.

We consider a second class obtained by the nesting of the *globally* and the *eventually* operator, called the *persistence class*. Persistence properties are those expressing that "a good thing will constantly happen from a certain point on". The normal form of this class is represented by the formulas of type $\mathsf{FG}\alpha$, with $\alpha \in \mathsf{LTL}{+}\mathsf{P_P}$.

We consider now Boolean combinations of the classes defined above. The *obligation class* is obtained by taking any Boolean combination of safety and co-safety languages. Its name derives from the fact that its properties impose a conditional obligation. Take for instance the obligation property $\mathsf{G}\neg\alpha_1 \vee \mathsf{F}\alpha_2$ (which is equivalent to $\mathsf{F}\alpha_1 \rightarrow \mathsf{F}\alpha_2$). This property forces the fact that, if the condition α_1 will happen, then also α_2 will happen. The normal form of the obligation class is the set of formulas of type $\bigwedge_{i=1}^{n}(\mathsf{G}\alpha_i \vee \mathsf{F}\beta_i)$, where $\alpha_i, \beta_i \in \mathsf{LTL}{+}\mathsf{P_P}$, for some $n \in \mathbb{N}$. Moreover, it holds that the obligation class is strictly greater than the safety and co-safety classes, and actually it is exactly the intersection of the liveness and the persistence class.

Finally, we consider the *reactivity class*, defined as the set of any Boolean combination of liveness and persistence languages. A reactivity property, similarly to an obligation property, expresses a conditional obligation. In particular, it states that "if a good thing happens infinitely many times, so does another good thing". The normal form of this class comprises all and only the formulas of type $\bigwedge_{i=1}^{n}(\mathsf{GF}\alpha_i \vee \mathsf{FG}\beta_i)$, where $\alpha_i, \beta_i \in \mathsf{LTL}{+}\mathsf{P_P}$ and for some $n \in \mathbb{N}$.

The temporal hierachy is the hierarchy obtained from the safety, co-safety, obligation, liveness, persistence, and reactivity classes. The temporal hierarchy is depicted in Fig. 2.1 (the picture has been taken and adapted from the one in [140]).

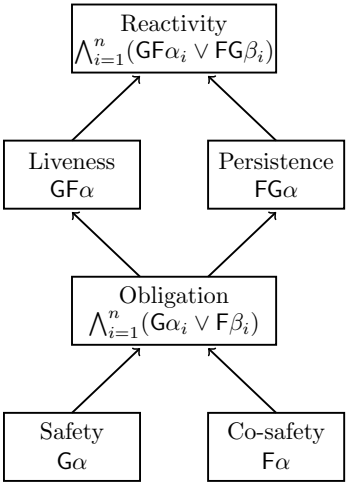


Figure 2.1: Temporal Hierarchy [140]. The picture displays the classes of the hierarchy, the relation among them along with their normal forms. The arrows represent the inclusions between the classes.

Recall from Section 2.4.3 that any LTL formula can be expressed by one of the form

$$\bigwedge_{i=1}^{n}(\mathsf{GF}(\alpha_i) \vee \neg\mathsf{GF}(\beta_i))$$

where $\alpha_i, \beta_i \in \mathsf{LTL}{+}\mathsf{P_P}$, for $i = 1, \ldots, n$ and for some $n \in \mathbb{N}$. This is exactly the normal form for the *reactivity class*. This implies that the reactivity class is exactly the set $[\![\mathsf{LTL}]\!]$. Actually, there is an entire hierarchy inside the reactivity class. We define the Reactivity(N) ($\mathsf{R(N)}$, for short) class as the reactivity class but such that we fix the number of conjuncts to be exactly N.

Definition 17 (The Reactivity(N) class)**.** *For all $n \in \mathbb{N}$, the Reactivity(N) class ($\mathsf{R(N)}$, for short) is defined as the set of formulas of the form*

$$\bigwedge_{i=1}^{n}(\mathsf{GF}(\alpha_i) \vee \mathsf{FG}(\beta_i))$$

each $\alpha_i, \beta_i \in \mathsf{LTL}{+}\mathsf{P_P}$ for each $i \in \{1, \ldots, n\}$.

It can be proved that, for each $N \in \mathbb{N}$, the Reactivity(N) class strictly contains Reactivity($N-1$) and it is strictly contained into Reactivity($N+1$).

Proposition 8. *For each $n \in \mathbb{N}$, it holds that:*

$$[\![\mathsf{R(N)}]\!] \subsetneq [\![\mathsf{R(N+1)}]\!]$$

Generalized Reactivity(1)

Bloem *et al.* [25] studied the realizability problem of a particular logic obtained from the Reactivity(1) class by generalizing the number of formulas of type GF and FG that it can contain. In particular, $\mathsf{GR(1)}$ formulas can contain conjunctions of formulas of type GF and disjunctions of formulas of type FG. For this reason, this class is called *Generalized Reactivity(1)* ($\mathsf{GR(1)}$, for short). Its syntax is defined as follows.

Definition 18 ($\mathsf{GR(1)}$ [25])**.** *The class of* Generalized Reactivity(1) *formulas (*$\mathsf{GR(1)}$*, for short) comprises all and only the formulas of type:*

$$(\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \vee (\bigvee_{i=1}^{n} \mathsf{FG}\beta_i)$$

or equivalenty of type:

$$(\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

for any $m, n \in \mathbb{N}$, where $\alpha_i, \beta_j \in \mathsf{LTL{+}P_P}$ for each $i \in \{1, \ldots, m\}$ and for each $j \in \{1, \ldots, n\}$.

Since each $\mathsf{R(1)}$ formula is also a $\mathsf{GR(1)}$ formula with only one conjunct on both sides of the implication, the following result holds.

Proposition 9. *It holds that:*

$$[\![\mathsf{R(1)}]\!] \subseteq [\![\mathsf{GR(1)}]\!]$$

Fig. 2.2 and Fig. 2.3 summarize the relation between the various formalisms discussed in this section, over finite and infinite linear order interpretation, respectively.

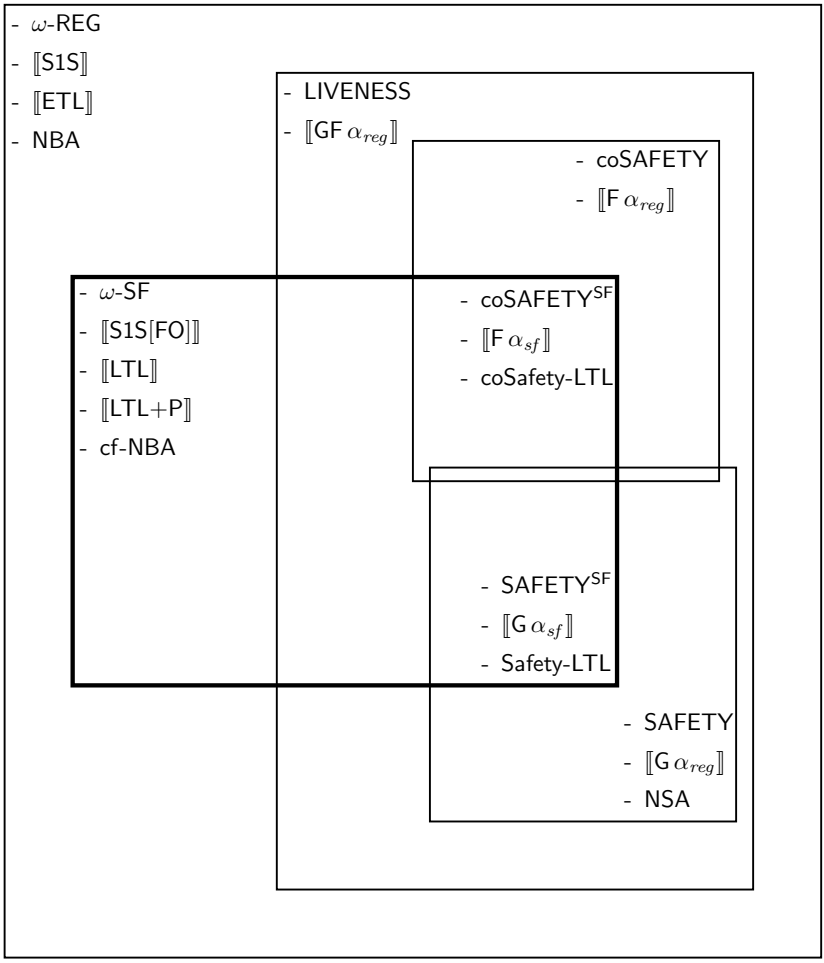


Figure 2.2: Comparison of the expressiveness of the various formalism under the *infinite* linear order interpretation. For ease of exposition, we highlighted the rectangle corresponding to LTL with thick borders.

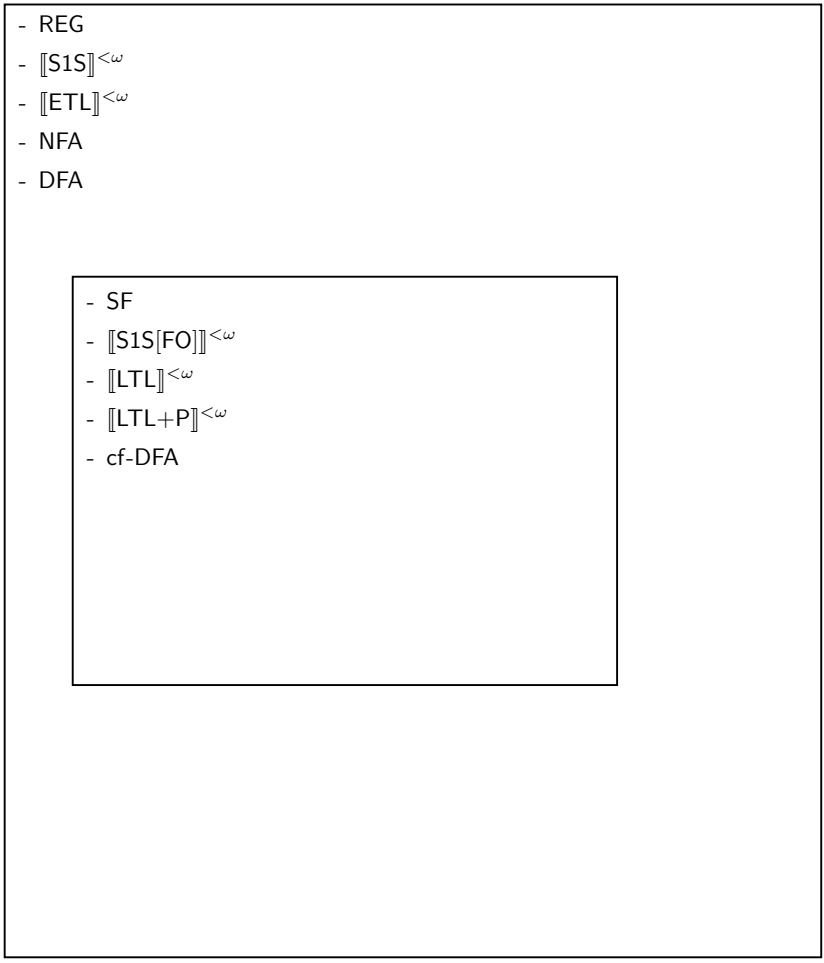


Figure 2.3: Comparison of the expressiveness of the various formalism under the *finite* linear order interpretation.

CHAPTER

3

THE $\mathsf{LTL}_{\mathsf{EBR}}$+P, $\mathsf{LTL}_{\mathsf{EBR}}$ AND GR-EBR LOGICS

In this chapter, we introduce three fragments of Linear Temporal Logic with Past (LTL+P) and we study their expressive power.

In the first section, we define the syntax of *Extended Bounded Response* LTL *with Past* ($\mathsf{LTL}_{\mathsf{EBR}}$+P, for short), obtained by a careful application of universal temporal operators (*i.e.*, X, G and R) to pure past formulas. As we will see in the next sections, the main motivation for the choices of the syntax of $\mathsf{LTL}_{\mathsf{EBR}}$+P is to allow for a fully symbolic compilation of $\mathsf{LTL}_{\mathsf{EBR}}$+P formulas into deterministic and symbolic automata. We will then investigate the expressive power of $\mathsf{LTL}_{\mathsf{EBR}}$+P and show that it is *complete* with respect to the safety fragment of LTL, *i.e.*, $[\![\mathsf{LTL}_{\mathsf{EBR}}+\mathsf{P}]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$. Actually, we will see that even a small fragment of $\mathsf{LTL}_{\mathsf{EBR}}$+P is still able to capture $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$, and we will motivate the choice for the syntax of $\mathsf{LTL}_{\mathsf{EBR}}$+P with an argument of naturalness of $\mathsf{LTL}_{\mathsf{EBR}}$+P specifications with respect to those written with the smaller fragment.

A fundamental role in the completeness proof of $\mathsf{LTL_{EBR}}$+P is played by pure past formulas. For this reason, in the second section we will study the expressive power of $\mathsf{LTL_{EBR}}$, that is the logic obtained from $\mathsf{LTL_{EBR}}$+P by fobidding past operators (and thus pure past formulas). The main result of the second section is a proof that $\mathsf{LTL_{EBR}}$ is *strictly less expressive* than $\mathsf{LTL_{EBR}}$+P. Interestingly, this shows that past operators, despite not being important for the expressiveness of full LTL (since $[\![\mathsf{LTL}]\!] = [\![\mathsf{LTL+P}]\!]$), can be crucial for the expressiveness of *fragments* of LTL, like for instance $\mathsf{LTL_{EBR}}$+P.

In the third section of this chapter, we introduce the logic of *Generalized Reactivity(1)* $\mathsf{LTL_{EBR}}$+P, or GR-EBR, for short. This logic extends $\mathsf{LTL_{EBR}}$+P by adding: (i) assumptions and guarantees (in the form of logical implications), and (ii) fairness contraints (in the form recurrence formulas). The choices underlying the syntax of GR-EBR are also guided by algorithmic motivations (as for the case of $\mathsf{LTL_{EBR}}$+P). In fact, we will see in the next chapters that the realizability problem of GR-EBR specifications can be solved in a fully symbolic way. We conclude the third section with some observations about the expressiveness of GR-EBR, which, in contrast to the case of $\mathsf{LTL_{EBR}}$+P, goes beyond the safety fragment.

3.1 The $\mathsf{LTL_{EBR}}$+P logic

In this section, we introduce the logic of *Extended Bounded Response* LTL+P ($\mathsf{LTL_{EBR}}$+P, for short) as *safety fragment* of LTL+P. In the first place, we give its syntax, which is articulated over layers. As we will see in the next chapters, each layer of the syntax corresponds to a step of a fully symbolic algorithm for solving the realizability problem from $\mathsf{LTL_{EBR}}$+P specifications. We then give a normal form for the $\mathsf{LTL_{EBR}}$+P logic and show some meaningful examples of requirements that can be formalized within this logic. We conclude this section by proving the *completeness* of $\mathsf{LTL_{EBR}}$+P with respect to the safety fragment of LTL.

3.1.1 Definition and Normal Form

To start with, we define the *bounded until* operator $\psi_1 \,\mathsf{U}^{[a,b]}\, \psi_2$ as a shortcut for the LTL formula $\bigvee_{i=a}^{b}(\mathsf{X}_1 \ldots \mathsf{X}_i(\psi_2) \wedge \bigwedge_{j=0}^{i-1} \mathsf{X}_1 \ldots \mathsf{X}_j(\psi_1))$. The *bounded eventually* ($\mathsf{F}^{[a,b]}\psi$), the *bounded globally* ($\mathsf{G}^{[a,b]}\psi$) and

the *bounded release* ($\psi_1 \mathsf{R}^{[a,b]} \psi_2$) operators derive from the bounded until in the classical way. We refer to all these four operators as *bounded future operators*. A distinguished feature of bounded future operators is that they can contrain only a finite and bounded interval in the future. This bound depends on the number of *nested next operators* inside the formula obtained by expanding the bounded operators as shown above.

Let $\mathsf{LTL{+}P_{BF}}$ (for *Bounded Future* LTL+P) be the variant of LTL+P that features past modalities and bounded future modalities, where past modalities can occur in the scope of bounded ones, but not vice versa. $\mathsf{LTL_{EBR}{+}P}$ extends $\mathsf{LTL{+}P_{BF}}$ by including Boolean combinations of the universal unbounded future modalities *globally* (G) and *release* (R).

Definition 19 (The logic $\mathsf{LTL_{EBR}{+}P}$)**.** *Let $a, b \in \mathbb{N}$. An $\mathsf{LTL_{EBR}{+}P}$ formula χ is inductively defined as follows:*

$$\eta := p \mid \neg\eta \mid \eta_1 \vee \eta_2 \mid \mathsf{Y}\eta \mid \eta_1 \,\mathsf{S}\, \eta_2 \qquad \textit{Pure Past Layer}$$

$$\psi := \eta \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{X}\psi \mid \psi_1 \,\mathsf{U}^{[a,b]}\, \psi_2 \qquad \textit{Bounded Future Layer}$$

$$\phi := \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid \psi \,\mathsf{R}\, \phi \qquad \textit{Future Layer}$$

$$\chi := \phi \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2 \qquad \textit{Boolean Layer}$$

We refer the reader to Section 2.4 for the semantics of $\mathsf{LTL_{EBR}}$ modalities.

The syntax of $\mathsf{LTL_{EBR}{+}P}$ is articulated over layers, that impose some syntactical restrictions on the formulas that can be generated from the grammar. For example, $\mathsf{LTL_{EBR}{+}P}$ forces the leftmost argument of any *release* operator to contain no universal temporal modalities (*i.e.*, R and G). As we will see in the next chapters, this layered structure is guided by the steps of the algorithm for the construction of symbolic automata starting from $\mathsf{LTL_{EBR}{+}P}$-formulas.

$\mathsf{LTL_{EBR}{+}P}$ is safety fragment of LTL+P

Since bounded future operators can be expressed by (actually they are shortcuts of) LTL+P formulas, we have that each formula of $\mathsf{LTL_{EBR}{+}P}$ is also a formula of LTL+P, and thus $\mathsf{LTL_{EBR}{+}P}$ is a *fragment* of LTL+P. Moreover, it is straightforward to see that any $\mathsf{LTL_{EBR}{+}P}$ formula contains only universal temporal modalities (*i.e.*, X, G, and R) and pure past formulas. Therefore, by Proposition 7, $\mathsf{LTL_{EBR}{+}P}$ is also a *safety* fragment of LTL+P.

A normal form for $\mathsf{LTL_{EBR}}$+P

A normal form of a logic has the big advantage to decrease the complexity of general formulas of that logic. Here, we give a normal form for $\mathsf{LTL_{EBR}}$+P which will be very useful in the chapters dedicated to algoritms, in particular when we will show a fully symbolic algorithm for realizability from $\mathsf{LTL_{EBR}}$+P specifications. The following is the normal form of the logic $\mathsf{LTL_{EBR}}$+P.

Definition 20 (Normal Form of $\mathsf{LTL_{EBR}}$+P)**.** *The* normal form *of* $\mathsf{LTL_{EBR}}$+P *is the set of all and only the formulas of the following type:*

$$\mathsf{X}^{i_1}\alpha_{i_1} \otimes \cdots \otimes \mathsf{X}^{i_j}\alpha_{i_j} \otimes$$
$$\mathsf{X}^{i_{j+1}}\mathsf{G}\alpha_{i_{j+1}} \otimes \cdots \otimes \mathsf{X}^{i_k}\mathsf{G}\alpha_{i_k} \otimes$$
$$\mathsf{X}^{i_{k+1}}(\alpha_{i_{k+1}}\ \mathsf{R}\ \beta_{i_{k+1}}) \otimes \cdots \otimes \mathsf{X}^{i_h}(\alpha_{i_h}\ \mathsf{R}\ \beta_{i_h})$$

where each $\alpha_i, \beta_i \in \mathsf{LTL{+}P_P}$ *is a pure past formula,* $\otimes \in \{\wedge, \vee\}$*, and* $i, j, k, h \in \mathbb{N}$*.*

We refer to Normal-$\mathsf{LTL_{EBR}}$+P as the set of formula of $\mathsf{LTL_{EBR}}$+P that are syntactically in normal form. Obviously, the following result holds.

Proposition 10. $[\![\mathsf{LTL_{EBR}}{+}\mathsf{P}]\!] = [\![\text{Normal-}\mathsf{LTL_{EBR}}{+}\mathsf{P}]\!]$.

The transformation of $\mathsf{LTL_{EBR}}$+P formulas to Normal-$\mathsf{LTL_{EBR}}$+P ones stands at the basis of the algorithm for solving the realizability problem of $\mathsf{LTL_{EBR}}$+P, that we will describe in the next sections. For this reason, the proof that any formula of $\mathsf{LTL_{EBR}}$+P can be transformed into an equivalent one in Normal-$\mathsf{LTL_{EBR}}$+P (*i.e.*, $[\![\mathsf{LTL_{EBR}}{+}\mathsf{P}]\!] = [\![\text{Normal-}\mathsf{LTL_{EBR}}{+}\mathsf{P}]\!]$) will be described later in the chapter dedicated to this algorithm.

3.1.2 Examples

We now give some examples of requirements that can be expressed in $\mathsf{LTL_{EBR}}$+P in a fairly natural and compact way.

The first one is a typical *bounded response* requirement. Consider an arbiter that has to assign a grant g at most k time units after a request r is issued. It can be expressed by the following $\mathsf{LTL_{EBR}}$+P formula:

$$\mathsf{G}(r \rightarrow \mathsf{F}^{[0,k]}g) \tag{3.1}$$

Another common requirement is *mutual exclusion*. As an example, the case of an arbiter that has to grant a resource to at most one client at a time can be expressed as follows (for each i, g_i means that the resource has been granted to client i):

$$\mathsf{G}(\bigwedge_{1 \leq i < j \leq n} \neg(g_i \wedge g_j)) \tag{3.2}$$

When a set of *clients with different priorities* has to be managed, it is possible to introduce a requirement stating that, whenever two or more clients simultaneously send a request, clients with a higher priority must be granted before those with a lower one ($i < j$ means that the priority of client i is higher than that of client j):

$$\bigwedge_{1 \leq i < j \leq n} \mathsf{G}((r_i \wedge r_j) \rightarrow (\neg g_j)\, \mathsf{U}^{[0,k]}\, g_i) \tag{3.3}$$

In many situations it is important to include requirements about the *configuration* of a system model. Consider the case of a thermostat. One may ask that if the `prog` modality is off, then the controller has to communicate the signal `on` to the boiler for an indefinitely long amount of time, while, in case the `prog` modality is on, it has to do that only for a specific interval of time, say $[h_1, h_2]$, after which it has to stop the communication with the boiler. This can be expressed in $\mathsf{LTL_{EBR}}$+P by the following formula:

$$(\neg \mathtt{prog} \wedge \mathsf{G}(\mathtt{on})) \vee (\mathtt{prog} \wedge \mathsf{G}^{[h_1,h_2]}(\mathtt{on}) \wedge \mathsf{X}^{h_2+1}\mathsf{G}(\mathtt{off})) \tag{3.4}$$

Finally, the use of the Past Layer in the definition of $\mathsf{LTL_{EBR}}$+P (Definition 19) enables the specification of some types of *assumptions-guarantees* properties. Consider, for instance, the class of formulas of the form $\mathsf{G}(\alpha) \rightarrow \mathsf{G}(\beta)$, where α and β are two Boolean formulas representing the assumptions for the environment and the guarantees for the controller, respectively. The specification expresses the fact that, if the environment *always* complies with its assumptions, than the controller has *always* to fulfill its guarantees. The formula $\mathsf{G}(\alpha) \rightarrow \mathsf{G}(\beta)$ does not belong to $\mathsf{LTL_{EBR}}$+P. Nevertheless, consider now the *strict implication* between the assumption α and the guarantee β [25], which corresponds to the following property: the controller fulfills its guarantees *as long as* the environment complies with its assumptions. This property can be expressed by the $\mathsf{LTL_{EBR}}$+P formula $\mathsf{G}(\mathsf{H}\alpha \rightarrow \beta)$.

3.1.3 Expressive Power

In this section, we study the expressiveness of the $\mathsf{LTL_{EBR}}$+P logic. In particular, we prove that $\mathsf{LTL_{EBR}}$+P is expressively *complete* with respect to the safety fragment of LTL+P, that is $[\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$. Consequently, by Theorem 20, $\mathsf{LTL_{EBR}}$+P and Safety-LTL are expressively equivalent (*i.e.*, $[\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!] = [\![\mathsf{Safety\text{-}LTL}]\!]$). We conclude this part showing a comparison between $\mathsf{LTL_{EBR}}$+P and other safety fragments of LTL+P.

$\mathsf{LTL_{EBR}}$+P is expressively complete

First we recall the normal form theorem stated in Theorem 16, establishing that $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY} = [\![\mathsf{G}\alpha]\!]$. Proving that $[\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$ is straightforward. The left-to-right direction follows from the fact that $\mathsf{LTL_{EBR}}$+P is a safety fragment of LTL+P (recall Section 3.1.1). For the right-to-left direction it suffices to show that the normal form $\mathsf{G}\alpha$ is syntactically definable in $\mathsf{LTL_{EBR}}$+P (*i.e.*, $\mathsf{G}\alpha \in \mathsf{LTL_{EBR}}+\mathsf{P}$ and thus also $\mathcal{L}(\mathsf{G}\alpha) \in [\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!]$, for any $\alpha \in \mathsf{LTL}+\mathsf{P_P}$).

Theorem 22 (Completeness of $\mathsf{LTL_{EBR}}$+P)**.** *It holds that:*

$$[\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$$

Proof. We first prove that $[\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!] \subseteq [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$. Let $\phi \in [\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!]$. By Definition 19, $\phi \in \mathsf{LTL}+\mathsf{P}$, and thus, since $[\![\mathsf{LTL}]\!] = [\![\mathsf{LTL}+\mathsf{P}]\!]$, it holds that $\mathcal{L}(\phi) \in [\![\mathsf{LTL}]\!]$. Moreover, by Proposition 7, $\mathcal{L}(\phi)$ is a safety language, that is $\mathcal{L}(\phi) \in \mathsf{SAFETY}$. Therefore, $\mathcal{L}(\phi) \in [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$.

We now prove that $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY} \subseteq [\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!]$. Let ϕ be a formula such that $\mathcal{L}(\phi) \in [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$. By Theorem 16, $\mathcal{L}(\phi) \in [\![\mathsf{G}\alpha]\!]$. Now, $\mathsf{G}\alpha$ (for any $\alpha \in \mathsf{LTL}+\mathsf{P_P}$) is a formula that syntactically belongs to $\mathsf{LTL_{EBR}}$+P, that is $\mathsf{G}\alpha \in \mathsf{LTL_{EBR}}+\mathsf{P}$, and thus $[\![\mathsf{G}\alpha]\!] \subseteq [\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!]$. It follows that $\mathcal{L}(\phi) \in [\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!]$. □

Comparison between $\mathsf{LTL_{EBR}}$+P, $\mathsf{G}\alpha$ and Safety-LTL

In this part, we compare the $\mathsf{LTL_{EBR}}$+P logic with the $\mathsf{G}\alpha$ normal form and with the Safety-LTL logic.

Previously, we proved that the set of languages definable in $\mathsf{LTL_{EBR}}$+P is exactly the set of safety languages definable in LTL+P.

In turn, Theorem 16 shows that these sets correspond to languages definable by a formula of type $\mathsf{G}\alpha$, where $\alpha \in \mathsf{LTL}{+}\mathsf{P_P}$. Despite being equivalent fragments, we think that $\mathsf{LTL_{EBR}}$+P offers a more natural language for safety properties than the $\mathsf{G}\alpha$ fragment. Consider for example the following property, expressed in natural language: either p_3 holds forever, or there exists two time points $t' \leq t$ such that (i) p_1 holds in t, (ii) p_2 holds in t', and (iii) p_2 holds from time point 0 to t. The property can be easily formalized in $\mathsf{LTL_{EBR}}$+P by the formula $p_1 \,\mathsf{R}\, (p_2 \,\mathsf{R}\, p_3)$. The equivalent formula in the $\mathsf{G}\alpha$ fragment is $\mathsf{G}(\mathsf{H}(p_3) \vee \mathsf{O}(p_2 \wedge \mathsf{O}(p_1) \wedge \mathsf{H}(p_3)))$, which is arguably more intricate.

Safety-LTL is the fragment of LTL (thus with only future temporal modalities) containing all and only the LTL-formulas that, when in negated normal form, do *not* contain any *until* or *eventually* operator (recall Definition 15). Recall from Theorem 20 that Safety-LTL captures exactly the safety fragment of $\mathsf{LTL}{+}\mathsf{P}$, *i.e.*, $[\![\text{Safety-LTL}]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$. It immediately follows that $\mathsf{LTL_{EBR}}$+P and Safety-LTL are expressively equivalent, namely $[\![\mathsf{LTL_{EBR}}{+}\mathsf{P}]\!] = [\![\text{Safety-LTL}]\!]$.

Differently from $\mathsf{LTL_{EBR}}$+P, Safety-LTL does *not* impose any syntactic restriction on the nesting of the logical operators; as a matter of fact, $\mathsf{G}(p_1 \vee \mathsf{G}p_2)$ belongs to the syntax of Safety-LTL but not to the syntax of $\mathsf{LTL_{EBR}}$+P, even though $\mathsf{G}(p_1 \vee \mathsf{G}p_2) \equiv \mathsf{G}(\neg p_2 \rightarrow \mathsf{H}p_1) \in \mathsf{LTL_{EBR}}{+}\mathsf{P}$. The restrictions on the syntax of $\mathsf{LTL_{EBR}}$+P are due to algorithimic aspects: each layer of the syntax of $\mathsf{LTL_{EBR}}$+P (recall Definition 19) corresponds to a step of the algorithm for the symbolic automata construction starting from $\mathsf{LTL_{EBR}}$+P-formulas. As a matter of fact, we will see in the chapters dedicated to algorothms that, in practice, $\mathsf{LTL_{EBR}}$+P avoids an exponential blowup in time with respect to known algorithms for automata contruction for safety specifications. Last but not least, we will prove that the realizability problem of $\mathsf{LTL_{EBR}}$+P is EXPTIME-complete, as opposed to the realizability of $\mathsf{LTL}{+}\mathsf{P}$, which is 2EXPTIME-complete [160, 164]. Consider now $\mathsf{LTL_{EBR}}$, that is the fragment of $\mathsf{LTL_{EBR}}$+P devoid of past operators. Since each formula of $\mathsf{LTL_{EBR}}$ syntactically belongs to Safety-LTL, it immediately follows that $[\![\mathsf{LTL_{EBR}}]\!] \subseteq [\![\text{Safety-LTL}]\!]$. In the next section, we will prove that the converse direction does *not* hold, that is $\mathsf{LTL_{EBR}}$ is *strictly* less expressive than $\mathsf{LTL_{EBR}}$+P, and thus less expressive than Safety-LTL as well.

3.2 The $\mathsf{LTL_{EBR}}$ logic

In the previous section, we showed that $\mathsf{LTL_{EBR}}$+P is expressively complete with respect to the safety fragment of LTL. A crucial role in the proof is played by the *pure past* layer of $\mathsf{LTL_{EBR}}$+P. In particular, since the $\mathsf{G}\alpha$ class is a normal form for the safety fragment of LTL, and since $\mathsf{G}\alpha$ is definable in $\mathsf{LTL_{EBR}}$ (for any $\alpha \in \mathsf{LTL{+}P_P}$), we have that $\mathsf{LTL_{EBR}}$+P capture exactly the safety fragment of LTL. In this section, we investigate whether the same holds also for the logic obtained from $\mathsf{LTL_{EBR}}$+P by forbidding past temporal modalities. In fact, one may wonder whether the pure past layer is really necessary, or whether the class $[\![\mathsf{G}\alpha]\!]$ can be expressed in $\mathsf{LTL_{EBR}}$+P without the use of past operators. We will give a *negative* answer to this question, showing that $\mathsf{LTL_{EBR}}$+P without past operators is *strictly less expressive* than $\mathsf{LTL_{EBR}}$+P.

3.2.1 Definition and Normal Form

We define the $\mathsf{LTL_{EBR}}$ logic as $\mathsf{LTL_{EBR}}$+P devoid of past temporal operators.

Definition 21 (The logic $\mathsf{LTL_{EBR}}$)**.** *Let $a, b \in \mathbb{N}$. An $\mathsf{LTL_{EBR}}$ formula χ is inductively defined as follows:*

$$\psi := p \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{X}\psi \mid \psi_1 \,\mathsf{U}^{[a,b]}\, \psi_2 \qquad \textit{Bounded Future Layer}$$
$$\phi := \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid \psi \,\mathsf{R}\, \phi \qquad \textit{Future Layer}$$
$$\chi := \phi \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2 \qquad \textit{Boolean Layer}$$

Before giving the normal form of $\mathsf{LTL_{EBR}}$, we define the $\mathsf{LTL{+}P_{BP}}$ set. A *bounded past* formula is any Boolean combination of propositional atoms and formulas of type $\mathsf{Y}\alpha$, where α is a bounded past formula. We refer to $\mathsf{LTL{+}P_{BP}}$ (*Bounded Past* LTL+P) as the set of all and only the bounded past formulas of LTL+P. Equivalently, $\mathsf{LTL{+}P_{BP}}$ is the set obtained from $\mathsf{LTL{+}P_P}$ by forbidding the *since* operator.

The absence of past modalities in $\mathsf{LTL_{EBR}}$ changes its normal form with respect to the one of $\mathsf{LTL_{EBR}}$+P. In particular, while the normal form of $\mathsf{LTL_{EBR}}$+P can have arbitrarily pure past formulas as arguments of *globally* and *release* operators (recall Definition 20), the normal form of $\mathsf{LTL_{EBR}}$ can have only *bounded past* formulas as such arguments.

Definition 22 (Normal Form of $\mathsf{LTL_{EBR}}$)**.** *The* normal form *of* $\mathsf{LTL_{EBR}}$+P *is the set of all and only the formulas of the following type:*

$$\begin{gathered}\mathsf{X}^{i_1}\alpha_{i_1} \otimes \cdots \otimes \mathsf{X}^{i_j}\alpha_{i_j} \otimes \\ \mathsf{X}^{i_{j+1}}\mathsf{G}\alpha_{i_{j+1}} \otimes \cdots \otimes \mathsf{X}^{i_k}\mathsf{G}\alpha_{i_k} \otimes \\ \mathsf{X}^{i_{k+1}}(\alpha_{i_{k+1}}\,\mathsf{R}\,\beta_{i_{k+1}}) \otimes \cdots \otimes \mathsf{X}^{i_h}(\alpha_{i_h}\,\mathsf{R}\,\beta_{i_h})\end{gathered}$$

where each $\alpha_i, \beta_i \in \mathsf{LTL{+}P_{BP}}$ *is a bounded past formula,* $\otimes \in \{\wedge, \vee\}$*, and* $i, j, k, h \in \mathbb{N}$*.*

We call $\mathsf{Normal\text{-}LTL_{EBR}}$ the set of $\mathsf{LTL_{EBR}}$ formulas in normal form. It holds that:

Lemma 1. $[\![\mathsf{LTL_{EBR}}]\!] = [\![\mathsf{Normal\text{-}LTL_{EBR}}]\!]$.

As before, the proof that $[\![\mathsf{LTL_{EBR}}]\!] = [\![\mathsf{LTL_{EBR}{+}P}]\!]$ is postponed to the chapters dedicated to algorithms, since it is the same of the one used for normalizing $\mathsf{LTL_{EBR}}$+P formulas.

3.2.2 Expressive power

In the previous sections, we have seen that:

$$[\![\mathsf{LTL_{EBR}{+}P}]\!] = [\![\mathsf{G}\alpha]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY} = [\![\mathsf{Safety\text{-}LTL}]\!]$$

In particular, thanks to the use of the *pure past layer* (recall Definition 21), $\mathsf{LTL_{EBR}}$+P can easily capture the whole class of $[\![\mathsf{G}\alpha]\!]$, and thus the whole class of $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$. However, one may wonder whether the pure past layer is really necessary, or whether the class $[\![\mathsf{G}\alpha]\!]$ can be expressed in $\mathsf{LTL_{EBR}}$+P without the use of past operators. We will prove that this is *not* the case, that is

$$[\![\mathsf{LTL_{EBR}}]\!] \subsetneq [\![\mathsf{LTL_{EBR}{+}P}]\!] \tag{3.5}$$

This result proves that *past modalities*, although being not important for the expressiveness of full LTL (since $[\![\mathsf{LTL}]\!] = [\![\mathsf{LTL{+}P}]\!]$ [97, 136, 143]), can play a crucial role for the expressive power of *fragments* of LTL, like, for instance, $\mathsf{LTL_{EBR}}$.

The general idea

We will prove Eq. (3.5) by showing that $[\![\mathsf{LTL_{EBR}}]\!] \subsetneq [\![\mathsf{Safety\text{-}LTL}]\!]$. The result in Eq. (3.5) follows from the fact that $[\![\mathsf{Safety\text{-}LTL}]\!] = [\![\mathsf{LTL_{EBR}{+}P}]\!]$. We will prove that the language of the Safety-LTL formula $\varphi_{\mathsf{G}} \coloneqq \mathsf{G}(p_1 \vee \mathsf{G}(p_2))$ cannot be expressed by any $\mathsf{LTL_{EBR}}$ formula. The formula φ_{G} belongs *syntactically* to Safety-LTL, and thus $\mathcal{L}(\varphi_{\mathsf{G}}) \in [\![\mathsf{Safety\text{-}LTL}]\!]$. We remark that, despite φ_{G} does not syntactically belong to $\mathsf{LTL_{EBR}}$+P, its language can be formalized in $\mathsf{LTL_{EBR}}$+P. In fact, it holds that:

$$\mathsf{G}(p_1 \vee \mathsf{G}(p_2)) \equiv \mathsf{G}(\neg p_2 \rightarrow \mathsf{H}(p_1)) \tag{3.6}$$

Since $\mathsf{G}(\neg p_2 \rightarrow \mathsf{H}(p_1)) \in \mathsf{LTL_{EBR}{+}P}$, it holds that $\mathcal{L}(\varphi_{\mathsf{G}}) \in [\![\mathsf{LTL_{EBR}{+}P}]\!]$. It is worth noting the following points: (i) $\mathsf{G}(\neg p_2 \rightarrow \mathsf{H}(p_1))$ is of the form $\mathsf{G}\alpha$, where $\alpha \in \mathsf{LTL{+}P_P}$ (α is a pure past formula); (ii) the formula φ_{G} is equivalent to $\mathsf{G}(b) \vee ((\mathsf{XG}p_2)\,\mathsf{R}\,p_1)$, but this formulation does not syntactically belong to $\mathsf{LTL_{EBR}}$, due to the restriction that forces the leftmost argument of any *release* operator to contain no universal temporal operators (*i.e.*, R and G). In fact, in the following, we will prove that $\mathcal{L}(\varphi_{\mathsf{G}}) \notin [\![\mathsf{LTL_{EBR}}]\!]$.

The proof of the undefinability of φ_{G} is based on the fact that each formula of $\mathsf{LTL_{EBR}}$ cannot constrain an arbitrarily long prefix of a state sequence, but only a finite prefix whose maximum length depends on the maximum number of nested *next* operators inside the $\mathsf{LTL_{EBR}}$ formula.

Consider again the formula $\varphi_{\mathsf{G}} \coloneqq \mathsf{G}(p_1 \vee \mathsf{G}(p_2))$. The language $\mathcal{L}(\varphi_{\mathsf{G}})$ is expressed by the ω-regular expression $(\{p_1\})^\omega + (\{p_1\})^* \cdot (\{p_2\})^\omega$. Written in natural language, each model of φ_{G} cannot contain a position in which $\neg p_2$ holds preceded by a position in which $\neg p_1$ holds.

Remark 1. *Let $\sigma \subseteq (2^\Sigma)^\omega$ be a state sequence. It holds that:*

$$\sigma \models \varphi_{\mathsf{G}} \;\Rightarrow \neg\exists i, j (j \leq i \wedge \sigma_j \models \neg p_1 \wedge \sigma_i \models \neg p_2)$$

We define ${}^{i,k}\sigma^j$ as *the* state sequence such that at the time points i and k it holds $p_1 \wedge \neg p_2$, at time point j it holds $\neg p_1 \wedge p_2$, and for all the other time points $p_1 \wedge p_2$ holds. The membership of ${}^{i,k}\sigma^j$ to $\mathcal{L}(\varphi_{\mathsf{G}})$ depends on the value of the three indices i, j and k, as follows.

Remark 2. *If $i < j$ and $k < j$, then ${}^{i,k}\sigma^{j} \models \varphi_{\mathsf{G}}$. Conversely, if $i \geq j$ or $k \geq j$, then ${}^{i,k}\sigma^{j} \not\models \varphi_{\mathsf{G}}$.*

As we will see, given a generic formula $\psi \in \mathsf{LTL_{EBR}}$, one can always find some values for the indices i, j and k such that (a) j is chosen sufficiently greater than i; (b) k is chosen sufficiently greater than j; (c) ψ is not able to distinguish the state sequence ${}^{i,i}\sigma^{j}$ from ${}^{i,k}\sigma^{j}$. Since, by Remark 2, ${}^{i,i}\sigma^{j} \in \mathcal{L}(\varphi_{\mathsf{G}})$ but ${}^{i,k}\sigma^{j} \notin \mathcal{L}(\varphi_{\mathsf{G}})$, this proves the undefinability of φ_{G} in $\mathsf{LTL_{EBR}}$. The rationale is that the $\mathsf{LTL_{EBR}}$ logic combines bounded future formulas (*i.e.*, formulas obtained by a Boolean combination of propositional atoms and X operators) and universal temporal operators (*i.e.*, G and R). This implies that, for a generic model σ of an $\mathsf{LTL_{EBR}}$-formula ψ, *at each time point* $i \geq 0$ of σ (this corresponds to the universal temporal operators) only a *finite and bounded suffix* after i (this corresponds to the $\mathsf{LTL{+}P_{BF}}$-formulas) can be constrained by ψ (this can be thought of as a sort of bounded memory property of this logic). Equivalently, this means that each $\mathsf{LTL_{EBR}}$-formula is *not* able to constrain any finite but arbitrarly long (unbounded) prefix of a state sequence, contrary, for instance, to the case of the formula $\mathsf{G}(\neg p_2 \rightarrow \mathsf{H}(p_1))$ (that is equivalent to φ_{G}, see Eq. (3.6)).

The Role of the Normal Form

The limitation of $\mathsf{LTL_{EBR}}$-formulas mentioned before is more evident in the *normal form* for the $\mathsf{LTL_{EBR}}$ logic (recall Definition 22). We first give some preliminaries definitions. Recall that $\mathsf{LTL{+}P_{BP}}$ (*Bounded Past* $\mathsf{LTL{+}P_{P}}$) is the set of all and only the $\mathsf{LTL_{EBR}{+}P}$ formulas that are a Boolean combination of propositional atoms and *yesterday* operators (Y). We use the shortcut $\psi_1 \, \mathsf{S}^{[a,b]} \, \psi_2$ for denoting the formula $\bigvee_{i=a}^{b}(\mathsf{Y}_1 \ldots \mathsf{Y}_i(\psi_2) \wedge \bigwedge_{j=0}^{i-1} \mathsf{Y}_1 \ldots \mathsf{Y}_j(\psi_1))$. Given a formula $\alpha \in \mathsf{LTL{+}P_{BP}}$, we define its *temporal depth*, denoted as $D(\alpha)$, as follows:

- $D(p) = 0$, for all $p \in \Sigma$
- $D(\neg\alpha_1) = D(\alpha_1)$
- $D(\alpha_1 \wedge \alpha_2) = \max\{D(\alpha_1), D(\alpha_2)\}$
- $D(\mathsf{Y}\alpha_1) = 1 + D(\alpha_1)$
- $D(\alpha_1 \, \mathsf{S}^{[a,b]} \, \alpha_2) = b + \max\{D(\alpha_1), D(\alpha_2)\}$

For each $\alpha \in \mathsf{LTL}{+}\mathsf{P_{BP}}$, the language $\mathcal{L}^{<\omega}(\alpha)$ consists only of words of length at most $D(\alpha)+1$. Recall from Section 2.4 that, given a infinite state sequence $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ and some $n \geq 0$, $\sigma_{[n-d,n]}$ is the interval of σ of length *at most* d ending at index n: $\sigma_{[n-d,n]} = \langle \sigma_{n-d}, \ldots, \sigma_n \rangle$ if $n \geq d$, or $\sigma_{[n-d,n]} = \langle \sigma_0, \ldots, \sigma_n \rangle$ otherwise. The crucial property of $\mathsf{LTL}{+}\mathsf{P_{BP}}$-formulas, that can be shown with a simple induction, is that their truth over a state sequence σ can be checked by considering only a finite and *bounded* interval of σ, whose length depends on the *temporal depth* of the formula.

Remark 3. *For any $\alpha \in \mathsf{LTL}{+}\mathsf{P_{BP}}$, with temporal depth $d = D(\alpha)$, and for any $n \geq 0$, it holds that $\sigma, n \models \alpha$ if and only if $\sigma_{[n-d,n]} \models \alpha$.*

Recall from Definition 22 the definition of $\mathsf{Normal\text{-}LTL_{EBR}}$. The normal form makes it easier to prove Eq. (3.5). In particular, the fact that in the normal form we have no nested universal temporal operators dramatically simplifies the proof by induction. Take for example the formula $\mathsf{XXG}(p \vee \mathsf{Y}p \vee \mathsf{YY}p)$, that belongs to $\mathsf{Normal\text{-}LTL_{EBR}}$. It is clear that, at each time point, this formula can constrain only the interval consisting of the current state and its two previous states (in fact its temporal depth is 3).

The main proof

In this part, we show the undefinability of the formula φ_{G} in the $\mathsf{Normal\text{-}LTL_{EBR}}$ logic. The undefinability in $\mathsf{LTL_{EBR}}$ follows from Lemma 1.

Given three indices $i, j, k \in \mathbb{N}$ such that $i \neq j$ and $k \neq j$, we formally define the state sequence ${}^{i,k}\sigma^j = \langle {}^{i,k}\sigma^j_0,\ {}^{i,k}\sigma^j_1, \ldots \rangle$ as follows:

$$ {}^{i,k}\sigma^j_h = \begin{cases} \{p_1\} & \text{if } h \in \{i,k\} \\ \{p_2\} & \text{if } h = j \\ \{p_1, p_2\} & \text{otherwise} \end{cases} $$

The core of the main theorem is based on the fact that any formula of type $\mathsf{G}\alpha$ or $\alpha \mathsf{R} \beta$, where α and β are *bounded past* $\mathsf{LTL}{+}\mathsf{P_P}$ formulas, is not able to distinguish the state sequence ${}^{i,i}\sigma^j$ with $i < j$ (which is a model of φ_{G}) from ${}^{i,k}\sigma^j$ with $k > j$ (which is *not* a model of φ_{G}), for sufficiently large values of i, j and k. The choice for the values of the three indices is based on the values of the *temporal depth* of α and β. Since the *globally* operator is a special case of the *release*

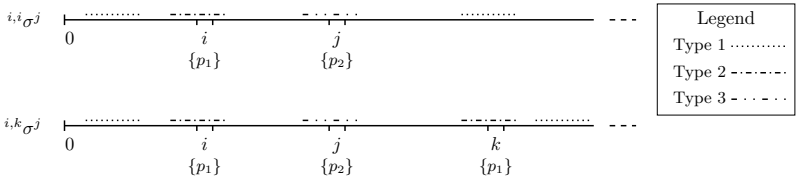


Figure 3.1

operator, that is $\mathsf{G}\alpha \equiv \bot \mathsf{R}\, \alpha$, it suffices to prove the property for formulas of type $\alpha \mathsf{R} \beta$. We first prove the two fundamental properties that show that, for any interval of $^{i,i}\sigma^j$ of length at most d (for any $d \in \mathbb{N}$), we can find the exact same interval in $^{i,k}\sigma^j$, and *vice versa*. Fig. 3.1 shows the idea of this correspondence.

Lemma 2. *Let $d \in \mathbb{N}$. For all $i \geq d$, for all $j \geq i + d$, and for all $k \geq j + d$, it holds that:*

$$\textit{Property 1: } \forall n' \geq 0 \,.\, \exists n \geq 0 \,.\, {}^{i,k}\sigma^j_{[n'-d,n']} = {}^{i,i}\sigma^j_{[n-d,n]}$$

$$\textit{Property 2: } \forall n \geq 0 \,.\, \exists n' \geq 0 \,.\, {}^{i,i}\sigma^j_{[n-d,n]} = {}^{i,k}\sigma^j_{[n'-d,n']}$$

Proof. Take any value for i, j, and k such that: (i) $i \geq d$, (ii) $j \geq i+d$, (iii) $k \geq j + d$. Given any interval of length d of the state sequence $^{i,i}\sigma^j$, we show how to find an exact same one in $^{i,k}\sigma^j$, and viceversa.

The constraints above on the three indices ensure that both the state sequences $^{i,i}\sigma^j$ and $^{i,k}\sigma^j$ contain *only three* types of intervals of length at most d. Consider $^{i,k}\sigma^j$ (the case for $^{i,i}\sigma^j$ is specular). The three types are the following:

Type 1: $(\{p_1, p_2\})^n$ for some $0 \leq n \leq d$;

Type 2: $(\{p_1, p_2\})^n \cdot (\{p_1\}) \cdot (\{p_1, p_2\})^{d-n-1}$, for some $0 \leq n < d$;

Type 3: $(\{p_1, p_2\})^n \cdot (\{p_2\}) \cdot (\{p_1, p_2\})^{d-n-1}$, for some $0 \leq n < d$;

The situation is depicted in Fig. 3.1. Given any interval of any of the three types above, we show below how to find the very same interval in $^{i,i}\sigma^j$ (Fig. 3.1 tries to show visually this correspondence):

- each interval of $^{i,k}\sigma^j$ of type $(\{p_1, p_2\})^n$ is equal to $^{i,i}\sigma^j_{[0,n]}$;
- each interval of $^{i,k}\sigma^j$ of type $(\{p_1, p_2\})^n \cdot (\{p_1\}) \cdot (\{p_1, p_2\})^{d-n-1}$ is equal to $^{i,i}\sigma^j_{[i-n,i+d-n-1]}$.

- each interval of ${}^{i,k}\sigma^j$ of type $(\{p_1,p_2\})^n\cdot(\{p_2\})\cdot(\{p_1,p_2\})^{d-n-1}$ is equal to ${}^{i,i}\sigma^j_{[j-n,j+d-n-1]}$;

This proves *Property 1.*

Similarly, the correspondence between intervals of ${}^{i,i}\sigma^j$ and intervals of ${}^{i,k}\sigma^j$ is the following:

- each interval of ${}^{i,i}\sigma^j$ of type $(\{p_1,p_2\})^n$ is equal to ${}^{i,k}\sigma^j_{[0,n]}$;
- each interval of ${}^{i,i}\sigma^j$ of type $(\{p_1,p_2\})^n\cdot(\{p_1\})\cdot(\{p_1,p_2\})^{d-n-1}$ is equal to ${}^{i,k}\sigma^j_{[i-n,i+d-n-1]}$.
- each interval of ${}^{i,i}\sigma^j$ of type $(\{p_1,p_2\})^n\cdot(\{p_2\})\cdot(\{p_1,p_2\})^{d-n-1}$ is equal to ${}^{i,k}\sigma^j_{[j-n,j+d-n-1]}$;

This proves *Property 2.* □

We can now prove that the state sequences ${}^{i,i}\sigma^j$ and ${}^{i,k}\sigma^j$ are indistinguishable for each formula of type $\alpha\,\mathsf{R}\,\beta$ (and, consequently, of type $\mathsf{G}\alpha$), with $\alpha,\beta\in\mathsf{LTL{+}P_{BP}}$.

Lemma 3. *Let $\alpha,\beta\in\mathsf{LTL{+}P_{BP}}$, and let $d=\max\{D(\alpha),D(\beta)\}$ be the maximum between the temporal depths of α and β. It holds that ${}^{i,i}\sigma^j\models\alpha\,\mathsf{R}\,\beta$ iff ${}^{i,k}\sigma^j\models\alpha\,\mathsf{R}\,\beta$, for all $i\geq d$, for all $j\geq i+d$, and for all $k\geq j+d$.*

Proof. Take any value for i, j, and k such that: (i) $i\geq d$, (ii) $j\geq i{+}d$, (iii) $k\geq j+d$.

We first prove the left-to-right direction. Suppose that ${}^{i,i}\sigma^j\models\alpha\,\mathsf{R}\,\beta$. We divide in cases:

1. Suppose that ${}^{i,i}\sigma^j,n\models\beta$ for all $n\geq 0$. Since $\beta\in\mathsf{LTL{+}P_{BP}}$ and $D(\beta)\leq d$, it holds that ${}^{i,i}\sigma^j_{[n-d,n]}\models\beta$, for all $n\geq 0$. Suppose by contradiction that there exists some $n'\geq 0$ such that ${}^{i,k}\sigma^j_{[n'-d,n']}\models\neg\beta$. By *Property 1* of Lemma 2, this means that there exists some $n''\geq 0$ such that ${}^{i,i}\sigma^j_{[n''-d,n'']}\models\neg\beta$. But this is a contradiction. Thus, it holds that ${}^{i,k}\sigma^j_{[n'-d,n']}\models\beta$ for all $n'\geq 0$, that is, for all $n'\geq 0$, and thus ${}^{i,k}\sigma^j\models\alpha\,\mathsf{R}\,\beta$.
2. Suppose that $\exists n\geq 0$. $({}^{i,i}\sigma^j,n\models\alpha\wedge\forall 0\leq m\leq n$. ${}^{i,i}\sigma^j,m\models\beta)$. We divide again in cases:

(a) Suppose that $n < k$. Then ${}^{i,i}\sigma^j_{[0,n]} = {}^{i,k}\sigma^j_{[0,n]}$. Clearly, it holds that ${}^{i,k}\sigma^j, n \models \alpha$ and ${}^{i,k}\sigma^j, m \models \beta$ for all $0 \leq m \leq n$. Therefore ${}^{i,k}\sigma^j \models \alpha \,\mathsf{R}\, \beta$.

(b) Suppose that $n \geq k$. In particular, it holds that ${}^{i,i}\sigma^j_{[n-d,n]} \models \alpha \wedge \beta$. We use a *contraction argument* for proving that in this case there exists a smaller index at which the *release* satisfies its existential part (*i.e.*, the formula α). Consider the time point $i-1$. It holds that ${}^{i,i}\sigma^j_{[i-1-d,i-1]} = {}^{i,i}\sigma^j_{[n-d,n]}$ and thus, since ${}^{i,i}\sigma^j_{[n-d,n]} \models \alpha \wedge \beta$ and $\alpha, \beta \in \mathsf{LTL{+}P_{BP}}$, we have that ${}^{i,i}\sigma^j_{[i-1-d,i-1]} \models \alpha \wedge \beta$. Moreover, ${}^{i,i}\sigma^j_{[0,i-1]}$ is a prefix of ${}^{i,i}\sigma^j_{[0,n]}$, and thus, given that ${}^{i,i}\sigma^j_{[p-d,p]} \models \beta$ for all $0 \leq p \leq n$, it holds that ${}^{i,i}\sigma^j_{[p-d,p]} \models \beta$ for all $0 \leq p \leq i-1$. From this, it follows that ${}^{i,i}\sigma^j, i-1 \models \alpha$ and ${}^{i,i}\sigma^j, m \models \beta$ for all $0 \leq m \leq i-1$. Since $i - 1 < k$, by Item 2a, it holds that ${}^{i,k}\sigma^j \models \alpha \,\mathsf{R}\, \beta$.

We now prove the right-to-left direction. Suppose that ${}^{i,k}\sigma^j \models \alpha \,\mathsf{R}\, \beta$. We divide in cases:

1. Suppose that ${}^{i,k}\sigma^j, n \models \beta$. This case is specular to Item 1.

2. Suppose that $\exists n \geq 0 \,.\, ({}^{i,k}\sigma^j, n \models \alpha \wedge \forall 0 \leq m \leq n \,.\, {}^{i,k}\sigma^j, m \models \beta)$. Since $\alpha, \beta \in \mathsf{LTL{+}P_{BP}}$ and $D(\alpha), D(\beta) \leq d$, it holds that $\exists n \geq 0 \,.\, ({}^{i,k}\sigma^j_{[n-d,n]} \models \alpha \wedge \forall 0 \leq m \leq n \,.\, {}^{i,k}\sigma^j_{[m-d,m]} \models \beta)$. We divide again in cases:

   (a) If $n < k$, then ${}^{i,k}\sigma^j_{[0,n]} = {}^{i,i}\sigma^j_{[0,n]}$ and thus ${}^{i,i}\sigma^j, n \models \alpha$ and ${}^{i,i}\sigma^j, m \models \beta$ for all $0 \leq m \leq n$, that is ${}^{i,i}\sigma^j \models \alpha \,\mathsf{R}\, \beta$.

   (b) If $k \leq n \leq k + d$, then ${}^{i,k}\sigma^j_{[n-d,n]} = {}^{i,k}\sigma^j_{[n-k-i-d,n-k-i]}$ (we used again a contraction argument). Since by hypothesis ${}^{i,k}\sigma^j_{[n-d,n]} \models \alpha$, it holds also that ${}^{i,k}\sigma^j_{[n-k-i-d,n-k-i]} \models \alpha$. Moreover, ${}^{i,k}\sigma^j_{[0,n-k-i]}$ is a prefix of ${}^{i,k}\sigma^j_{[0,n]}$, and thus, since by hypothesis ${}^{i,k}\sigma^j_{[p-d,p]} \models \beta$ for all $0 \leq p \leq n$, it also holds that ${}^{i,k}\sigma^j_{[p-d,p]} \models \beta$ for all $0 \leq p \leq n-k-i$. Therefore ${}^{i,k}\sigma^j_{[n-k-i-d,n-k-i]} \models \alpha$ and ${}^{i,k}\sigma^j_{[m-d,m]} \models \beta$ for all $0 \leq m \leq n-k-i$. Since $l + n - i < k$, by Item 2a, it holds that ${}^{i,i}\sigma^j \models \alpha \,\mathsf{R}\, \beta$.

(c) Otherwise $n > k + d$. We have that ${}^{i,k}\sigma^{j}_{[n-d,n]}$ $= {}^{i,k}\sigma^{j}_{[i-1,i-1-d]}$ (also in this case we used a contraction argument). Since by hypothesis ${}^{i,k}\sigma^{j}_{[n-d,n]} \models \alpha$, it also hold that ${}^{i,k}\sigma^{j}_{[i-1,i-1-d]} \models \alpha$. Moreover ${}^{i,k}\sigma^{j}_{[0,i-1]}$ is a prefix of ${}^{i,k}\sigma^{j}_{[0,n]}$ and thus, since by hypothesis ${}^{i,k}\sigma^{j}_{[p-d,p]} \models \beta$ for all $0 \leq p \leq n$, it also holds that ${}^{i,k}\sigma^{j}_{[p-d,p]} \models \beta$ for all $0 \leq p \leq i-1$. Therefore ${}^{i,k}\sigma^{j}, i-1 \models \alpha$ and ${}^{i,k}\sigma^{j}, m \models \beta$ for all $0 \leq m \leq i-1$. Since $i-1 < k$, by Item 2a, it holds that ${}^{i,i}\sigma^{j} \models \alpha \,\mathsf{R}\, \beta$.

□

By using Lemma 3 as the proof for the base case, we prove by induction on the structure of the formula that any formula in $\mathsf{Normal\text{-}LTL_{EBR}}$ is not able to distinguish the state sequences ${}^{i,i}\sigma^{j}$ and ${}^{i,k}\sigma^{j}$ for sufficiently large values of i, j, k. In the following, given a formula $\psi \in \mathsf{Normal\text{-}LTL_{EBR}}$, we will denote with m_ψ the maximum number of nested *next* operators in ψ, and with d_ψ the maximum temporal depth between all its $\mathsf{LTL{+}P_{BP}}$-subformulas.

Lemma 4. *Let $\psi \in \mathsf{Normal\text{-}LTL_{EBR}}$. It holds that ${}^{i,i}\sigma^{j} \models \psi$ iff ${}^{i,k}\sigma^{j} \models \psi$, for all $i \geq m_\psi + d_\psi$, for all $j \geq i + d_\psi$, and for all $k \geq j + d_\psi$.*

Proof. Take any value for i, j, and k such that: (i) $i \geq m_\psi + d_\psi$, (ii) $j \geq i + d_\psi$, (iii) $k \geq j + d_\psi$. We proceed by induction on the structure of the formula ψ.

For the base case, we consider three cases: (i) formulas in $\mathsf{LTL{+}P_{BP}}$, that is such that all its temporal operators refer to the past and are bounded; (ii) formulas of type $\mathsf{G}\alpha$, where $\alpha \in \mathsf{LTL{+}P_{BP}}$; (iii) formulas of type $\alpha \,\mathsf{R}\, \beta$, where $\alpha, \beta \in \mathsf{LTL{+}P_{BP}}$;

We consider the case of a formula $\alpha \in \mathsf{LTL{+}P_{BP}}$, and suppose that ${}^{i,i}\sigma^{j} \models \alpha$. By definition of ${}^{i,i}\sigma^{j}$ and ${}^{i,k}\sigma^{j}$, it always holds that ${}^{i,i}\sigma^{j}_0 = {}^{i,k}\sigma^{j}_0$. Since $\alpha \in \mathsf{LTL{+}P_{BP}}$ refers only to the current state or to the past, it follows that ${}^{i,i}\sigma^{j} \models \alpha$ if and only if ${}^{i,k}\sigma^{j} \models \alpha$.

Consider now the case for $\alpha \,\mathsf{R}\, \beta$, where $\alpha, \beta \in \mathsf{LTL{+}P_{BP}}$. Since $m_{\alpha \mathsf{R} \beta} = 0$ (*i.e.*, the are no *next* operators in this formula), we can apply Lemma 3, having that ${}^{i,i}\sigma^{j} \models \alpha \mathsf{R} \beta$ if and only if ${}^{i,k}\sigma^{j} \models \alpha \mathsf{R} \beta$. Since $\mathsf{G}\alpha = \bot \,\mathsf{R}\, \alpha$, this proves also the case for the *globally* operator.

For the inductive step, since by hypothesis ψ belongs to the normal form of $\mathsf{LTL_{EBR}}$, it suffices to consider only the case for the *next operator*, *conjunctions* and *disjunctions*.

Consider first the case for the *next* operator, and suppose that ${}^{i,i}\sigma^j \models \mathsf{X}\psi'$. For any indices k, i and j such that $i \geq m_{\mathsf{X}\psi'} + d_{\mathsf{X}\psi'}$, $j \geq i + d_{\mathsf{X}\psi'}$ and $k \geq j + d_{\mathsf{X}\psi'}$, we want to prove that ${}^{i,k}\sigma^j \models \mathsf{X}\psi'$. By definition of the next operator, it holds that ${}^{i,i}\sigma^j, 1 \models \psi'$. Now, let τ be the state sequence obtained from ${}^{i,i}\sigma^j$ by discarding its initial state, that is $\tau \coloneqq {}^{i,i}\sigma^j_{[1,\infty)}$. Obviously, $\tau \models \psi'$. We observe that τ is equal to the state sequence ${}^{i-1,i-1}\sigma^{j-1}$. Since the maximum number $m_{\psi'}$ of nested next operators in ψ' is $m_{\mathsf{X}\psi'} - 1$ (while $\alpha_{\psi'}$ remains the same), we can apply the inductive hypothesis on ψ', having that ${}^{i-1,k-1}\sigma^{j-1} \models \psi'$. By definition of τ, it follows that ${}^{i,k}\sigma^j \models \mathsf{X}\psi'$.

We consider now the case for conjunctions, and suppose that ${}^{i,i}\sigma^j \models \psi_1 \wedge \psi_2$, for generic indices k, i and j such that $i \geq m_{\psi_1 \wedge \psi_2} + d_{\psi_1 \wedge \psi_2}$, $j \geq i + d_{\psi_1 \wedge \psi_2}$, and $k \geq j + d_{\psi_1 \wedge \psi_2}$. It holds that ${}^{i,i}\sigma^j \models \psi_1$ and ${}^{i,i}\sigma^j \models \psi_2$. Moreover, $m_{\psi_1} \leq m_{\psi_1 \wedge \psi_2}$ and $m_{\psi_2} \leq m_{\psi_1 \wedge \psi_2}$. Similarly, $d_{\psi_1} \leq d_{\psi_1 \wedge \psi_2}$ and $d_{\psi_2} \leq d_{\psi_1 \wedge \psi_2}$. This means that we can apply the inductive hypothesis both on ψ_1 and ψ_2 on the *current* indices k, i and j. By inductive hypothesis, we have that ${}^{i,k}\sigma^j \models \psi_1$ and ${}^{i,k}\sigma^j \models \psi_2$. It follows that ${}^{i,k}\sigma^j \models \psi_1 \wedge \psi_2$. The case for $\psi_1 \vee \psi_2$ is specular. □

Thanks to Lemma 4, it is simple to prove the undefinability of $\mathsf{G}(p_1 \vee \mathsf{G}(p_2))$ in $\mathsf{LTL_{EBR}}$, that proves that $\mathsf{LTL_{EBR}}$ is *strictly less expressive* than Safety-LTL.

Theorem 23. $[\![\mathsf{LTL_{EBR}}]\!] \subsetneq [\![\text{Safety-LTL}]\!]$.

Proof. Consider the formula $\varphi_{\mathsf{G}} \coloneqq \mathsf{G}(p_1 \vee \mathsf{G}(p_2))$. We prove that there does *not* exists a formula $\psi \in \mathsf{LTL_{EBR}}$ such that $\mathcal{L}(\psi) = \mathcal{L}(\varphi_{\mathsf{G}})$. We proceed by contradiction. Suppose that there exists a formula $\psi \in \mathsf{LTL_{EBR}}$ such that $\mathcal{L}(\psi) = \mathcal{L}(\varphi_{\mathsf{G}})$. By Lemma 1, there exists a formula $\psi' \in \text{Normal-}\mathsf{LTL_{EBR}}$ such that $\mathcal{L}(\psi) = \mathcal{L}(\psi')$. Let $m_{\psi'}$ be the maximum number of *nested next* operators in ψ', and let $d_{\psi'}$ be the maximum temporal depth between all the $\mathsf{LTL{+}P_{BP}}$-subformulas in ψ'. Let k, i and j be three indices such that: (i) $i \geq m_{\psi'} + d_{\psi'}$; (ii) $j \geq i + d_{\psi'}$; (iii) and $k \geq j + d_{\psi'}$. Consider the two state sequences ${}^{i,i}\sigma^j$ and ${}^{i,k}\sigma^j$. By Lemma 4, ${}^{i,i}\sigma^j \in \mathcal{L}(\psi')$ if and only if ${}^{i,k}\sigma^j \in \mathcal{L}(\psi')$, that is ${}^{i,i}\sigma^j \in \mathcal{L}(\varphi_{\mathsf{G}})$ if and only if ${}^{i,k}\sigma^j \in \mathcal{L}(\varphi_{\mathsf{G}})$.

Since it holds that ${}^{i,i}\sigma^j \in \mathcal{L}(\varphi_{\mathsf{G}})$ but ${}^{i,k}\sigma^j \notin \mathcal{L}(\varphi_{\mathsf{G}})$, this is clearly a contradiction. □

Corollary 6. $[\![\mathsf{LTL_{EBR}}]\!] \subsetneq [\![\mathsf{LTL_{EBR}{+}P}]\!]$.

3.3 The GR-EBR logic

The objective of this section is to show an extension of $\mathsf{LTL_{EBR}}$+P that goes beyond the safety fragment. We call the resulting logic *Generalized Reactivity(1)* $\mathsf{LTL_{EBR}}$+P (GR-EBR, for short). We first give the syntax of GR-EBR, and show that is obtained from $\mathsf{LTL_{EBR}}$+P by adding two key ingredients: (i) assumptions and guarantees, (ii) fairness contraints. We then show a meaningful example that can be formalized with the language of GR-EBR. We conclude this section discussing the expressive power of this fragment.

3.3.1 Definition

The logic of *Generalized Reactivity(1)* $\mathsf{LTL_{EBR}}$+P (GR-EBR, for short) is obtained starting from $\mathsf{LTL_{EBR}}$+P (recall Definition 19) by adding (i) assumptions and guarantees, that correspond to the antecedent and the consequent of a logical implication between $\mathsf{LTL_{EBR}}$+P formulas; (ii) and fairness contraints, in the form of conjunctions of recurrence formulas (recall Section 2.4.5), that is of the form $\bigwedge_i \mathsf{GF}\alpha_i$ with $\alpha_i \in \mathsf{LTL{+}P_P}$. The syntax of GR-EBR is the following.

Definition 23 (The logic GR-EBR)**.** *The* GR-EBR *logic comprises all and only those formulas that can be written in the following form:*

$$(\psi^1_{ebr} \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\psi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

for some $m, n \in \mathbb{N}$, $\psi^1_{ebr}, \psi^2_{ebr} \in \mathsf{LTL_{EBR}{+}P}$ *and* $\alpha_i, \beta_j \in \mathsf{LTL{+}P_P}$, *for each* $i, j \in \mathbb{N}$.

3.3.2 Example

We now give an example of specifications that can be expressed in GR-EBR. Suppose that we want to design an arbiter that, given a request from client i (for any $i \in \{1, \ldots, n\}$) in the environment,

assigns the grant to the corresponding client, in such a way to guarantee the following properties: (1) *bounded response*: the grant is assigned at most k time units (for some $k > n$) after the request is issued; (2) *mutual exclusion*: the arbiter can assign a grant at most to one client at a time. The previous requirements form the guarantees for the arbiter. The assumptions for the environment are as follows: (1) initially, there are no requests; (2) if a request is issued at time i, then it cannot be issued until time $i + k$; (3) there are infinitely many requests from each client.

In order to write a specification of the arbiter, we can model the requests for the n clients with the variables $r_1, \ldots, r_n$. Similarly, the grant corresponding to the request r_i can be modeled with the variable g_i, for each $i \in \{1, \ldots, n\}$. The assumption for the environment corresponds to the $\mathsf{LTL}_{\mathsf{EBR}}$ formula ϕ_e defined as follows:

$$\bigwedge_{i=1}^{n} \neg r_i \ \wedge \ \bigwedge_{i=1}^{n} \mathsf{G}(r_i \rightarrow \mathsf{G}^{[1,k]} \neg r_i) \ \wedge \ \bigwedge_{i=1}^{n} \mathsf{GF} r_i$$

The guarantees for the controller correspond to the $\mathsf{LTL}_{\mathsf{EBR}}$ formula ϕ_c defined as follows:

$$\bigwedge_{i=1}^{n} \mathsf{G}(r_i \rightarrow \mathsf{F}^{[0,k]} g_i) \ \wedge \ \mathsf{G}\Big(\bigwedge_{1 \leq i < j \leq n} \neg(g_i \wedge g_j) \Big)$$

The overall specification is $\phi_e \rightarrow \phi_c$ and syntactically belongs to GR-EBR.

3.3.3 Expressive Power

In this part, we investigate the expressive power of the GR-EBR logic by comparing it with other formalisms.

Comparison with $\mathsf{LTL}_{\mathsf{EBR}}$+P

We start by comparing GR-EBR with $\mathsf{LTL}_{\mathsf{EBR}}$+P. Each $\mathsf{LTL}_{\mathsf{EBR}}$+P formula ϕ is a GR-EBR formula as well. In fact, the formula ϕ is equivalent to $(\top \wedge \top) \rightarrow (\phi \wedge \top)$ which belongs to GR-EBR, and thus:

$$\mathsf{LTL}_{\mathsf{EBR}}\text{+P} \subseteq \text{GR-EBR}$$
$$[\![\mathsf{LTL}_{\mathsf{EBR}}\text{+P}]\!] \subseteq [\![\text{GR-EBR}]\!]$$

From Theorem 22, it follows that any safety language definable in LTL is definable in GR-EBR as well. In addition, GR-EBR is *strictly* more expressive than $\mathsf{LTL_{EBR}}$+P, since the former can express also non-safety properties, like $\mathsf{G}(p) \rightarrow \mathsf{G}(q)$, and thus:

$$\mathsf{LTL_{EBR}}+\mathsf{P} \subsetneq \mathsf{GR\text{-}EBR}$$
$$[\![\mathsf{LTL_{EBR}}+\mathsf{P}]\!] \subsetneq [\![\mathsf{GR\text{-}EBR}]\!]$$

Comparison with the Temporal Hierarchy

Recall from Section 2.4.5 the temporal hierarchy defined by Manna and Pnueli in [140]. The Reactivity class is defined as the set of all and only those languages definable by formulas of type $\bigwedge_i(\mathsf{GF}\alpha_i \rightarrow \mathsf{GF}\beta_i)$ where each α_i and each β_i is a pure-past LTL formula. It is known that LTL is expressively equivalent to the Reactivity class (see Sections 2.4.3 and 2.4.5). Moreover, if we fix the number of conjuncts of the formula above to be N, than the resulting class (called Reactivity(N)) strictly contains Reactivity(N-1) and is strictly contained in Reactivity(N+1). With respect to the temporal hierarchy, we can prove the following result.

Theorem 24. *It holds that:*

$$[\![\mathsf{R}(1)]\!] \subseteq [\![\mathsf{GR\text{-}EBR}]\!]$$

Proof. Let $\phi \in \mathsf{R}(1)$. By definition of $\mathsf{R}(1)$ (recall Definition 17), it holds that $\phi \equiv \mathsf{GF}\alpha \vee \mathsf{FG}\beta$, for some $\alpha, \beta \in \mathsf{LTL}+\mathsf{P_P}$. Since it holds that

$$\phi \equiv \mathsf{GF}\neg\beta \rightarrow \mathsf{GF}\alpha \equiv (\top \wedge \mathsf{GF}\neg\beta) \rightarrow (\top \wedge \mathsf{GF}\alpha) \in \mathsf{GR\text{-}EBR}$$

we have that $\mathcal{L}(\phi) \in [\![\mathsf{GR\text{-}EBR}]\!]$ and thus $[\![\mathsf{R}(1)]\!] \subseteq [\![\mathsf{GR\text{-}EBR}]\!]$. □

Comparison with GR(1)

Recall from Definition 18 that the logic of GR(1) (*Generalized Reactivity(1)*) is defined as the set of formulas of type:

$$(\bigwedge_{i=1}^{m} \mathsf{GF}\beta_i) \rightarrow (\bigwedge_{i=1}^{n} \mathsf{GF}\alpha_i)$$

for any $m, n \in \mathbb{N}$, where $\alpha_i, \beta_i \in \mathsf{LTL}+\mathsf{P_P}$ for each $i \in \{1, \ldots, n\}$. Despite being a fragment with en efficient realizability problem, GR(1)

presents some restrictions that limit its use as a specification language:

- safety assumptions/guarantees are either Boolean formulas or formulas of the form $\mathsf{G}\alpha$, where the only temporal operator admitted in α is the *next* operator X;
- assumptions are syntactically constrained to be formulas *controlled* by Environment, in the sense that the variables inside the *next* operators of the safety part of the assumptions must be *uncontrollable*.

In GR-EBR we relax that syntactical restrictions of GR(1): for example, the safety assumptions and guarantees can be any arbitrary $\mathsf{LTL}_{\mathsf{EBR}}$+P formula, like, for instance, $\mathsf{G}(r \rightarrow \mathsf{F}^{[0,10]}g)$. For this reason, GR-EBR can be considered an extension not only of $\mathsf{LTL}_{\mathsf{EBR}}$+P, but also of GR(1).

On the semantics side, we can prove the following result.

Theorem 25. *It holds that:*

$$[\![\mathsf{GR\text{-}EBR}]\!] \subseteq [\![\mathsf{GR(1)}]\!]$$

Proof. Consider a formula $\phi \in$ GR-EBR. By Definition 23, ϕ is of the following form:

$$(\psi^1_{ebr} \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\psi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

for some $m, n \in \mathbb{N}$, $\psi^1_{ebr}, \psi^2_{ebr} \in \mathsf{LTL}_{\mathsf{EBR}}$+P and $\alpha_i, \beta_j \in \mathsf{LTL}$+$\mathsf{P}_\mathsf{P}$, for each $i, j \in \mathbb{N}$. Since $\mathsf{LTL}_{\mathsf{EBR}}$+P is expressively complete with respect to the safety fragment of LTL+P (Theorem 22) and since $\mathsf{G}\alpha$ (with $\alpha \in \mathsf{LTL}$+P_P) is a normal form for the safety fragment of LTL+P (Theorem 16), it follows that there exist two pure past formulas $\gamma_1, \gamma_2 \in \mathsf{LTL}$+$\mathsf{P}_\mathsf{P}$ such that: (i) $\mathsf{G}\gamma_1 \equiv \psi^1_{ebr}$, (ii) and $\mathsf{G}\gamma_2 \equiv \psi^2_{ebr}$. Therefore, ϕ is equivalent to:

$$(\mathsf{G}\gamma_1 \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\mathsf{G}\gamma_2 \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

Now, since $\mathsf{G}\gamma_1 \equiv \mathsf{GFH}\gamma_1$ (and the same holds for γ_2 as well), we obtain the following formula:

$$(\mathsf{GF}(\mathsf{H}\gamma_1) \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\mathsf{GF}(\mathsf{H}\gamma_2) \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

Since the argument of all the formulas of type GF is a pure past formula (in both the assumptions and the guarantees), this formula belongs to GR(1). Therefore, $[\![\mathsf{GR\text{-}EBR}]\!] \subseteq \mathsf{GR}(1)$. □

From the previous two theorems, it follows the truth of this theorem.

Theorem 26. *It holds that:*

$$[\![\mathsf{R}(1)]\!] \subseteq [\![\mathsf{GR\text{-}EBR}]\!] \subseteq [\![\mathsf{GR}(1)]\!]$$

Comparison with (co-)Safety and Liveness classes

Since the R(1) class contains the co-safety, the safety and the liveness fragments of LTL (recall Fig. 2.1), as a corollary of Theorem 26, we obtain the result that $[\![\mathsf{GR\text{-}EBR}]\!]$ contains, $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$, $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$ and $[\![\mathsf{LTL}]\!] \cap \mathsf{LIVENESS}$.

Corollary 7. *It holds that:*

1. $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY} \subseteq [\![\mathsf{GR\text{-}EBR}]\!]$
2. $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY} \subseteq [\![\mathsf{GR\text{-}EBR}]\!]$
3. $[\![\mathsf{LTL}]\!] \cap \mathsf{LIVENESS} \subseteq [\![\mathsf{GR\text{-}EBR}]\!]$

Fig. 3.2 shows the expressive power of $\mathsf{LTL_{EBR}}$+P, $\mathsf{LTL_{EBR}}$, and GR-EBR compared to the other fragments already pictured in Fig. 2.2.

3.4 Conclusions

We introduced the $\mathsf{LTL_{EBR}}$+P logic, a safety fragment of LTL+P. The syntax of $\mathsf{LTL_{EBR}}$+P make it difficult to exactly characterize its expressive power. We studied the expressive power of $\mathsf{LTL_{EBR}}$+P and of its pure future fragment, $\mathsf{LTL_{EBR}}$, and compare it with other safety fragments of LTL. It turned out that $\mathsf{LTL_{EBR}}$+P is expressively complete with respect to the safety fragment of LTL, and, consequently, it is expressively equivalent to Safety-LTL. We found out that past modalities are crucial for the expressive power of $\mathsf{LTL_{EBR}}$+P. In fact, $\mathsf{LTL_{EBR}}$ is strictly less expressive than full $\mathsf{LTL_{EBR}}$+P. This was somehow surprising, since it proves that, despite not being fundamental for the expressiveness of full LTL, past modalities are crucial for fragments of LTL, like, for instance, $\mathsf{LTL_{EBR}}$+P.

We also introduced the logic of GR-EBR, which can be seen as an extension of $\mathsf{LTL}_{\mathsf{EBR}}$+P able to express properties beyond the safety fragment. In particular, GR-EBR extends $\mathsf{LTL}_{\mathsf{EBR}}$ by allowing assumptions and guarantees, and fairness conditions. We investigated the expressive power of GR-EBR, comparing it with the classical temporal hierarchy.

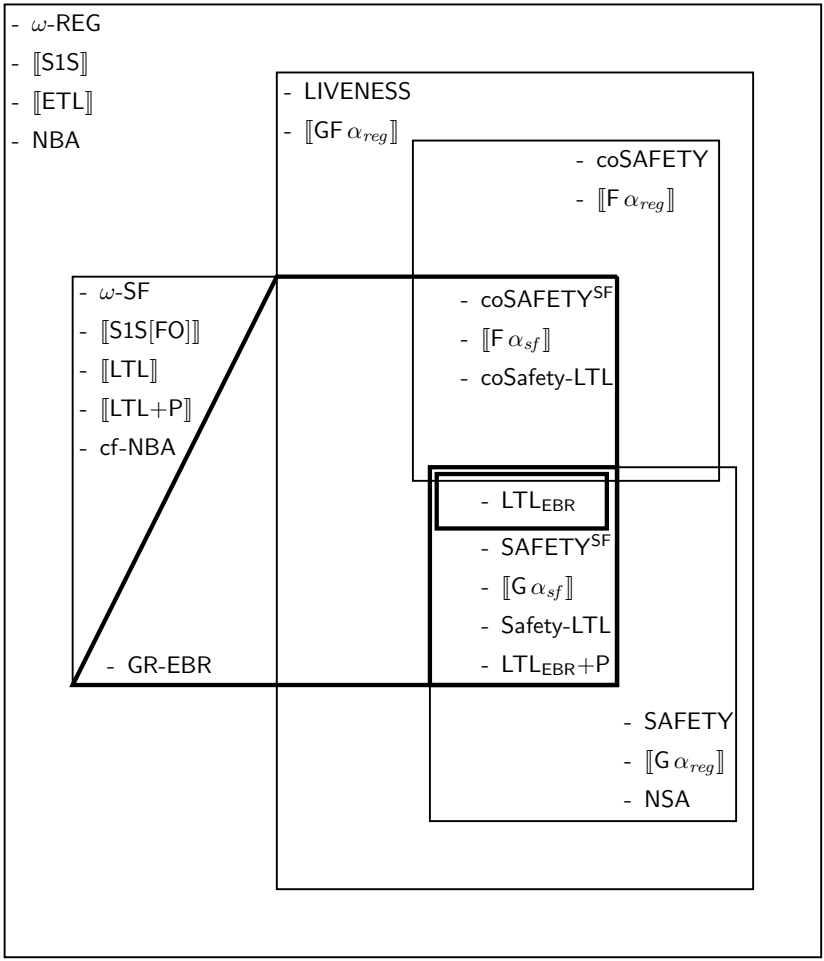


Figure 3.2: Comparison of the expressive power of $\mathsf{LTL}_{\mathsf{EBR}}$+P, $\mathsf{LTL}_{\mathsf{EBR}}$ and GR-EBR. It is an extension of Fig. 2.2. For ease of exposition, we highlighted the rectangles corresponding to $\mathsf{LTL}_{\mathsf{EBR}}$+P, $\mathsf{LTL}_{\mathsf{EBR}}$ and GR-EBR with thick borders.

CHAPTER

4

A FIRST-ORDER LOGIC CHARACTERISATION OF SAFETY AND CO-SAFETY LANGUAGES

In this chapter, we provide a novel syntactical fragment of S1S[FO] (the first-order fragment of S1S) that captures exactly the semantically safety fragment of S1S[FO], and therefore it is also *expressively complete* with respect to the safety fragment of LTL (*i.e.*, $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$) and to the Safety-LTL logic (namely LTL devoid of existential temporal operators). In this fragment, called Safety-FO, only a particularly constrained kind of existential and universal quantifications are allowed. We prove these results by showing first the same correspondences for the dual case, that is for the co-safety fragment of LTL, the coSafety-LTL logic (namely LTL devoid of universal temporal operators) and the coSafety-FO fragment.

The main result of this chapter is the proof of the correspondence between coSafety-FO and coSafety-LTL, which extends naturally to their duals (*i.e.*, Safety-FO and Safety-LTL). This can also be considered as a version of Kamp's theorem [120] specialized for safety and co-safety properties, helping to create a clearer picture of the correspondence between (fragments of) temporal and first-order logics.

Then, we use this result for deriving the *expressive completeness* of the *syntactical* fragment coSafety-FO (resp. Safety-FO) with respect to the *semantically* co-safety (resp. safety) fragment of S1S[FO]. This also proves the correspondence between the semantically co-safety fragment of LTL (*i.e.*, $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$) and the coSafety-FO logic, thus establishing also the equivalence between the first one and coSafety-LTL. Again, this result extends to the dual case. This provides an alternative proof of the fact that Safety-LTL captures exactly the set of LTL-definable safety languages [40], which can be regarded as another contribution of this chapter. The interest of this alternative proof is twofold: on the one hand, the original proof by Chang *et al.* [40] is only outlined and it relies on two non-trivial translations scattered across different sources [202, 174]; on the other hand, such an equivalence result seems not to be very much known, as the problem was presented as open as lately as 2021 [201, 71].[1] Thus, a compact and self-contained proof of the result seems to be a useful contribution for the community. Finally, as a by-product of this proof, we provide some results that assess the expressive power of the *weak next* operator of Safety-LTL when interpreted over finite *vs.* infinite traces.

4.1 Safety-FO and coSafety-FO

In this section we introduce the core contribution of this chapter, *i.e.*, two fragments of S1S[FO] that precisely capture Safety-LTL and coSafety-LTL, respectively, and we prove this relationship.

Definition 24 (Safety-FO)**.** *The logic* Safety-FO *is generated by the*

[1] As a matter of fact, we discovered about Chang *et al.* [40] after setting up the proof shown in this chapter.

following grammar:

$$\begin{aligned} \mathit{atomic} &:= x < y \mid x = y \mid x \neq y \mid P(x) \mid \neg P(x) \\ \phi &:= \mathit{atomic} \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \\ &\quad \exists y(x < y < z \wedge \phi_1) \mid \\ &\quad \forall y(x < y \rightarrow \phi_1) \end{aligned}$$

where x, y, and z are first-order variables, P is a unary predicate, and ϕ_1 and ϕ_2 are Safety-FO *formulas.*

Definition 25 (coSafety-FO)**.** *The logic* coSafety-FO *is generated by the following grammar:*

$$\begin{aligned} \mathit{atomic} &:= x < y \mid x = y \mid x \neq y \mid P(x) \mid \neg P(x) \\ \phi &:= \mathit{atomic} \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \\ &\quad \exists y(x < y \wedge \phi_1) \mid \\ &\quad \forall y(x < y < z \rightarrow \phi_1) \end{aligned}$$

where x, y, and z are first-order variables, P is a unary predicate, and ϕ_1 and ϕ_2 are coSafety-FO *formulas.*

4.1.1 Notations

For sake of clarity, given a finite alphabet Σ, we denote with the same symbol both a the infinite (resp. finite) state sequence $\sigma \in (2^\Sigma)^\omega$ (resp. $\sigma \cup (2^\Sigma)^*$) and the corresponding model-theoretic structure $\sigma = (D, 0, +1, <, \{Q_a\}_{a\in\Sigma})$, with $D = \mathbb{N}$ (resp. D such that $|D| < \omega$).

Recall from Section 2.1 that, given an S1S[FO] formula $\phi(x_1, \ldots, x_n)$ with n free variables, a structure $\sigma = (\mathbb{N}, 0, +1, <, \{Q_a\}_{a\in\Sigma})$, and n values $v_1, \ldots, v_n$ in the domain $\mathbb{N}$, we refer with $\sigma, v_1, \ldots, v_n \models \phi(x_1, \ldots, x_n)$ to the fact that σ is a *model* of $\phi(x_1, \ldots, x_n)$ when x_i is interpreted with the value v_i, for each $i \in \{1, \ldots, n\}$. Under finite word semantics, we write $\models^{<\omega}$ instead of $\models$.

In Section 2.1, we defined the notion of *language* only for S1S sentences, that is formulas with no free variables. In the following, it will be handy to consider instead formulas with *exactly one* free variable, representing the instant in time when it is interpreted, and to define the notion of *language* accordingly. Therefore, given a formula $\phi(x)$ with exactly one free variable, the *language* of $\phi(x)$,

denoted with $\mathcal{L}(\phi(x))$ (resp. $\mathcal{L}^{<\omega}(\phi(x))$), is the set of infinite (resp. finite) structures σ such that $\sigma, 0 \models \phi(x)$ (resp. $\sigma, 0 \models^{<\omega} \phi(x)$). It follows that the sets $[\![\mathsf{Safety\text{-}FO}]\!]$ and $[\![\mathsf{coSafety\text{-}FO}]\!]$ are defined as follows:

$$[\![\mathsf{Safety\text{-}FO}]\!] = \{\mathcal{L}(\phi(x)) \mid \phi(x) \in \mathsf{Safety\text{-}FO}\}$$
$$[\![\mathsf{coSafety\text{-}FO}]\!] = \{\mathcal{L}(\phi(x)) \mid \phi(x) \in \mathsf{coSafety\text{-}FO}\}$$

The same applies to the sets $[\![\mathsf{Safety\text{-}FO}]\!]^{<\omega}$ and $[\![\mathsf{coSafety\text{-}FO}]\!]^{<\omega}$.

4.1.2 Discussion on the two fragments

We need to make a few observations on the syntax of the two fragments. First of all, note how any formula of Safety-FO is the negation of a formula of coSafety-FO and *vice versa*. Then, note that the two fragments are defined in *negated normal form*, *i.e.*, negation only appears on atomic formulas. The particular kind of existential and universal quantifications allowed are the culprit of these fragments. In particular, each Safety-FO sentence (that is, a formula without free variable) restricts any existentially quantified variable to be bounded between two already quantified variables. The same applies to universal quantifications in coSafety-FO. Moreover Safety-FO and coSafety-FO formulas are *future formulas*, *i.e.*, the quantifiers can only range over values *greater* than already quantified variables. These two features are essential to precisely capture Safety-LTL and coSafety-LTL. Finally, note that the comparisons in the guards of the quantifiers are strict, but non-strict comparisons can be used as well. In particular, $\exists y(x \leq y \wedge \phi)$ can be rewritten as $\phi[y/x] \vee \exists y(x < y \wedge \phi)$, where $\phi[y/x]$ is the formula obtained by replacing all occurrences of y with x. Similarly, $\forall z(x \leq z \leq y \rightarrow \phi)$ can be rewritten as $\phi[z/x] \wedge \phi[z/y] \wedge \forall z(x < z < y \rightarrow \phi)$.

4.1.3 Lemmas

To prove the relationship between these fragments and Safety-LTL and coSafety-LTL, we focus now on coSafety-FO. By duality, all the results transfer to Safety-FO. Let us start by defining the coSafety-LTL$(-\widetilde{\mathsf{X}})$ logic as coSafety-LTL devoid of the *weak next* operator. We observe that, since the *weak next* operator, over infinite words, coincides with the *next* operator, coSafety-LTL and coSafety-LTL$(-\widetilde{\mathsf{X}})$ have the same expressive power.

Observation 1. $[\![\mathsf{coSafety\text{-}LTL}]\!] = [\![\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})]\!]$

When reasoning over finite words, the *weak next* operator plays a crucial role, since it can be used to recognize when we are at the last position of a word. In fact, the formula $\sigma, i \models^{<\omega} \widetilde{\mathsf{X}}\bot$ is true if and only if $i = |\sigma| - 1$, for any $\sigma \in (2^{\Sigma})^*$.

Now, let us note that, when the *weak next* operator is not considered, reasoning over infinite words in some sense reduces to reasoning over finite words. In fact, the following result holds.

Lemma 5. $[\![\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})]\!] = [\![\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$

Proof. We have to prove that, for each formula $\phi \in \mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})$, it holds that:

$$\mathcal{L}(\phi) = \mathcal{L}^{<\omega}(\phi) \cdot (2^{\Sigma})^{\omega}$$

We proceed by induction on the structure of ϕ. For the base case, consider $\phi \equiv p \in \Sigma$. The case for $\phi \equiv \neg p$ is similar. Let $\sigma \in \mathcal{L}(p)$. It holds that $\sigma_0 \models p$ and $\sigma_0 \cdot \sigma' \models p$, for all $\sigma' \models (2^{\Sigma})^{\omega}$, and in particular for $\sigma' = \sigma_{[1,\infty)}$. This is equivalent to say that $\sigma \in \mathcal{L}^{<\omega}(\phi) \cdot (2^{\Sigma})^{\omega}$. For the inductive step:

1. Let $\phi \equiv \phi_1 \wedge \phi_2$. Suppose that $\sigma \in \mathcal{L}(\phi)$. Obviously, $\sigma \models \phi_1$ and $\sigma \models \phi_2$, and therefore $\sigma \in \mathcal{L}(\phi_1)$ and $\sigma \in \mathcal{L}(\phi_2)$. By the inductive hypothesis, $\sigma \in \mathcal{L}^{<\omega}(\phi_1) \cdot (2^{\Sigma})^{\omega}$ and $\sigma \in \mathcal{L}^{<\omega}(\phi_2) \cdot (2^{\Sigma})^{\omega}$. This means that there exists two indices $i, j \in \mathbb{N}$ such that $\sigma_{[0,i]} \models^{<\omega} \phi_1$ and $\sigma_{[0,j]} \models^{<\omega} \phi_2$. Let m be the greatest between i and j. It holds that $\sigma_{[0,m]} \models^{<\omega} \phi_1 \wedge \phi_2$. Therefore $\sigma \in \mathcal{L}^{<\omega}(\phi_1 \wedge \phi_2) \cdot (2^{\Sigma})^{\omega}$.

2. Let $\phi \equiv \phi_1 \vee \phi_2$ and let $\sigma \in \mathcal{L}(\phi)$. We have that $\sigma \models \phi_1$ or $\sigma \models \phi_2$. Without loss of generality, we consider the case that $\sigma \models \phi_1$ (the other case is specular). By the inductive hypothesis, $\sigma \in \mathcal{L}^{<\omega}(\phi_1) \cdot (2^{\Sigma})^{\omega}$. Therefore, it also holds that $\sigma \in \mathcal{L}^{<\omega}(\phi_1 \vee \phi_2) \cdot (2^{\Sigma})^{\omega}$.

3. Let $\phi \equiv \mathsf{X}\phi_1$ and let $\sigma \in \mathcal{L}(\mathsf{X}\phi_1)$. By the semantics of the *next* operator, it holds that $\sigma_{[1,\infty)} \models \phi_1$. By the inductive hypothesis, $\sigma_{[1,\infty)} \in \mathcal{L}^{<\omega}(\phi_1) \cdot (2^{\Sigma})^{\omega}$. This means that there exists an index $i \geq 1$ such that $\sigma_{[1,i]} \models^{<\omega} \phi_1$. Therefore, it also holds that the state sequence $\sigma_{[0,i]} = \sigma_0 \cdot \sigma_{[1,i]}$ satisfies $\mathsf{X}\phi_1$ over finite words, that is, $\sigma_{[0,i]} \models^{<\omega} \mathsf{X}\phi_1$. This means that $\sigma \in \mathcal{L}^{<\omega}(\mathsf{X}\phi_1) \cdot (2^{\Sigma})^{\omega}$.

4. Let $\phi \equiv \phi_1 \mathsf{U} \phi_2$. Let $\sigma \in \mathcal{L}(\phi)$. By the semantics of the *until* operator, it holds that there exists an index $i \in \mathbb{N}$ such that $\sigma_{[i,\infty)} \models \phi_2$ and $\sigma_{[j,\infty)} \models \phi_1$ for all $0 \leq j < i$. By the inductive hypothesis, we have that $\sigma_{[i,\infty)} \in \mathcal{L}^{<\omega}(\phi_2) \cdot (2^\Sigma)^\omega$ and $\sigma_{[j,\infty)} \in \mathcal{L}^{<\omega}(\phi_1) \cdot (2^\Sigma)^\omega$ for all $0 \leq j < i$. This means that there exists an index $i \in \mathbb{N}$ and $i+1$ indices $k_0 \dots k_i \in \mathbb{N}$ such that $\sigma_{[i,k_i]} \models^{<\omega} \phi_2$ and $\sigma_{[j,k_j]} \models^{<\omega} \phi_1$ for all $0 \leq j < i$. Let m be the greatest between $k_0 \dots k_i$. It holds that there exists an index $i \in \mathbb{N}$ such that $\sigma_{[i,m]} \models^{<\omega} \phi_2$ and $\sigma_{[j,m]} \models^{<\omega} \phi_1$ for all $0 \leq j < i$. Therefore, $\sigma \in \mathcal{L}^{<\omega}(\phi_1 \mathsf{U} \phi_2) \cdot (2^\Sigma)^\omega$. □

The following lemma proves that the result above applies to coSafety-FO as well.

Lemma 6. $[\![\mathsf{coSafety\text{-}FO}]\!] = [\![\mathsf{coSafety\text{-}FO}]\!]^{<\omega} \cdot (2^\Sigma)^\omega$

Proof. We have to prove that, for each formula $\psi \in \mathsf{coSafety\text{-}FO}$ with one free variable, it holds that $\mathcal{L}(\psi) = \mathcal{L}^{<\omega}(\psi) \cdot (2^\Sigma)^\omega$. We proceed by induction, but with a more general statement. Let $\phi(x_1, \dots, x_k)$ have k free variables. We prove by induction on ϕ that for any infinite state sequence σ such that $\sigma, n_1, \dots, n_k \models \phi(x_1, \dots, x_k)$, there exists a prefix $\sigma_{[0,i]}$ of σ such that for all $\sigma' \in (2^\Sigma)^*$, $\sigma_{[0,i]}\sigma', n_1, \dots, n_k \models^{<\omega} \phi(x_1, \dots, x_k)$. The base case considers the four kinds of atomic formulas. If $\sigma, n_1, n_2 \models x_1 < x_2$, then $n_1 < n_2$ and we know that $\sigma_{[0,n_2]}\sigma', n_1, n_2 \models^{<\omega} x_1 < x_2$ for all $\sigma' \in (2^\Sigma)^*$. The case of $x_1 = x_2$ is similar. Now, if $\sigma, n_1 \models P(x_1)$, then $p \in \sigma_{n_1}$ and we know that $\sigma_{[0,n_1]}\sigma', n_1 \models^{<\omega} P(x_1)$ for all $\sigma' \in (2^\Sigma)^*$. The case for $\neg P(x)$ is similar. For the inductive step:

1. if $\sigma, n_1, \dots, n_k \models \phi_1(x_1, \dots, x_k) \wedge \phi_2(x_1, \dots, x_k)$, by the induction hypothesis we know that there are two prefixes $\sigma_{[0,i]}$ and $\sigma_{[0,j]}$ such that $\sigma_{[0,i]}\sigma', n_1, \dots, n_k \models^{<\omega} \phi_1(x_1, \dots, x_k)$ and $\sigma_{[0,j]}\sigma'', n_1, \dots, n_k \models^{<\omega} \phi_2(x_1, \dots, x_k)$, for all $\sigma', \sigma'' \in (2^\Sigma)^*$. Then, supposing *w.l.o.g.* that $i \leq j$, we know that $\sigma_{[0,j]}\sigma'', n_1, \dots, n_k \models^{<\omega} \phi_1(x_1, \dots, x_k) \wedge \phi_2(x_1, \dots, x_k)$. The case for $\phi_1(x_1, \dots, x_k) \vee \phi_2(x_1, \dots, x_k)$ is similar.

2. If $\sigma, n_1, \dots, n_k \models \exists x_{k+1}(x_u < x_{k+1} \wedge \phi_1(x_1, \dots, x_{k+1}))$ for some $1 \leq u \leq k$, then there exists an $n_{k+1} > n_u$ such that $\sigma, n_1, \dots, n_{k+1} \models \phi_1(x_1, \dots, x_{k+1})$. Then, by the induction hypothesis, we have that $\sigma_{[0,i]}\sigma', n_1, \dots, n_{k+1} \models^{<\omega} \phi_1(x_1, \dots, x_{k+1})$ for some $i \geq 0$ and all $\sigma' \in (2^\Sigma)^*$. It follows that $\sigma_{[0,i]}\sigma', n_1, \dots, n_k \models^{<\omega} \exists x_{k+1}(x_u < x_{k+1} \wedge \phi_1(x_1, \dots, x_{k+1}))$.

3. if $\sigma, n_1, \ldots, n_k \models \forall x_{k+1}(x_u < x_{k+1} < x_v \rightarrow \phi_1(x_1, \ldots, x_{k+1}))$ for some $1 \leq u, v \leq k$, then for all n_{k+1} with $n_u < n_{k+1} < n_v$ it holds that $\sigma, n_1 \ldots, n_{k+1} \models \phi_1(x_1, \ldots, x_{k+1})$. Then, for the induction hypothesis, for all n_{k+1} with $n_u < n_{k+1} < n_v$ there is a prefix $\sigma_{[0,i_{n_{k+1}}]}$ such that $\sigma_{[0,i_{n_{k+1}}]}\sigma', n_1, \ldots, n_{k+1} \models^{<\omega} \phi_1(x_1, \ldots, x_{k+1})$ for all $\sigma' \in (2^{\Sigma})^*$. Then, if $n_* = \max_{n_u < n_{k+1} < n_v}(i_{n_{k+1}})$, it holds that $\sigma_{[0,n_*]}\sigma', n_1, \ldots, n_k \models^{<\omega} \forall x_{k+1}(x_u < x_{k+1} < x_v \rightarrow \phi_1(x_1, \ldots, x_{k+1}))$.

Now, let $\psi(x)$ be a coSafety-FO formula with exactly one free variable x. Thanks to the above induction we can conclude that each infinite state sequence σ such that $\sigma, 0 \models \phi(x)$ is of the form $\sigma_{[0,i]} \cdot \sigma'$, where $\sigma_{[0,i]} \models^{<\omega} \phi(x)$, and this implies that $\mathcal{L}(\psi) = \mathcal{L}^{<\omega}(\psi) \cdot (2^{\Sigma})^{\omega}$. □

It is worth to note that Lemmas 5 and 6 show that coSafety-LTL($-\widetilde{\mathsf{X}}$) and coSafety-FO are *insensitive to infiniteness* as defined by De Giacomo *et al.* [104].

Now, we can focus on the relationship between coSafety-LTL($-\widetilde{\mathsf{X}}$) and coSafety-FO on finite words. If we can prove that $[\![\text{coSafety-LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega} = [\![\text{coSafety-FO}]\!]^{<\omega}$, we are done. At first, we show how to encode coSafety-LTL($-\widetilde{\mathsf{X}}$) formulas into coSafety-FO.

Lemma 7. $[\![\text{coSafety-LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega} \subseteq [\![\text{coSafety-FO}]\!]^{<\omega}$

Proof. Let $\mathcal{L} \in [\![\text{coSafety-LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega}$, and let $\phi \in \text{coSafety-LTL}(-\widetilde{\mathsf{X}})$ such that $\mathcal{L} = \mathcal{L}^{<\omega}(\phi)$. By following the semantics of the operators in ϕ, we can obtain an equivalent coSafety-FO formula $\phi_{\mathsf{S1S[FO]}}$. We inductively define the formula $FO(\phi, x)$, where x is a variable, as follows:

- $FO(p, x) = P(x)$, for each $p \in \Sigma$
- $FO(\neg p, x) = \neg P(x)$, for each $p \in \Sigma$
- $FO(\phi_1 \wedge \phi_2, x) = FO(\phi_1, x) \wedge FO(\phi_2, x)$
- $FO(\phi_1 \vee \phi_2, x) = FO(\phi_1, x) \vee FO(\phi_2, x)$
- $FO(\mathsf{X}\phi_1, x) = \exists y(x < y \wedge y = x + 1 \wedge FO(\phi_1, y))$ where $y = x+1$ can be expressed with the coSafety-FO formula $\forall z(x < z < y \rightarrow \bot)$.

- $FO(\phi_1 \mathsf{U} \phi_2, x) = \exists y(x \leq y \wedge FO(\phi_2, y) \wedge \forall z(x \leq z < y \rightarrow FO(\phi_1, z)))$

For each $\phi \in \mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})$, the formula $FO(\phi, x)$ has exactly one free variable x. It is easy to see that for all finite state sequences $\sigma \in (2^{\Sigma})^*$, it holds that $\sigma \models^{<\omega} \phi$ if and only if $\sigma, 0 \models^{<\omega} FO(\phi, x)$, and $FO(\phi, x) \in \mathsf{coSafety\text{-}FO}$. Therefore, $\mathcal{L} \in [\![\mathsf{coSafety\text{-}FO}]\!]^{<\omega}$. □

4.1.4 Main Theorem

It is time to show the opposite direction of Lemma 7, *i.e.*, that any coSafety-FO formula can be translated into a $\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})$ formula which is equivalent over finite words. To prove this fact we adapt a proof of Kamp's theorem by Rabinovich [162]. Our case is relatively simpler because we do not have to deal with negations, but we need to explicitly account for the universal quantification. The proof goes by introducing a *normal form*, and showing that (i) any coSafety-FO formula can be translated into such normal form and (ii) any formula in normal form can be straightforwardly translated into a $\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})$ formula. We start by introducing such a normal form.

Definition 26 ($\overrightarrow{\exists}\forall^g$-formulas)**.** *An $\overrightarrow{\exists}\forall^g$-formula $\phi(z_0, \ldots, z_m)$ with m free variables is a formula of this form:*

$$
\begin{aligned}
\phi(z_0,\ldots,z_m) \coloneqq \exists x_0 \ldots \exists x_n \big(& \\
& x_0 < x_1 < \cdots < x_n && \textit{ordering constraints} \\
\wedge\, & z_0 = x_0 \wedge \bigwedge_{k=1}^{m} (z_k = x_{i_k}) && \textit{binding constraints} \\
\wedge\, & \bigwedge_{j=0}^{n} \alpha_j(x_j) && \textit{punctual constraints} \\
\wedge\, & \bigwedge_{j=1}^{n} \forall y(x_{j-1} < y < x_j \rightarrow \beta_j(y))\big) && \textit{interval constraints}
\end{aligned}
$$

where $i_k \in \{0, \ldots, n\}$ for each $0 \leq k \leq m$, and α_j and β_j, for each $1 \leq j \leq n$, are quantifier-free formulas with exactly one free variable.

Some explanations are due. Each $\overrightarrow{\exists}\forall^g$-formula states a number of requirements for its free variables and for its quantified variables.

Through the binding constraints, the free variables are identified with a subset of the quantified variables in order to uniformly state the punctual and interval constraints, and the ordering constraints which sort all the variable in a total order. Note that there is no relationship between n and m: there might be more quantified variables than free variables, or less. Note as well that the binding constraint $z_0 = x_0$ is always present, *i.e.*, at least one free variable has to be the minimal element of the ordering. This ensures that $\overrightarrow{\exists}\forall^g$-formulas are always *future* formulas.

We say that a formula of coSafety-FO is in *normal form* if and only if it is a disjunction of $\overrightarrow{\exists}\forall^g$-formulas. To see how formulas in normal form make sense, let us immediately show how to translate them into coSafety-LTL$(-\widetilde{\mathsf{X}})$ formulas.

Lemma 8. *For any formula $\phi(z) \in$ coSafety-FO in normal form, with exactly one free variable, there exists formula $\psi \in$ coSafety-LTL$(-\widetilde{\mathsf{X}})$ such that $\mathcal{L}^{<\omega}(\phi(z)) = \mathcal{L}^{<\omega}(\psi)$.*

Proof. We show how any $\overrightarrow{\exists}\forall^g$-formula is equivalent to an coSafety-LTL$(-\widetilde{\mathsf{X}})$-formula, over finite words. Since each formula in normal form is a disjunction of $\overrightarrow{\exists}\forall^g$-formulas, and since coSafety-LTL$(-\widetilde{\mathsf{X}})$ is closed under disjunction, this implies the proposition. Let $\phi(z)$ be a $\overrightarrow{\exists}\forall^g$-formula with a single free variable. Having only one free variable, $\phi(z)$ is of the form:

$$\begin{aligned}\exists x_0 \dots \exists x_n \big(& \\ & x_0 < \dots < x_n \\ & \wedge z = x_0 \\ & \wedge \bigwedge_{j=0}^{n} \alpha_j(x_j) \wedge \bigwedge_{j=1}^{n} \forall y (x_{j-1} < y < x_j \rightarrow \beta_j(y))\big)\end{aligned}$$

Now, let A_i be the temporal formulas corresponding to α_i and B_i be the ones corresponding to β_i. Recall that α_i and β_i are quantifier free with only one free variable, hence this correspondence is trivial. Since z is the first time point of the ordering mandated by the formula, we only need future temporal operators to encode ϕ into a coSafety-LTL$(-\widetilde{\mathsf{X}})$ formula ψ defined as follows:

$$\psi \equiv A_0 \wedge \mathsf{X}(B_0 \,\mathsf{U}\, (A_1 \wedge \mathsf{X}(B_1 \,\mathsf{U}\, A_2 \wedge \dots \mathsf{X}(B_{n-1} \,\mathsf{U}\, A_n) \dots))))$$

It can be seen that $\sigma, k \models^{<\omega} \psi$ if and only if $\sigma, k \models^{<\omega} \phi(z)$, for each $\sigma \in (2^{\Sigma})^+$ and each $k \geq 0$. Thus, $\mathcal{L}^{<\omega}(\phi(z)) = \mathcal{L}^{<\omega}(\psi)$. □

Two differences between our $\overrightarrow{\exists}\forall^g$-formulas and those used by Rabinovich [162] are crucial: first, we do not have unbounded universal requirements, but all interval constraints use bounded quantifications, hence we do not need the *globally* operator to encode them (which does not belong to the syntax of coSafety-LTL$(-\widetilde{\mathsf{X}})$); second, our $\overrightarrow{\exists}\forall^g$-formulas are *future* formulas, hence we only need future operators to encode them.

We now show that any coSafety-FO formula can be translated into normal form, that is, into a *disjunction* of $\overrightarrow{\exists}\forall^g$-formulas.

Lemma 9. *Any* coSafety-FO *formula is equivalent to a disjunction of* $\overrightarrow{\exists}\forall^g$*-formulas.*

Proof. Let ϕ be a coSafety-FO formula. We proceed by structural induction on ϕ. For the base case, for each atomic formula $\phi(z_0, z_1)$ we provide an equivalent $\overrightarrow{\exists}\forall^g$-formula $\psi(z_0, z_1)$:

1. if $\phi \equiv z_0 < z_1$ then $\psi \equiv \exists x_0 \exists x_1 (z_0 = x_0 \wedge z_1 = x_1 \wedge x_0 < x_1)$;
2. if $\phi \equiv z_0 = z_1$, then $\psi \equiv \exists x_0 (z_0 = x_0 \wedge z_1 = x_0)$.
3. if $\phi \equiv z_0 \neq z_1$, we can note that $\phi \equiv z_0 < z_1 \vee z_1 < z_0$ and then apply Item 1;
4. If $\phi \equiv P(z_0)$ then we define $\psi \coloneqq \exists x_0 (z_0 = x_0 \wedge P(x_0))$. Similarly if $\phi \equiv \neg P(z_0)$.

For the inductive step:

1. The case of a disjunction is trivial.
2. If $\phi(z_0, \ldots, z_k)$ is a conjunction, by the inductive hypothesis each conjunct is equivalent to a disjunction of $\overrightarrow{\exists}\forall^g$-formulas. By distributing the conjunction over the disjunction we can reduce ourselves to the case of a conjunction $\psi_1(z_0, \ldots, z_k) \wedge \psi_2(z_0, \ldots, z_k)$ of two $\overrightarrow{\exists}\forall^g$-formulas. In this case we have that:

$$\psi_1 \equiv \exists x_0 \ldots \exists x_n \big(x_0 < \cdots < x_n \wedge z_0 = x_0 \wedge \ldots\big)$$
$$\psi_2 \equiv \exists x_{n+1} \ldots \exists x_m (x_{n+1} < \cdots < x_m \wedge z_0 = x_{n+1} \wedge \ldots)$$

Since the set of quantified variables in ψ_1 is disjoint from the set of quantified variables in ψ_2, we can distribute the existential quantifiers over the conjunction $\psi_1 \wedge \psi_2$, obtaining:

$$\begin{aligned}\psi_1 \wedge \psi_2 \equiv\ & \exists x_0 \ldots \exists x_n \exists x_{n+1} \ldots \exists x_m \\ & \big(x_0 < \cdots < x_n \wedge x_{n+1} < \cdots < x_m \wedge \\ & z_0 = x_0 \wedge z_0 = x_{n+1} \wedge \ldots\big)\end{aligned}$$

Note that we can identify x_0 and x_{n+1}, obtaining:

$$\begin{aligned}\psi_1 \wedge \psi_2 \equiv\ & \exists x_0 \ldots \exists x_n \exists x_{n+2}, \ldots \exists x_m \\ & \big(x_0 < \cdots < x_n \wedge x_0 < x_{n+2} < \cdots < x_m \wedge \\ & z_0 = x_0 \wedge \bigwedge_{i=1}^{k} (z_i = x_{j_i''}) \wedge \\ & \bigwedge_{i=0}^{m} \alpha_i(x_i) \wedge \\ & \bigwedge_{i=1}^{m} \forall y (x_{i-1} < y < x_i \rightarrow \beta_i(y))\big)\end{aligned}$$

Now, to turn this formula into a disjunction of $\vec{\exists}\forall^g$-formulas, we consider all the possible interleavings of the variables that respect the two imposed orderings and explode the formula into a disjunction that consider each such interleaving. Let $X = \{x_0, \ldots, x_n, x_{n+2}, \ldots, x_m\}$ and let Π be the set of all the permutations of X compatible with the orderings $x_0 < \cdots < x_n$ and $x_0 < x_{n+1} < \cdots < x_m$. Note that $\pi(0) = 0$. Now, $\psi_1 \wedge \psi_2$ becomes the disjunction of a set of $\vec{\exists}\forall^g$-formulas ψ_π,

for each $\pi \in \Pi$, defined as:

$$\begin{aligned}\psi_\pi \equiv \exists x_{\pi(0)} \ldots \exists x_{\pi(m)} \\ \big(x_{\pi(0)} < \cdots < x_{\pi(m)} \wedge \\ z_0 = x_0 \wedge \bigwedge_{i=1}^{k} (z_i = x_{\pi(j_i'')}) \wedge \\ \bigwedge_{i=0}^{m} \alpha_i(x_i) \wedge \\ \bigwedge_{i=0}^{m} \forall y (x_{\pi(i-1)} < y < x_{\pi(i)} \rightarrow \beta_i^*(y))\big)\end{aligned}$$

where β_i^* suitably combines the formulas β according to the interleaving of the orderings of the original variables, and is defined as follows:

$$\beta_i^* = \begin{cases} \beta_{\pi(i)} & \text{if both } \pi(i), \pi(i-1) \leq n \\ & \text{or both } \pi(i), \pi(i-1) > n \\ \beta_{\pi(i)} \wedge \beta_{\pi(i-1)} & \text{if } \pi(i) \leq n \text{ and } \pi(i-1) > n \\ & \text{or } \textit{vice versa} \end{cases}$$

Then we have that $\psi_1 \wedge \psi_2 \equiv \bigvee_{\pi \in \Pi}(\psi_\pi)$, which is a disjunction of $\overrightarrow{\exists}\forall^g$-formulas.

3. Let $\phi(z_0, \ldots, z_m) \equiv \exists z_{m+1} \,.\, (z_i < z_{m+1} \wedge \phi_1(z_0, \ldots, z_m, z_{m+1}))$, for some $0 \leq i \leq m$. By the inductive hypothesis, this is equivalent to the formula $\exists z_{m+1}(z_i < z_{m+1} \wedge \bigvee_{k=0}^{j} \psi_k(z_0, \ldots, z_m, z_{m+1}))$, where $\psi_k(z_0, \ldots, z_m, z_{m+1})$ is a $\overrightarrow{\exists}\forall^g$-formula, for each $0 \leq k \leq j$, that is:

$$\begin{aligned}\exists z_{m+1} \,.\, (z_i < z_{m+1} \wedge \\ \bigvee_{k=0}^{j} (\exists x_0 \ldots \exists x_{n_k} \psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})))\end{aligned}$$

By distributing the conjunction over the disjunction, we ob-

tain:

$$\exists z_{m+1} \,.\, (\bigvee_{k=0}^{j} ((z_i < z_{m+1}) \wedge \exists x_0 \ldots \exists x_{n_k} \psi'_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})))$$

and by distributing the existential quantifier over the disjunction, we have:

$$\bigvee_{k=0}^{j} (\exists z_{m+1}((z_i < z_{m+1}) \wedge \exists x_0 \ldots \exists x_{n_k} \psi'_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})))$$

Since the subformula $z_i < z_{m+1}$ does not contain the variables $x_0, \ldots, x_n$, we can push it inside the existential quantification, obtaining:

$$\bigvee_{k=0}^{j} (\exists z_{m+1} \,.\, \exists x_0 \ldots \exists x_{n_k} \,.\, ((z_i < z_{m+1}) \wedge \psi'_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})))$$

Now we divide in cases:

(a) suppose that the formula $\psi'_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ contains the following conjuncts: $z_i = x_{l_i}$ and $z_{m+1} = x_{l_{m+1}}$, with $l_i = l_{m+1}$. It holds that these formulas are in contradiction with the formula $z_i < z_{m+1}$, that is:

$$(z_i < z_{m+1}) \wedge (z_i = x_{l_i}) \wedge (z_{m+1} = x_{l_{m+1}}) \equiv \bot$$

Therefore, in this case, the disjunct

$$(z_i < z_{m+1}) \wedge \psi'_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$$

is equivalent to $\bot$, and thus can be safely removed from the disjunction.

(b) suppose that the formula $\psi'_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ contains the following conjuncts: $z_i = x_{l_i}$, $z_{m+1} = x_{l_{m+1}}$

(with $l_i \neq l_{m+1}$), and $x_{l_{m+1}} < \cdots < x_{l_i}$. As in the previous case, it holds that:

$$\begin{aligned}&(z_i < z_{m+1}) \wedge (z_i = x_{l_i}) \wedge \\ &(z_{m+1} = x_{l_{m+1}}) \\ &\wedge (x_{l_{m+1}} < \cdots < x_{l_i})\end{aligned}$$

is equivalent to $\bot$. Thus, also in this case, this disjunct can be safely removed from the disjunction.

(c) otherwise, it holds that the formula $\psi'_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ contains the following conjuncts: $z_i = x_{l_i}$, $z_{m+1} = x_{l_{m+1}}$ (with $l_i \neq l_{m+1}$), and $x_{l_i} < \cdots < x_{l_{m+1}}$. Therefore, the subformula $z_i < z_{m+1}$ is redundant, and can be safely removed from $\psi'_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$. The resulting formula is a $\overrightarrow{\exists}\forall^g$-formula.

After the previous transformation, we obtain:

$$\bigvee_{k=0}^{j'} (\exists z_{m+1} \,.\, \exists x_0 \ldots \exists x_{n_k} \,.\, \psi''_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k}))$$

Finally, since each formula $\psi''_k(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ contains the conjunct $z_{m+1} = x_{l_{m+1}}$, we can safely remove the quantifier $\exists z_{m+1}$. We obtain the formula:

$$\bigvee_{k=0}^{j'} (\exists x_0 \ldots \exists x_{n_k} \,.\, \psi''_k(z_0, \ldots, z_m, x_0, \ldots, x_{n_k}))$$

which is a disjunction of $\overrightarrow{\exists}\forall^g$-formulas.

4. Let $\phi(z_0, \ldots, z_m)$ be the formula

$$\forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \phi_1(z_0, \ldots, z_m, z_{m+1}))$$

for some $0 \leq i, j \leq m$. By the induction hypothesis we know that ϕ_1 is equivalent to a disjunction $\bigvee_k \psi_k$ where ψ_k are $\overrightarrow{\exists}\forall^g$-

formulas, *i.e.*, each ψ_k is of the form:

$$\psi_k \equiv \exists x_0, \ldots, x_n \big(x_0 < \ldots < x_n \wedge z_0 = x_0 \wedge \bigwedge_{l=1}^{m+1} (z_l = x_{u_l}) \wedge \bigwedge_{l=0}^{n} \alpha_l(x_l) \wedge \bigwedge_{l=1}^{n} \forall y (x_{l-1} < y < x_l \rightarrow \beta_l(y)) \big)$$

We now note that we can suppose *w.l.o.g.* that the ordering constraint and the binding constraint of ψ_k imply that z_i, z_{m+1} and z_j are ordered consecutively, *i.e.*, $z_i < z_{m+1} < z_j$ with no other variable inbetween. That is because otherwise the constraints would be in conflict with the guard of the universal quantification and the disjunct could be removed from the disjunction. Take for example a disjunct of ψ_k with an ordering constraint of the type $z_i < z_h < z_{m+1}$, for some h. The existence of such a z_h is not guaranteed for each z_{m+1} between z_i and z_j because when $z_{m+1} = z_i + 1$ there is no value between z_i and $z_i + 1$ (we are on discrete time models). That said, we can now isolate all the parts of ψ_k that talk about z_{m+1}, bringing them out of the existential quantification, obtaining $\psi_k \equiv \theta_k \wedge \eta_k$, where:

$$\begin{aligned} \theta_k \equiv\ & z_i < z_{m+1} < z_j \wedge \\ & \alpha(z_{m+1}) \wedge \\ & \forall y (z_i < y < z_{m+1} \rightarrow \beta(y)) \wedge \\ & \forall y (z_{m+1} < y < z_i \rightarrow \beta'(y)) \end{aligned}$$

$$\eta_k \equiv \exists x_0, \ldots, x_n \big(x_0 < \ldots < x_n \wedge z_0 = x_0 \wedge \bigwedge_{l=1}^{m} (z_l = x_{u_l}) \wedge \bigwedge_{\substack{l=0 \\ l \neq u_{m+1}}}^{n} \alpha_l(x_l) \wedge \bigwedge_{\substack{l=1 \\ l-1 \neq u_i \\ l \neq u_j}}^{n} \forall y (x_{l-1} < y < x_l \rightarrow \beta_l(y)) \big)$$

Now, we have $\phi \equiv \forall z_{m+1} (z_i < z_{m+1} < z_j \rightarrow \bigvee_k (\theta_k \wedge \eta_k))$. We can distribute the head of the implication over the disjunction, and then over the conjunction, obtaining that ϕ is equivalent to:

$$\forall z_{m+1} \big(\bigvee_k ((z_i < z_{m+1} < z_j \rightarrow \theta_k) \wedge (z_i < z_{m+1} < z_j \rightarrow \eta_k)) \big)$$

In order to simplify the exposition, we now show how to proceed in the case of two disjuncts, which is easily generalizable. So suppose we have that ϕ is equivalent to:

$$\forall z_{m+1} \left(\bigvee \begin{matrix} (z_i < z_{m+1} < z_j \rightarrow \theta_1) \wedge (z_i < z_{m+1} < z_j \rightarrow \eta_1) \\ (z_i < z_{m+1} < z_j \rightarrow \theta_2) \wedge (z_i < z_{m+1} < z_j \rightarrow \eta_2) \end{matrix} \right)$$

Now we can a) distribute the disjunction over the conjunction (*i.e.*, convert in conjunctive normal form in the case of multiple disjuncts), b) factor out the head of the implications and c) distribute the universal quantification over the conjunction, obtaining:

$$\phi \equiv \left(\begin{matrix} \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1 \vee \theta_2) \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1 \vee \eta_2) \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \eta_1 \vee \theta_2) \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \eta_1 \vee \eta_2) \end{matrix} \right)$$

Now, note that η_1 and η_2 do not contain z_{m+1} as a free variable, because we factored out all the parts mentioning z_{m+1} into θ_1 and θ_2 before. Therefore we can push them out from the universal quantifications, obtaining:

$$\phi \equiv \left(\begin{matrix} \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1 \vee \theta_2) \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1) \vee \eta_2 \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_2) \vee \eta_1 \\ \wedge \neg\exists z_{m+1}(z_i < z_{m+1} < z_j) \vee \eta_1 \vee \eta_2 \end{matrix} \right)$$

Now, note that $\neg\exists z_{m+1}(z_i < z_{m+1} < z_j)$ is equivalent to $z_i = z_j \vee z_j = z_i + 1$, which is the disjunction of two formulas that can be turned into $\overrightarrow{\exists}\forall^g$-formulas. Since both η_1 and η_2 are already $\overrightarrow{\exists}\forall^g$-formulas and since we already know how to deal with conjunctions and disjunctions of $\overrightarrow{\exists}\forall^g$-formulas, it remains to show that the universal quantifications in the formula above can be turned into $\overrightarrow{\exists}\forall^g$-formulas. Take $\forall z_{m+1}(z_i < z_{m+1} <$

$z_j \to \theta_1$), *i.e.*:

$$\forall z_{m+1} \left(z_i < z_{m+1} < z_j \to \begin{array}{l} z_i < z_{m+1} < z_j \\ \wedge\, \alpha(z_{m+1}) \\ \wedge\, \forall y(z_i < y < z_{m+1} \to \beta(y)) \\ \wedge\, \forall y(z_{m+1} < y < z_j \to \beta'(y)) \end{array} \right)$$

Note that the first conjunct of the consequent can be removed, since it is redundant. Now, this formula is requesting $\beta(y)$ for all y between z_i and z_{m+1}, but with z_{m+1} that ranges between z_i and $z_j - 1$, hence effectively requesting $\beta(y)$ to hold between z_i and z_j. Similarly for $\beta'(y)$, which has to hold for all y between $z_i + 1$ and z_j.

Hence, it is equivalent to:

$$\begin{array}{l} z_i = z_j \\ \vee\, z_j = z_i + 1 \\ \vee\, \exists x_{i+1}(z_i < x_{i+1} \wedge x_{i+1} = z_i + 1 \wedge z_j = x_{i+1} + 1 \wedge \alpha(x_{i+1})) \\ \vee\, \exists x_i \exists x_{i+1} \exists x_{j-1} \exists x_j \left(\begin{array}{l} x_i < x_{i+1} < x_{j-1} < x_j \\ \wedge\, z_i = x_i \wedge z_j = x_j \\ \wedge\, \alpha(x_{i+1}) \wedge \alpha(x_{j-1}) \\ \wedge\, \forall y(x_i < y < x_{i+1} \to \bot) \\ \wedge\, \forall y(x_{j-1} < y < x_j \to \bot) \\ \wedge\, \forall y(x_i < y < x_{j-1} \to \alpha(y) \wedge \beta(y)) \\ \wedge\, \forall y(x_{i+1} < y < x_j \to \alpha(y) \wedge \beta'(y)) \end{array} \right) \end{array}$$

which is a disjunction of a $\overrightarrow{\exists}\forall^g$-formula and others that can be turned into disjunctions of $\overrightarrow{\exists}\forall^g$-formulas. The reasoning is at all similar for $\forall z_{m+1}(z_i < z_{m+1} < z_j \to \theta_1 \vee \theta_2)$. □

Any coSafety-FO formula can be translated into a disjunction of $\overrightarrow{\exists}\forall^g$-formulas by Lemma 9, and then to a coSafety-LTL$(-\widetilde{\mathsf{X}})$ formula by Lemma 8. Together with Lemma 7, we obtain the following.

Corollary 8. $[\![\text{coSafety-FO}]\!]^{<\omega} = [\![\text{coSafety-LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega}$

We are now ready to state the main result of this section.

Theorem 27. $[\![\mathsf{coSafety\text{-}LTL}]\!] = [\![\mathsf{coSafety\text{-}FO}]\!]$

Proof. We know that $[\![\mathsf{coSafety\text{-}LTL}]\!] = [\![\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega}\cdot(2^{\Sigma})^{\omega}$ by Observation 1 and Lemma 5. Since $[\![\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega} = [\![\mathsf{coSafety\text{-}FO}]\!]^{<\omega}$ by Corollary 8, we have that $[\![\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega}\cdot(2^{\Sigma})^{\omega} = [\![\mathsf{coSafety\text{-}FO}]\!]^{<\omega}\cdot(2^{\Sigma})^{\omega}$. Then, by Lemma 6 we have that $[\![\mathsf{coSafety\text{-}FO}]\!]^{<\omega}\cdot(2^{\Sigma})^{\omega} = [\![\mathsf{coSafety\text{-}FO}]\!]$, hence $[\![\mathsf{coSafety\text{-}LTL}]\!] = [\![\mathsf{coSafety\text{-}FO}]\!]$. □

Corollary 9. $[\![\mathsf{Safety\text{-}LTL}]\!] = [\![\mathsf{Safety\text{-}FO}]\!]$

4.2 Safety-FO captures the safety fragment of LTL

In this section, we prove that coSafety-FO captures LTL-definable co-safety languages. By duality, we have that Safety-FO captures LTL-definable safety languages, and by the equivalence shown in the previous section, this provides a novel proof of the fact that Safety-LTL captures the safety fragment of LTL. We start by characterizing co-safety languages in terms of LTL over finite words.

Lemma 10. $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY} = [\![\mathsf{LTL}]\!]^{<\omega}\cdot(2^{\Sigma})^{\omega}$

Proof. ($\subseteq$) By Theorem 18 we know that each language $\mathcal{L} \in [\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$ is definable by a formula of the form $\mathsf{F}\alpha$ where $\alpha \in \mathsf{LTL}{+}\mathsf{P_P}$. Hence for each $\sigma \in \mathcal{L}$ there exists an n such that $\sigma, n \models^{<\omega} \alpha$, hence $\sigma_{[0,n]}, n \models^{<\omega} \alpha$. Note that $\sigma_{[n+1,\infty]}$ is unconstrained, and thus $\mathcal{L} = \mathcal{L}^{<\omega}(\alpha)\cdot(2^{\Sigma})^{\omega}$. Since, by Proposition 6, $\mathsf{LTL}{+}\mathsf{P_P}$ is equivalent to LTL under finite words interpretation, there exists a formula $\beta \in \mathsf{LTL}$ such that $\mathcal{L}^{<\omega}(\alpha) = \mathcal{L}^{<\omega}(\beta)$. Hence $\mathcal{L} = \mathcal{L}^{<\omega}(\beta)\cdot(2^{\Sigma})^{\omega}$, and thus $\mathcal{L} \in [\![\mathsf{LTL}]\!]^{<\omega}\cdot(2^{\Sigma})^{\omega}$.

($\supseteq$) Given $\mathcal{L} \in [\![\mathsf{LTL}]\!]^{<\omega}\cdot(2^{\Sigma})^{\omega}$, we know $\mathcal{L} = \mathcal{L}^{<\omega}(\beta)\cdot(2^{\Sigma})^{\omega}$ for some LTL formula β. Since, by Proposition 6, $\mathsf{LTL}{+}\mathsf{P_P}$ is equivalent to LTL under finite words interpretation, there exists a formula $\alpha \in \mathsf{LTL}{+}\mathsf{P_P}$ such that $\mathcal{L}^{<\omega}(\beta) = \mathcal{L}^{<\omega}(\alpha)$. Hence, for each $\sigma \in \mathcal{L}$ there is an n such that $\sigma_{[0,n]}, n \models \alpha$, *i.e.*, $\sigma \models \mathsf{F}\alpha$. Therefore, by Theorem 18, $\mathcal{L} \in [\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$, and this in turn implies that $[\![\mathsf{LTL}]\!]^{<\omega}\cdot(2^{\Sigma})^{\omega} \subseteq [\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$. □

Then, we observe that, in LTL interpreted over finite word, universal temporal operators can be defined in terms of the existential

ones, and *vice versa*. As before, the *weak next* operator plays a crucial role.

Lemma 11. $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{Safety\text{-}LTL}]\!]^{<\omega} = [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega}$

Proof. Since Safety-LTL and coSafety-LTL are fragments of LTL, we only need to show one direction, *i.e.*, that $[\![\mathsf{LTL}]\!]^{<\omega} \subseteq [\![\mathsf{Safety\text{-}LTL}]\!]^{<\omega}$ and $[\![\mathsf{LTL}]\!]^{<\omega} \subseteq [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega}$. At first, we show that universal temporal operators are not needed over finite words. For each LTL formula ϕ, we can build an equivalent coSafety-LTL with only existential temporal operators. The *globally* operator can be replaced by means of an *until* operator whose existential part always refers to the last position of the word. In turn, this can be done with the formula $\widetilde{\mathsf{X}}\bot$, which is true only at the final position:

$$\mathsf{G}\phi \equiv \phi \,\mathsf{U}\, (\phi \wedge \widetilde{\mathsf{X}}\bot)$$

Similarly, the *release* operator can be expressed by means of a *globally* operator in disjunction with an *until* operator:

$$\begin{aligned}\phi_1 \,\mathsf{R}\, \phi_2 &\equiv \mathsf{G}\phi_2 \vee (\phi_2 \,\mathsf{U}\, (\phi_1 \wedge \phi_2)) \\ &\equiv \big(\phi_2 \,\mathsf{U}\, (\phi_2 \wedge \widetilde{\mathsf{X}}\bot)\big) \vee \big(\phi_2 \,\mathsf{U}\, (\phi_1 \wedge \phi_2)\big)\end{aligned}$$

Hence $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega}$. Now, if we exploit the fact that the *eventually* and *until* operators are the negation of the *globally* and the *release* operators, we obtain:

$$\begin{aligned}\mathsf{F}\phi &\equiv \phi \,\mathsf{R}\, (\phi \vee \mathsf{X}\top) \\ \phi_1 \,\mathsf{U}\, \phi_2 &\equiv \phi_2 \,\mathsf{R}\, (\phi_2 \vee \mathsf{X}\top) \wedge \phi_2 \,\mathsf{R}\, (\phi_1 \vee \phi_2)\end{aligned}$$

Hence $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{Safety\text{-}LTL}]\!]^{<\omega}$. □

Then, we relate coSafety-LTL on finite words and coSafety-FO.

Lemma 12. $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{coSafety\text{-}FO}]\!]$

Proof. ($\subseteq$) We have that $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} = [\![\mathsf{LTL}]\!]^{<\omega}$ by Lemma 11, and this implies that $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$, and $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{S1S[FO]}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$ by Theorem 13. Now, let $\phi \in \mathsf{S1S[FO]}$, and suppose *w.l.o.g.* that ϕ is in *negated normal form*. We define the formula $\phi'(x, y)$, where x and y are two fresh variables that do not occur in ϕ, as the formula obtained from ϕ by a) replacing each subformula of ϕ of type $\exists z \phi_1$ with $\exists z(x \leq z \wedge \phi_1)$,

and b) by replacing each subformula of ϕ of type $\forall z \phi_1$ with $\forall z(x \leq z < y \rightarrow \phi_1)$. Now, consider the formula $\psi \equiv \exists y(x \leq y \wedge \phi'(x, y))$. Note that ψ is a coSafety-FO formula. When interpreted over *infinite* words, the models of ψ are exactly those containing a prefix that belongs to $\mathcal{L}^{<\omega}(\phi)$, with the remaining suffix unconstrained, that is $\mathcal{L}(\psi) = \mathcal{L}^{<\omega}(\phi) \cdot (2^{\Sigma})^{\omega}$, hence $[\![\text{S1S[FO]}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} \subseteq [\![\text{coSafety-FO}]\!]$, and this implies that $[\![\text{coSafety-LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} \subseteq [\![\text{coSafety-FO}]\!]$.

($\supseteq$) We know by Lemma 6 that $[\![\text{coSafety-FO}]\!] = [\![\text{coSafety-FO}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. Since coSafety-FO formulas are also S1S[FO] formulas, we have $[\![\text{coSafety-FO}]\!] \subseteq [\![\text{S1S[FO]}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. By Theorem 13 and Lemma 11, we obtain that $[\![\text{coSafety-FO}]\!] \subseteq [\![\text{coSafety-LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. □

We are ready now to state the main result.

Theorem 28. $[\![\text{LTL}]\!] \cap \text{coSAFETY} = [\![\text{coSafety-FO}]\!]$

Proof. We know that $[\![\text{LTL}]\!] \cap \text{coSAFETY} = [\![\text{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$ by Lemma 10. Then, by Lemma 11 we know that $[\![\text{LTL}]\!]^{<\omega} = [\![\text{coSafety-LTL}]\!]^{<\omega}$, and this in turn implies that $[\![\text{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\text{coSafety-LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. Since $[\![\text{coSafety-LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\text{coSafety-FO}]\!]$ by Lemma 12, we conclude that $[\![\text{LTL}]\!] \cap \text{coSAFETY} = [\![\text{coSafety-FO}]\!]$. □

This result together with Theorem 27 allow us to conclude the following.

Theorem 29. $[\![\text{Safety-LTL}]\!] = [\![\text{LTL}]\!] \cap \text{SAFETY}$

Note that by Observation 1 and Lemma 5 on one hand, and by Lemmas 10 and 11 on the other, the question of whether $[\![\text{Safety-LTL}]\!] = [\![\text{LTL}]\!] \cap \text{SAFETY}$ can be reduced to the question of whether $[\![\text{coSafety-LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\text{coSafety-LTL}(-\widetilde{\text{X}})]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. If coSafety-LTL and coSafety-LTL$(-\widetilde{\text{X}})$ were equivalent over finite words, this would prove Theorem 29 without the need of coSafety-FO. However, we can prove this is not the case.

Theorem 30. $[\![\text{coSafety-LTL}]\!]^{<\omega} \neq [\![\text{coSafety-LTL}(-\widetilde{\text{X}})]\!]^{<\omega}$

Proof. Note that in coSafety-LTL$(-\widetilde{\text{X}})$ we only have existential temporal modalities and we cannot hook the final position of the word because the *weak next* operator is not available. For these reasons,

given a coSafety-LTL($-\widetilde{\mathsf{X}}$) formula ϕ, with a simple structural induction we can prove that for each $\sigma \in (2^{\Sigma})^{+}$ such that $\sigma \models \phi$, it holds that $\sigma\sigma' \models \phi$ for any $\sigma' \in (2^{\Sigma})^{+}$, *i.e.*, all the extensions of σ satisfy ϕ as well. This implies that $\mathcal{L}^{<\omega}(\phi)$ is either empty (*i.e.*, if ϕ is unsatisfiable) or infinite. Instead, by using the *weak next* operator to hook the last position of the word, we can describe a finite non-empty language, for example as in the formula $\phi \equiv a \wedge \mathsf{X}(a \wedge \widetilde{\mathsf{X}}\bot)$. The language of ϕ is $\mathcal{L}(\phi) = \{\mathsf{aa}\}$, including exactly one word, hence $\mathcal{L}(\phi)$ cannot be described without the *weak next* operator. □

Note that Theorem 30 does *not* contradict Theorem 29, that is, it does not imply that $[\![\text{coSafety-LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} \neq [\![\text{coSafety-LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. For example, consider again the formula $a \wedge \mathsf{X}(a \wedge \widetilde{\mathsf{X}}\bot)$. It cannot be expressed without the *weak next* operator, yet it holds that: $\mathcal{L}^{<\omega}(a \wedge \mathsf{X}(a \wedge \widetilde{\mathsf{X}}\bot)) \cdot (2^{\Sigma})^{\omega} = \mathcal{L}^{<\omega}(a \wedge \mathsf{X}a) \cdot (2^{\Sigma})^{\omega}$.

A summary of the results provided by this chapter is given in Fig. 4.1. The expressiveness of Safety-FO and coSafety-FO over infinite words is depicted in Fig. 4.2. A comparison between the expressive power of coSafety-LTL, coSafety-LTL($-\widetilde{\mathsf{X}}$), Safety-LTL, coSafety-FO, Safety-FO and other formalisms over finite words interpretation is shown in Fig. 4.3.

4.3 Conclusions

In this chapter, we gave a characterisation of the first-order definable safety and co-safety languages, by means of two syntactical fragments of first-order logic, namely Safety-FO and coSafety-FO. These logics support a particularly constrained kind of existential and universal quantifications that capture the essence of safety and co-safety languages.

The core theorem establishes a correspondence between Safety-FO (resp., coSafety-FO) and Safety-LTL (resp., coSafety-LTL), and thus it can be viewed as a special version of Kamp's theorem for safety (resp., co-safety) properties. We used this theorem for proving the *expressive completeness* of coSafety-FO (resp. Safety-FO) with respect to the semantically co-safety (resp. safety) fragment of S1S[FO]. Thanks to these new fragments, we were able to provide a novel, compact, and self-contained proof of the fact that Safety-LTL captures LTL-definable safety languages. Such a result was previously proved by Chang *et al.* [40], but in terms of the properties

of a non-trivial transformation from star-free languages to LTL by Zuck [202]. As a by-product, we provided a number of results that relate the considered languages when interpreted over finite and infinite words. In particular, we highlighted the expressive power of the *weak tomorrow* temporal modality, showing that it turns out to be essential in coSafety-LTL over finite words.

Different equivalent characterisations of LTL are known, in terms of (i) first-order logics, (ii) regular expressions, (iii) automata, and (iv) monoids (they have been summarised by Thomas in [192]). This work focuses on the first item, but for LTL-definable safety languages. A natural follow-up would be to investigate the other items, looking for what kind of automata (resp., regular expressions, monoids) captures exactly safety and co-safety LTL-definable languages. While on finite traces simple characterizations in terms of automata and syntactic monoids exist, the infinite-traces scenario is more complex: there exists a characterization of LTL in terms of counter-free automata [149] and the one for safety ω-regular languages seems not to be difficult (see *e.g.*, terminal automata [186, 39]), but their combination requires to have a canonical (minimal) representation of a (Muller/Rabin/Streett) automata corresponding to any ω-regular language.

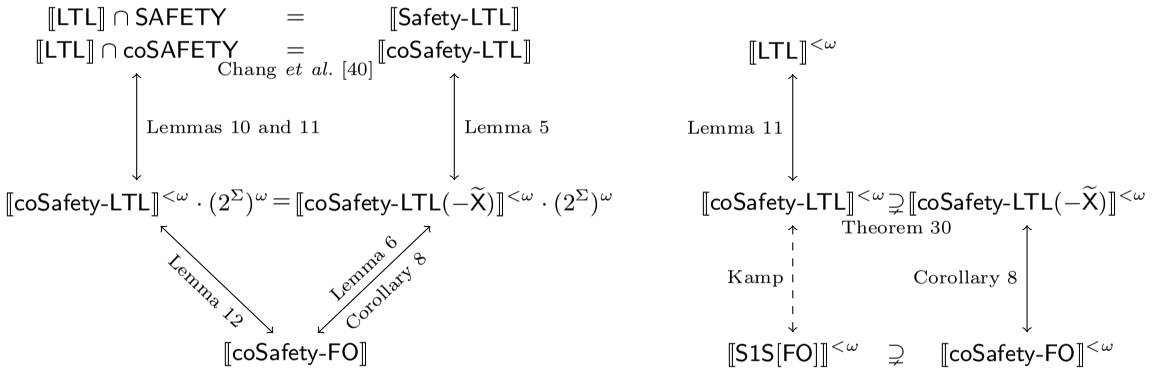


Figure 4.1: Summary of the results of this chapter, about languages over infinite words on the left, and over finite words on the right. Solid arrows are own results. Dashed arrows are known from literature.

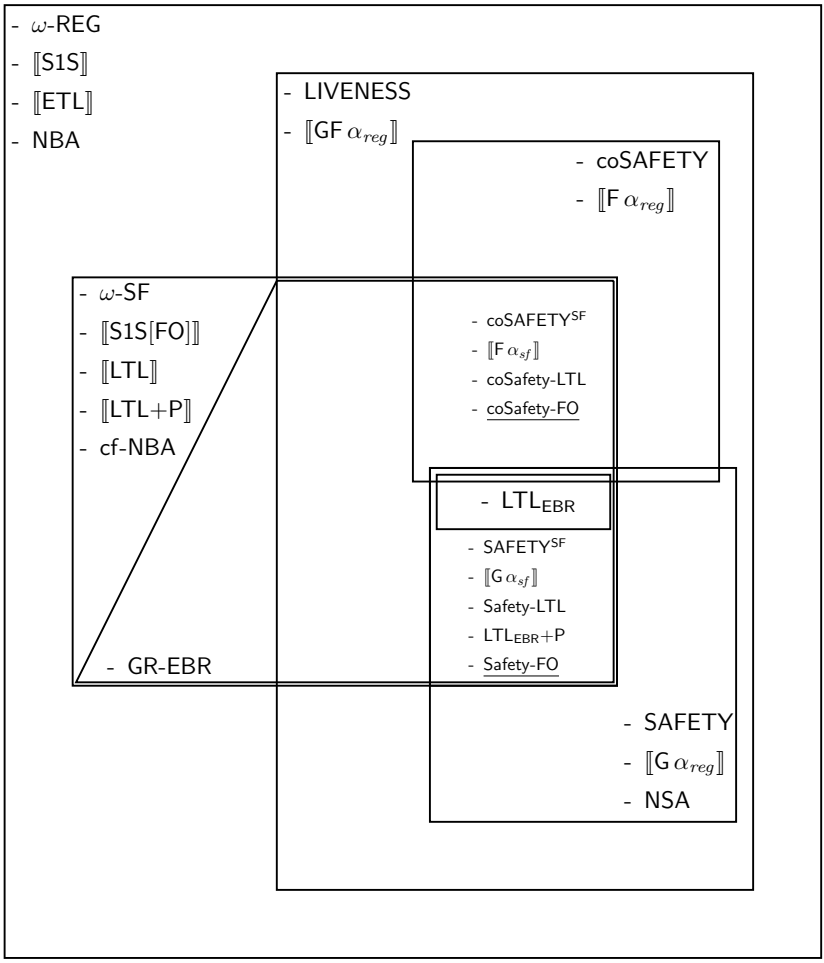


Figure 4.2: Extension of Fig. 3.2 that takes into account also of the expressive power of Safety-FO and coSafety-FO (underlined).

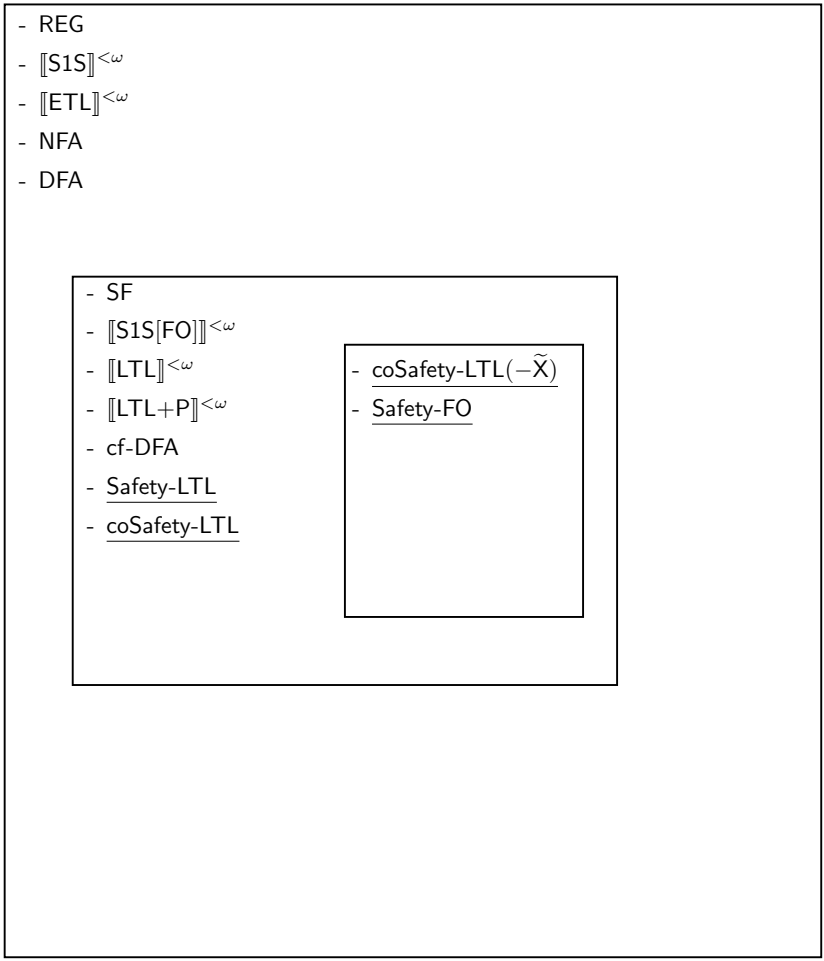


Figure 4.3: Comparison between the expressive power of coSafety-LTL with and without *weak next*, Safety-LTL and Safety-FO under finite word interpretation. For sake of clarity, we underlined the logics we add with respect to Fig. 2.3.

Part III

Problems and Algorithms

CHAPTER

5

BACKGROUND

In this chapter, we give the necessary background for the *Problems and Algorithms part*. We start with the *satisfiability* problem, for which we recap the theoretical complexity of this problem for the logics of S1S, S1S[FO] and LTL+P. From now on, unless otherwise stated, for all logics under consideration, we interpret them always over *infinite* linear orders (*equiv.* infinite words). We then take a look at some classical techniques for solving the satisfiability problem from LTL+P specifications, with particular attention to the details of the one-pass and tree-shaped tableau for LTL [163, 106], since one of the contributions of this thesis is a symbolic encoding for this system.

We then define Kripke structures and the *model checking* problem for LTL. After defining what a symbolic representation is, we recap the main ideas behind Bounded Model Checking [17] and K-Liveness [57]. We show also how LTL satisfiability can be reduced to LTL model checking.

We then focus on realizability. We define the basic notion of *strategy* and the formal definition of *realizability* and *reactive syn-*

thesis. Also in this case, we will take a look at classical techniques for LTL+P realizability [160] and also on *bounded synthesis* [92, 88]. Some attention will be devoted also to safety synthesis.

5.1 LTL Satisfiability

Given a logic $\mathbb{L}$, we have already seen in Chapter 2 that the *satisfiability problem* for $\mathbb{L}$ is the problem of finding whether there exists a model of a given formula $\phi \in \mathbb{L}$. Recall also that, since we are mainly dealing with temporal logics and related formalisms, the structures over which the formulas are interpreted are always (finite or infinite) *linear orders*, or, equivalently, (finite or infinite) *words*.

In Notation 2, we introduced a notation for differentiating the *syntax* from the *semantics* of a given logic. We briefly recap here the notation. Given a logic $\mathbb{L}$, we denote with $\mathbb{L}$ also the set of formulas that *syntactically* belong to $\mathbb{L}$, while with $[\![\mathbb{L}]\!]$ we denote the set of languages over infinite words that can be expressed with a formula in $\mathbb{L}$ (*semantic* definition), *i.e.*,

$$[\![\mathbb{L}]\!] \coloneqq \{\mathcal{L} \mid \mathcal{L} = \mathcal{L}(\phi), \phi \in \mathbb{L}\}$$

The satisfiability problem of $\mathbb{L}$ can be reformulated as the problem of establishing whether $\mathcal{L}(\phi) \stackrel{?}{=} \varnothing$, given a formula $\phi \in \mathbb{L}$.

5.1.1 Complexity

Satisfiability is one of the first theoretical questions that are answered for a particular logic. In fact, when one talks about the *complexity* of a given logic, he usually refers to the complexity of the satisfiability problem for that logic.

We start by recalling the complexity of S1S and S1S[FO] (recall that, from now on, we will focus only on the infinite words interpretion, unless otherwise stated).

Theorem 31 (Complexity of S1S and S1S[FO] [185])**.** *The satisfiability of* S1S *and* S1S[FO] *is nonelementary.*

In [179], Sistla and Clarke prove that finding whether an LTL (or LTL+P) formula admits at least one model is PSPACE-complete.

Theorem 32 (Complexity of LTL [179, 135])**.** *The satisfiability problem of* LTL *(and* LTL+P*) is* PSPACE*-complete.*

Theorems 31 and 32 are very interesting. In fact, recall from Theorem 13 and Fig. 2.2 that $[\![\mathsf{LTL}]\!] = [\![\mathsf{S1S[FO]}]\!]$, that is, LTL and S1S[FO] have the same expressive power. Nevertheless, the nonelementary complexity of S1S[FO] is avoided by LTL by the careful choice of its modal operators.

The same happens with *Extended Temporal Logic* (ETL). Recall from Section 2.4 that ETL is the extension of LTL with operators corresponding to regular expressions, and that $[\![\mathsf{ETL}]\!] = [\![\mathsf{S1S}]\!]$ (Theorem 14). Despite being expressively equivalent, the complexity of ETL is PSPACE-complete (the same of LTL+P), while the complexity of S1S is nonelementary. This is not the case for other extensions of temporal logics: for example LTL+P extended with first-order quantifications [142] is expressively equivalent to S1S but its complexity is still nonelementary.

Regarding the Safety-LTL logic, we weren't able to find any paper in the literature addressing the complexity of its satisfiability problem. Our conjecture is that it is still PSPACE-complete, like for LTL.

We now take a look at four methods for solving the satisfiability problem from LTL+P specifications.

5.1.2 The automata-theoretic algorithm

One of the classical methods for solving satisfiability of LTL+P specifications is to reduce the problem to the emptiness check of a Büchi automaton equivalent to the starting formula. There are a lot of papers describing this technique and related optimizations for producing small Büchi automata [196, 129, 127, 63, 181, 189, 198, 98]. The classical method, without optimizations, can be summarized as follows:

- take an LTL+P formula ϕ; let n be its dimension;
- build a Büchi automaton $\mathcal{A}(\phi)$ such that $\mathcal{L}(\mathcal{A}(\phi)) = \mathcal{L}(\phi)$; in the worst case, the number of states of the automaton is $\mathcal{O}(2^n)$;
- check whether $\mathcal{L}(\mathcal{A}(\phi)) \stackrel{?}{=} \varnothing$, that is check the emptiness of the automaton; if it is empty, then ϕ is unsatisfiable, otherwise ϕ is satisfiable.

Recall from Section 2.2 that checking the emptiness of $\mathcal{A}(\phi)$ amounts to checking whether there exists a directed path from one of its initial states to one of its final state, and from such a final state to itself. Since this is a reachability problem, it can be solved in nondeterministic logarithmic space [196, 168], that is $\mathcal{O}(\log_2(2^n))$: therefore, this step takes $\mathcal{O}(n)$ space. Since the emptiness check can be performed on-the-fly (*i.e.*, while building the automaton $\mathcal{A}(\phi)$), the whole algorithm runs in nondeterministic polynomial space, and, by Savitch Theorem [168], this is still in PSPACE. Since LTL+P satisfiability is PSPACE-complete, this shows the optimality of this algorithm.

The construction of the NBA (nondeterministic Büchi automaton, recall Definition 11) starting from an LTL+P formula can be done in several ways. Here we recall two of them.

Using Alternating Büchi Automata

In [196, 129], the construction passes through *Alternating Büchi automata*. These are basically Büchi automata except from the fact that the transition relation is specified in terms of Boolean formulas. For example, the transition relation:

$$\delta(q, a) = (q_1 \wedge q_2) \vee (q_3 \wedge q_4)$$

means that, starting from state q and reading letter a, the automaton can reach any *set of states* that is a model of the above Boolean formula, *i.e.*, either the set $\{q_1, q_2\}$, or $\{q_3, q_4\}$, or $\{q_1, q_2, q_3, q_4\}$. In particular, this means that a run of an Alternating Büchi automaton is no more a sequence of states, but rather a tree of states, since the automaton can be in different states at a given time point. For a run to be accepting, each of its branches has to contain infinitely many occurrences of a final state. We refer to [196] for more details about Alternating Büchi automata.

The construction of the Büchi automaton follows the following steps: (i) from an LTL+P formula ϕ of size n, build an equivalent Alternating Büchi automaton of linear size ($\mathcal{O}(n)$); (ii) from the previous alternating automaton, use the Miyano-Hayashi construction [151] for building an equivalent Büchi automaton of exponential size in the worst case ($\mathcal{O}(2^n)$).

Using Generalized Büchi Automata

In [98, 181], the transformation from LTL to Büchi uses the so-called *Generalized Büchi automata*. A Generalized Büchi automaton is like a Büchi automaton, except from the fact that there is not a single set of final states, but many ones. A run is accepting if it visits infinitely many times at least one state for each set of final states of the automaton.

In order to check the emptiness of a Generalized Büchi automaton, there two options: (i) either using an emptiness checking algorithm dedicated for generalized automata [79]; or (ii) degeneralizing the automaton into a classical Büchi automaton [98].

5.1.3 The two-pass and graph-shaped tableau system

In this section, we recap the classical tableau system for LTL+P [141, 135, 200]. As already mentioned, this sistem is *graph-shaped* and *two-pass*, meaning that: (i) in a first pass, a graph structure representing all candidate models is built; (ii) in a second pass, the graph is traversed looking for *fulfilling paths*, *i.e.*, directed paths representing models that fulfills all the pending temporal requests, therefore representing correct models. The existence of such a fulfilling path witnesses the satisfiability of the starting formula, while the absence of them proves the unsatisfiability.

From now on, let $\phi \in \mathsf{LTL{+}P}$ be the starting LTL+P formula over an alphabet Σ, for which we want to check the satisfiability. Each node of the graph for ϕ built by this tableau system will be a set of formulas "derived" from ϕ. Therefore, we start by defining the notion of *closure* of an LTL+P formula, which comprises all the formulas derived from ϕ.

Definition 27 (Closure of an LTL+P formula)**.** *Let ψ be an* LTL+P *formula built over Σ. The* closure *of ψ is the smallest set of formulas $\mathcal{C}(\psi)$ satisfying the following properties:*

1. $\psi \in \mathcal{C}(\psi)$;
2. *for each sub-formula ψ' of ψ, $\psi' \in \mathcal{C}(\psi)$;*
3. *for each $p \in \Sigma$, $p \in \mathcal{C}(\psi)$ if and only if $\neg p \in \mathcal{C}(\psi)$;*
4. *if $\psi_1 \mathsf{U} \phi_2 \in \mathcal{C}(\psi)$, then $\mathsf{X}(\psi_1 \mathsf{U} \psi_2) \in \mathcal{C}(\psi)$;*

Rule	ψ	$\Gamma_1(\psi)$	$\Gamma_2(\psi)$
DISJUNCTION	$\alpha \vee \beta$	$\{\alpha\}$	$\{\beta\}$
CONJUNCTION	$\alpha \wedge \beta$	$\{\alpha, \beta\}$	
UNTIL	$\alpha \mathsf{U} \beta$	$\{\beta\}$	$\{\alpha, \mathsf{X}(\alpha \mathsf{U} \beta)\}$
SINCE	$\alpha \mathsf{S} \beta$	$\{\beta\}$	$\{\alpha, \mathsf{Y}(\alpha \mathsf{S} \beta)\}$
RELEASE	$\alpha \mathsf{R} \beta$	$\{\alpha, \beta\}$	$\{\beta, \mathsf{X}(\alpha \mathsf{R} \beta)\}$
TRIGGERED	$\alpha \mathsf{T} \beta$	$\{\alpha, \beta\}$	$\{\beta, \mathsf{Z}(\alpha \mathsf{T} \beta)\}$

Table 5.1: Tableau expansion rules.

5. *if* $\phi_1 \mathsf{R} \psi_2 \in \mathcal{C}(\psi)$, *then* $\mathsf{X}(\psi_1 \mathsf{R} \phi_2) \in \mathcal{C}(\psi)$;

6. *if* $\psi_1 \mathsf{S} \psi_2 \in \mathcal{C}(\psi)$, *then* $\mathsf{Y}(\psi_1 \mathsf{S} \psi_2) \in \mathcal{C}(\psi)$;

7. *if* $\psi_1 \mathsf{T} \psi_2 \in \mathcal{C}(\psi)$, *then* $\mathsf{Z}(\psi_1 \mathsf{T} \psi_2) \in \mathcal{C}(\psi)$.

Consider now a generic formula $\psi \in \mathcal{C}(\phi)$, and let σ be a state sequence. If ψ holds in σ at position i (*i.e.*, $\sigma, i \models \psi$), then, in the general case, it requires also other formulas of the closure to hold in state i. For example, if $\psi \equiv \psi_1 \wedge \psi_2$, then it also holds that $\sigma, i \models \psi_1$ and $\sigma, i \models \psi_2$. This is summarized in Table 5.1 by means of *expansion rules*. Given a formula $\psi \in \mathcal{C}(\phi)$, an expansion rule returns one or two sets of formulas in $\mathcal{C}(\phi)$ such that the following statement is true for each $\sigma \in (2^{\Sigma})^{\omega}$ and for each $i \in \mathbb{N}$:

$$\sigma, i \models \psi \;\Rightarrow\; \sigma, i \models \Gamma_1(\psi) \text{ and } \sigma, i \models \Gamma_2(\psi)$$

Nodes of the graphs are formalized using the notion of *atom*. Intuitively, an atom is a *maximal mutually satisfiable* set of formulas in the closure, meaning that all formulas in an atom *can* be true at some point i of a candidate model σ. It is also maximal in the sense that no formulas of the closure can be added to an atom while at the same maintaining its consistency. The definition below formalizes the notion of atom. In the following, we identify the formula $\neg\neg\psi$ with the formula ψ.

Definition 28 (Atom)**.** *Let* $\phi \in \mathsf{LTL{+}P}$ *and let* $\mathcal{C}(\phi)$ *be its closure. An* atom over ϕ *is a set* $S \subseteq \mathcal{C}(\phi)$ *satisfying the following requirements:*

- *the conjunction of all atomic formulas in* S *is satisfiable;*

- *for all* $\psi \in \mathcal{C}(\phi)$, $\psi \in S$ *if and only if* $\neg\psi \notin S$;
- *for all* $\psi \in \mathcal{C}(\phi)$, $\psi \in S$ *implies* $\Gamma_1(\psi) \in S$ *and* $\Gamma_2 \in S$.

Therefore, since atoms in the graph will correspond to states of a candidate model, if a formula ψ is included in an atom, then it means that ψ is supposed to hold in the state represented by that atom, while the absence of ψ from the atom means that $\neg\psi$ holds in the corresponding state.

We are now ready for defining the tableau for ϕ.

Definition 29 (Graph-shaped tableau)**.** *Let* $\phi \in \mathsf{LTL}$. *The tableau for* ϕ *is a graph* $T_\phi = (V, E)$ *such that:*

- V *is the set of all and only the* atoms over ϕ;
- $E \subseteq V \times V$ *is the set of edges and it such that, for each atoms* $A, B \in V$, A *is connected to* B *if and only if the following three requirements are satisfied:*
  - *for every* $\mathsf{X}\psi \in \mathcal{C}(\phi)$, $\mathsf{X}\psi \in A$ *iff* $\psi \in B$;
  - *for every* $\mathsf{Y}\psi \in \mathcal{C}(\phi)$, $\psi \in A$ *iff* $\mathsf{Y}\psi \in B$;
  - *for every* $\mathsf{Z}\psi \in \mathcal{C}(\phi)$, $\psi \in A$ *iff* $\mathsf{Z}\psi \in B$.

An atom A of the tableau T_ϕ is called *initial* if it does not contain any formula of type $\mathsf{Y}\psi$ or $\neg Z\psi$, for some $\psi \in \mathcal{C}(\psi)$.

We say that a formula $\psi \in \mathcal{C}(\psi)$ *promises* the formula α if ψ is of one of the following forms: $\beta\ \mathsf{U}\ \alpha$, $\mathsf{F}\alpha$, $\neg\mathsf{G}\neg\alpha$ or $\neg(\neg\alpha)\ \mathsf{R}\ (\beta)$. Equivalently, we say that ψ is a *promising formula*.

Promising formulas are the ones that may give rise of paths of the tableau T_ϕ corresponding to models such that, despite being correct with respect to Boolean formulas or formulas of type $\mathsf{X}\alpha$, $\mathsf{Y}\alpha$ or $\mathsf{Z}\alpha$, they always postpone the fulfillment of a request, therefore being not correct models. This means that, once the tableau T_ϕ is built, we have to check for at least one *fulfilling path*, that is a path that fulfills all temporal requests promised by some formula in the closure.

The following property is a fundamental property of promising formulas [141]. Let ψ be a formula promising α and let σ be a state sequence. Then it *always* holds that there exists *infinitely* many positions $i \in \mathbb{N}$ such that:

$$\sigma, i \models \neg\psi \text{ or } \sigma, i \models \alpha$$

The previous property gives also a definition for fulfilling paths. We say that a path $\pi = \langle A_0, A_1, \ldots \rangle$ in T_ϕ is *fulfilling* if and only if A_0 is an initial atom and, for every formula $\psi \in \mathcal{C}(\phi)$ promising α, there exists infinitely many positions $i \in \mathbb{N}$ such that either $\neg\psi \in A_i$ or $\alpha \in A_i$. The existence of a model for the starting formula ϕ can be reduced to the existence of a fulfilling path π in the tableau T_ϕ such that the first atom of π contains the formula ϕ. This last condition ensures that ϕ holds in the first state of the model σ represented by π, that is $\sigma \models \phi$, thus witnessing the satisfiability of ϕ.

Theorem 33 ([141])**.** *Let $\phi \in \mathsf{LTL}{+}\mathsf{P}$ and let T_ϕ be its tableau. The formula ϕ is satisfiable if and only if there exists a fulfilling path $\pi = \langle A_0, A_1, \ldots \rangle$ such that $\phi \in A_0$.*

In order to have an effective procedure for checking the existence of fulfilling path, it suffices to find a *strongly connected component* (SCC) within which all the paths are fulfilling. Since there are plenty of algorithms for computing SCCs, this proves the effectiveness of this algorithm.

5.1.4 The one-pass and tree-shaped tableau system

We now briefly describe the tableau system for $\mathsf{LTL}{+}\mathsf{P}$ introduced in [101], which extends the tableau system for LTL by Reynolds [163]. In the next chapters, we will give a symbolic encoding of this system.

W.l.o.g. we assume formulas to be in NNF. A tableau for a formula ϕ is a tree where each node u is labeled by a set of formulas $\Gamma(u)$ belonging to the closure $\mathcal{C}(\phi)$ (recall Definition 27), with the root u_0 labeled with $\Gamma(u_0) = \{\phi\}$. At each step, a set of rules is applied to a leaf (*i.e.*, a node without children/descendants), either for creating new nodes children of that leaf or for *closing* the branch ending to that leaf by either *accepting* or *rejecting* the branch. The produre stops when all branches have been closed. Given a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$, the sequence of nodes $\langle u_i, \ldots, u_j \rangle$, for some $0 \leq i \leq j \leq n$, is denoted by $\overline{u}_{[i,j]}$.

At each step, the selected leaf is subject to a number of *expansion rules*, that select a formula of its label and expand it according to its semantics, by means of *expansion rules* (recall Table 5.1). Each expansion rule creates one or two children depending on the selected formula. After repeated applications of the expansion rules, a node

that only contains *elementary* formulas, that is, propositions, *tomorrow*, *yesterday*, or *weak yesterday* formulas, is obtained: we call such nodes *poised nodes*. Elementary formulas of the form $\mathsf{X}(\phi_1 \mathsf{U} \phi_2)$ are called X-*eventualities*. An X-eventuality is a formula that, intuitively, requests something to be fulfilled later, and, in some sense, corresponds to promising formulas of the graph-shaped tableau that we previously described in Section 5.1.3. Given an X-eventuality $\phi \equiv \mathsf{X}(\phi_1 \mathsf{U} \phi_2)$, ϕ is said to be *fulfilled* in a node u if $\phi_2 \in \Gamma(u)$.

The tableau advances through time by making *temporal steps*. To do that, the following rules are applied to poised nodes.

STEP A child u_{n+1} is added to u_n, with:

$$\Gamma(u_{n+1}) = \{\alpha \mid \mathsf{X}\alpha \in \Gamma(u_n)\}$$

FORECAST Let

$$G_n = \left\{ \alpha \in \mathcal{C}(\phi) \,\middle|\, \begin{array}{l} \mathsf{Y}\alpha \in \mathcal{C}(\psi) \text{ or} \\ \mathsf{Z}\alpha \in \mathcal{C}(\psi) \text{ for some } \psi \in \Gamma(u_n) \end{array} \right\}$$

For each subset $G'_n \subseteq G_n$ (including $\varnothing$), a child u'_n is added to u_n such that $\Gamma(u'_n) = \Gamma(u_n) \cup G'_n$. This is done once and only once before every application of the STEP rule.

The STEP rule advances the construction of the current branch to the subsequent temporal state. The FORECAST is essential to the well-functioning of the rule dealing with *past*, as it adds a number of branches that *nondeterministically* guess formulas that may be needed to fulfill past requests coming from future states. We refer to [101] for more details.

Since the STEP rule is not applied to all the poised nodes (to some of which the FORECAST rule is applied instead), we need the following definition.

Definition 30 (Step node)**.** *In a complete tableau for an* LTL+P *formula, a poised node u_n is a* step node *if it is either a poised leaf or a poised node to which the STEP rule was applied.*

Given a node u, we define u^* as the closest ancestor of u that is child of a step node, if any. $\Gamma^*(u)$ is the union of the labels of the nodes from u to u^* or to the root, if u^* does not exist. Intuitively, $\Gamma^*(u)$ represents a state of a candidate model, and, in some sense,

it corresponds to atoms (Definition 28) of the two-pass and graph-shaped tableau that we described in Section 5.1.3.

Before applying the STEP rule though, poised nodes are subject to the application of a few *termination rules*, that is, rules that decide whether the construction has to continue or the current branch has to be either rejected or accepted. Given a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$, with u_n a step node, the termination rules are the following.

CONTRADICTION If $\{p, \neg p\} \subseteq \Gamma(u_n)$, for some $p \in \Sigma$, then $\overline{u}$ is *rejected*.

EMPTY If $\Gamma(u_n) = \varnothing$, then $\overline{u}$ is *accepted*.

YESTERDAY If $\mathsf{Y}\alpha \in \Gamma(u_n)$, then the branch $\overline{u}$ is *rejected* if either u_n^* does not exist or $Y_n \not\subseteq \Gamma^*(u_n^*)$, where $Y_n = \{\psi \mid \mathsf{Y}\psi \in \Gamma(u_n)\}$.

W-YESTERDAY If $\mathsf{Z}\alpha \in \Gamma(u_n)$, then $\overline{u}$ is *rejected* if u_n^* exists and $Z_n \not\subseteq \Gamma^*(u_n^*)$, where $Z_n = \{\psi \mid \mathsf{Z}\psi \in \Gamma(u_n)\}$.

LOOP If there exists a position $i < n$ such that $\Gamma(u_i) = \Gamma(u_n)$ and all the X-eventualities requested in u_i are fulfilled in $\overline{u}_{[i+1\ldots n]}$, then $\overline{u}$ is *accepted*.

PRUNE If there exist two positions i and j such that $i < j \leq n$, $\Gamma(u_i) = \Gamma(u_j) = \Gamma(u_n)$, and all the X-eventualities requested in these nodes which are fulfilled in $\overline{u}_{[j+1\ldots n]}$ are also fulfilled in $\overline{u}_{[i+1\ldots j]}$, then $\overline{u}$ is *rejected*.

Intuitively, the CONTRADICTION, YESTERDAY, and W-YESTERDAY rules reject branches that contain some contradiction, either a propositional one or because of some unfulfilled past request. The EMPTY rule accepts a branch devoid of contradictions where there is nothing left to do, while the LOOP one accepts a looping branch where all the X-eventualities are proposed again and fulfilled at every repetition of the loop. Finally, the PRUNE rule, which was the main novelty of the system when introduced by Reynolds [163], rejects a branch that is doing redundant work without fulfulling all the X-eventualities.

The following has been proved to hold.

Proposition 11 (Soundness and completeness [101])**.** *Let ϕ be an* LTL+P *formula. The complete tableau for ϕ contains an accepted branch if and only if ϕ is satisfiable.*

Fig. 5.1 shows two examples tableaux built with this systems for the formulas $\mathsf{GF}(p \wedge \mathsf{X}\neg p)$ and $\mathsf{G}\neg p \wedge q\,\mathsf{U}\,p$ (the figure has been taken from [101]).

5.2 Model checking

Model checking (MC) is a technique for automatically verifying systems against a specification [58, 59, 161, 60]. Model checking has three distinguish features with respect to previous methods for the verification of digital systems, that helped to make it very successful and popular in practice: (i) it is fully *automatic*; (ii) it is *exhaustive*, meaning that *all* computations of the systems are explored and checked against the specification; (iii) in the case the specification does not hold in the system, a *counterexample* of the violation is generated.

The typical methodology for formal verification using model checking is the following:

1. model the system under investigation with a state-transition system; in the simplest case, the systems have finitely many states and are modeled by *Kripke structures* [60];

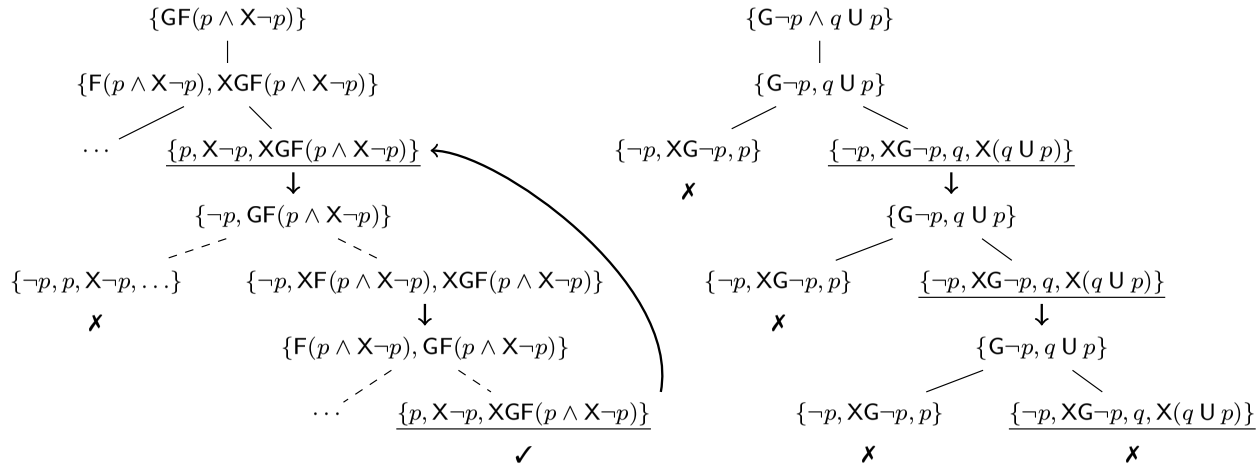
2. formalize the specification that the system must satisfy by means of a formal language; *temporal logics* (recall Section 2.4) typically represent a great tool for this task;

3. use an algorithm (*i.e.*, a fully automatic procedure) to check the (temporal) formula over *all* the paths of the (Kripke) structure. If a counterexample has been found, the algorithm returns a *negative* answer together with the counterexample, which is of the form of a path in the structure. Otherwise, it returns a *positive* result.

In this section, we consider state-transition systems with finitely many states, which usually are denoted with the name of *Kripke structures*.

Definition 31 (Kripke structure)**.** *A Kripke structure is a tuple* $M = (\Sigma, Q, I, T, L)$ *where:*

- Σ *is a finite alphabet,*



(a) Tableau for $\mathsf{GF}(p \wedge \mathsf{X}!p)$, closed by the LOOP rule.

(b) Tableau for $\mathsf{G}\neg p \wedge q \,\mathsf{U}\, p$, closed by the PRUNE rule.

Figure 5.1: Example tableaux for two formulae, involving the LOOP and PRUNE rules. Dashed edges represent subtrees collapsed to save space, bold arrows represent the application of a STEP rule to a poised label.

- Q *is the finite set of states,*
- $I \subseteq Q$ *is the set of initial states,*
- $T \subseteq Q \times Q$ *is a* complete *transition relation, and*
- $L : Q \to 2^{\Sigma}$ *is the labeling function that assigns to each state the set of atoms in* Σ *that are true in that state.*

Let $M = (\Sigma, Q, I, T, L)$ be a Kripke structure. A *trace* $\pi := \langle \pi_0, \pi_1, \ldots \rangle$ is a (finite or infinite) sequence of states of M such that $(\pi_i, \pi_{i+1}) \in T$, for all $i \geq 0$. If $\pi_0 \in I$, we say that π is an *initialized trace.*

The model checking problem takes as input a Kripke structure and a temporal formula, and asks to find whether *all* the initialized traces of the former satisfy the latter. For this reason, model checking is also referred to as *validity over structures.*

Definition 32 (The model checking problem)**.** *Given a Kripke structure* $M = (\Sigma, Q, I, T, L)$ *and a temporal formula* ϕ*, the model checking problem is the problem of finding whether* all *the initialized traces* π *of* M *are such that* $L(\pi) \models \phi$*, written* $M \models \mathsf{A}\phi$ *(where* A *is the* "for all paths" *operator of* CTL*).*

Here we are mainly interested on model checking from LTL specifications (from now on called LTL *model checking*), that is model checking of specification written in Linear Temporal Logic (Section 2.4). In [179, 135], the complexity of LTL model checking is proved to be PSPACE-complete.

Theorem 34 (Complexity of LTL model checking [179, 135])**.** *The model checking problem of* LTL *(and* LTL+P*) is* PSPACE*-complete.*

5.2.1 Symbolic Kripke Structures

The first algorithms for model checking were basically graph algorithms [58, 161]: since Kripke structures correspond to graphs with labels on states, those algorithms iteratively compute the set of states on which the starting formula holds, until they either reach a fixpoint or they find a counterexample.

The representation of systems by Kripke structures is usually referred to as the *explicit-state* representation, meaning that each state of the structure corresponds to a location on the memory of

the computer, and each transition corresponds to a pointer. Explicit-state representations revealed to be inefficient when model checking is applied to real-life systems. In fact, it is very common to have a system whose corresponding Kripke structure has too much states to be stored on memory: by considering a system made of several subsystems, the states of the former are the cartesian product between the set of states of each subsystem, thus taking into consideration every possible combination of states. In the general case, a system made of n subsystems can contain in the worst case $\mathcal{O}(2^n)$ states. This is known as the *state-space explosion problem*. Therefore the problem is how to succinctly represent huge graph structures on memory and, consequently, how to verify temporal formulas over this succinct graph representation.

A breakthrough was reached in 1993 with the introduction of *Symbolic Model Checking* [37, 146], which, instead of representing Kripke structures explicitly (with each state and transition stored on memory), it uses *Boolean formulas* for representing *sets of states* (actually, sets of edges). In particular, states and edges are *never* stored explicitly. In addition, the verification of temporal formulas over such graphs (which amounts to traversing the graph structure looking for the fulfillment of some conditions) is carried out by manipulating those Boolean formulas.

Let $M = (\Sigma, Q, I, T, L)$ be a Kripke structure with $n = |Q|$ states. The *symbolic representation* of M uses a set of $\log_2(n)$ Boolean variables $V = \{v_1, \ldots, v_{\log_2(n)}\}$ for representing the states in Q: each of the n assignments to the $\log_2(n)$ variables represents a state in Q. Given a state $q \in Q$, we denote with $\overline{q}$ the corresponding assignment to the variables in V.

A Boolean formula is used for representing the edges of the graph, that is the transition relation of M. In particular, we define $V' = \{v'_1, \ldots, v'_{\log_2(n)}\}$ as the set containing all variables in V but *primed* in order to represent the *next* state of a transition. The transition relation T of M is represented symbolically by a Boolean formula over the variables $V \cup V'$, whose models correspond to all and only the transitions in T.

Finally, the labeling function L is represented by a set of Boolean formulas ϕ_σ over V, one for each $\sigma \in \Sigma$, such that an assignment is a model of ϕ_σ if and only if the state encoded by that assignment is labeled by σ by L in M.

Definition 33 (Symbolic Kripke structure)**.** *Let $M = (\Sigma, Q, I, T, L)$ be a Kripke structure. The* symbolic Kripke structure *corresponding to M is a tuple $M_s = (\Sigma, V, \phi_I, \phi_T, \{\phi_\sigma\}_{\sigma\in\Sigma})$ such that:*

- *V is a set of $\log_2(|Q|)$ Boolean variables;*
- *ϕ_I is a Boolean formula over V for the set of initial states such that, for each state $q \in Q$*

$$\overline{q} \models \phi_I \Leftrightarrow q \in I$$

- *ϕ_T is a Boolean formula over $V \cup V'$ for the transition relation such that, for each $(q, q') \in T$*

$$(\overline{q}, \overline{q'}) \models \phi_T \Leftrightarrow (q, q') \in T$$

- *for each $\sigma \in \Sigma$, the Boolean formula ϕ_σ is such that, for each $q \in Q$, it holds that:*

$$\overline{q} \models \phi_\sigma \Leftrightarrow \sigma \in L(q)$$

In the following subsections, we will see how the symbolic representation can be used also for modeling automata (see Section 2.2) and we will take a look at the SMV language [147], a high-level language for modeling symbolic state-transition systems. Moreover, the symbolic algorithms of Bounded Model Checking [17] and K-Liveness [57] will be described. We conclude the section, by showing how LTL satisfiability can be reduced to LTL model checking.

5.2.2 Symbolic Automata

The symbolic representation described before can be used not only for modeling Kripke structures and state-transition systems, but also for modeling automata (recall Section 2.2). Since automata are graphs with labels on the edge while Kripke structures are graphs with labels on the states, the symbolic representation of an automaton (from now on simply called *symbolic automaton*) follows from that of a Kripke structure with only minor modifications. In this part of the thesis, we will focus mainly on infinite sequences, and therefore we give the definition of symbolic automata for automata over infinite words only.

Definition 34 (Symbolic Automata on Infinite Words)**.** *A* symbolic automaton on infinite words *over the alphabet* Σ *is a tuple* $\mathcal{A} = (V = X \cup \Sigma, I, T, \alpha)$*, such that*

- $V = X \cup \Sigma$*, where* X *is a set of* state variables *and* Σ *is a set of* input variables*,*
- $I(X)$ *and* $T(X, \Sigma, X')$*, with* $X' = \{x' \mid x \in X\}$*, are Boolean formulas which define the set of initial states and the transition relation, respectively, and*
- $\alpha(X)$ *is an* LTL *formula which defines the accepting condition.*

The definitions of runs, accepted words and languages of a symbolic automaton correspond naturally to those of standard (explicit state) automata.

Definition 35 (Languages of Symbolic Automata)**.** *Let* $\mathcal{A} = (V = X \cup \Sigma, I, T, \alpha)$ *be a symbolic automaton over infinite words. A* run *(or* trace*)* $\tau = \langle \tau_0, \tau_1, \ldots \rangle$ *is an infinite sequence of* state-symbol *pairs (*i.e.*, evaluations of the variables in* V*) that are in relation with respect to* T*,* i.e.*, such that any two consecutive evaluations satisfy the formula* T *(*$\overline{\tau_i}, \overline{\tau_{i+1}}' \models T$*).*

A run τ *is* induced *by the word* $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ *iff* $\tau_0 \models I$ *and* $(\tau_i, \sigma_i, \tau_{i+1}) \models T$*, for all* $i \geq 0$*. A word* σ *is* accepted *by* $\mathcal{A}$ *iff there exists an accepting run induced by* σ *in* $\mathcal{A}$*. The* language *of* $\mathcal{A}$*, denoted with* $\mathcal{L}(\mathcal{A})$*, is the set of all and only the words accepted by* $\mathcal{A}$*.*

Let us consider now the product between two symbolic automata. In the symbolic setting, the product is much simpler than in the explicit-state setting (Definition 9), since it amounts to consider the *conjunctions* between the two formulas for the initial states, the transition relations, and the accepting conditions.

Definition 36 (Product between symbolic automata)**.** *Let* $\mathcal{A} = (V = X \cup \Sigma, I, T, \alpha)$ *and* $\mathcal{A}' = (V' = X' \cup \Sigma, I', T', \alpha')$ *be two symbolic automata. The product* $\mathcal{A} \times \mathcal{A}'$ *between* $\mathcal{A}$ *and* $\mathcal{A}'$ *is defined as the automaton* $(V \cup V', I \wedge I', T \wedge T', \alpha \wedge \alpha')$*.*

When the accepting condition $\alpha(X)$ of Definition 34 is istantiated to a particular LTL formula, it gives rise to the symbolic representation of different types of automata, for example *safety automata* (Definition 13) or *Büchi automata* (Definition 11).

Definition 37 (Symbolic Safety Automata)**.** *A* symbolic safety automaton *(SSA) is a symbolic automaton on infinite words* $\mathcal{A} = (V = X \cup \Sigma, I, T, \alpha)$ *such that*

$$\alpha \coloneqq \mathsf{G}\psi$$

where G *is the* globally *operator of* LTL *and* ψ *is a Boolean formula over the variables in* X*.*

It is worth noting the differences between the explicit-state standard definition of a safety automaton (Definition 13) and the symbolic representation (Definition 37). The former requires each accepted word to induce at least a run that has only to satisfy the transition relation δ, while the latter requires the run to satisfy also the formula ψ for all of its states (in other words, it imposes also the accepting condition $\alpha(X)$ on the runs). Nevertheless it is not difficult to see that the two definitions are equivalent, in the sense that the set of languages that they can recognize is the same:

- a safety automaton corresponds to a symbolic safety automaton in which the accepting condition is $\alpha(X) \coloneqq \mathsf{G}\top$;
- a symbolic safety automaton corresponds to a safety automaton in which all states that does not satisfy ψ are removed.

The reason why the definition of symbolic safety automata imposes the additional constraint $\alpha(X)$ is that it is usually simpler to first construct symbolically the transition relation of an automaton and then give a pruning condition for the bad states by means of the accepting condition $\alpha(X)$.

By istantiating the accepting condition, we can similarly define also *symbolic Büchi automata*.

Definition 38 (Symbolic Büchi Automata)**.** *A* symbolic Büchi automaton *(SBA) is a symbolic automaton on infinite words* $\mathcal{A} = (V = X \cup \Sigma, I, T, \alpha)$ *such that*

$$\alpha \coloneqq \mathsf{GF}\psi$$

where G *and* F *are the* globally *and the* eventually *operators of* LTL *and* ψ *is a Boolean formula over the variables in* X*.*

The correspondence between standard explicit-state Büchi automata (Definition 11) and symbolic Büchi automata (Definition 38)

is tight: a Büchi automaton with final states F corresponds to a symbolic Büchi automaton with accepting condition $\mathsf{GF}\psi$, where ψ is the Boolean formula representing exactly the states in F.

Besides from the *safety* and *Büchi* acceptance conditions, one can use also more sophisticated formulas for $\alpha(X)$. In the following chapters, we will make use of the Reactivity(1) (Definition 17) and the Generalized Reactivity(1) (Definition 18) acceptance conditions (defined here below), which take their name from the classes of the Temporal Hierarchy (Section 2.4.5).

Definition 39 (The $\mathsf{R}(1)$ and $\mathsf{GR}(1)$ accepting conditions)**.** *Let $\mathcal{A} = (V = X \cup \Sigma, I, T, \alpha)$ be a symbolic automaton. The formula $\alpha(X)$ is called a:*

- Reactivity(1) *accepting condition (*$\mathsf{R}(1)$*, for short) iff it is of type $\mathsf{GF}\alpha \rightarrow \mathsf{GF}\beta$;*
- Generalized Reactivity(1) *accepting condition (*$\mathsf{GR}(1)$*, for short) iff it is of type $\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$;*

where each α, α_i, β, β_j belongs to $\mathsf{LTL}{+}\mathsf{P}_{\mathsf{P}}$.

In what follows, and in particular in the parts dedicated to reactive synthesis from $\mathsf{LTL}_{\mathsf{EBR}}{+}\mathsf{P}$ and GR-EBR specifications (Chapters 7 and 8), we will make a strong use of *deterministic symbolic automata*. We define here the symbolic representation of deterministic automata.

Definition 40 (Deterministic Symbolic Automata)**.** *We say that a symbolic automaton on infinite words $\mathcal{A} = (V = X \cup \Sigma, I, T, \alpha)$ is* deterministic *if:*

1. *the formula I has exactly one satisfying assignment, and*
2. *the transition relation is of the form:*

$$T(X, \Sigma, X') \coloneqq \bigwedge_{x \in X} (x' \leftrightarrow \beta_x(X \cup \Sigma)),$$

where $\beta_x(V = X \cup \Sigma)$ is a Boolean formula over V, for each $x \in X$.

It is worth noting how the $\leftrightarrow$ operator (in the second point of Definition 40) makes the value of any variable x be a *function* of the assignments of $\beta_x(X \cup \Sigma)$, just like the transition relation of the classical definition of deterministic automaton is a *function* of the current state (X) and an input letter (Σ).

5.2.3 The SMV modeling language

The SMV (Symbolic Model Verifier) language [147, 38] is a modeling language for both finite and infinite state-transitions symbolic systems, and consequently also symbolic automata. It offers high-level constructs for expressing (i) Boolean formulas; (ii) initial states; (iii) transition relations; (iv) final states, and so on and so forth.

The basic building blocks of an SMV file are *modules* (expressible with the MODULE keyword): each module can be thought of as a class in the object oriented paradigm. In any SMV file, there must be exactly one module named main.

Each module has to contain the declaration of the *state variables*, that is the variables in the set V of Definition 33. This is done with VAR keyword followed by the list of declarations of all the variables, that are of the form name_variable : type followed by a semicolon. There are several possible types for a variable: boolean, integer, real, array, bitvectore *etc.* It is important to note that using integer or real types brings inevitably to infinite states systems.

Additionally, a module can contain also *input variables*, that are meant to model labels on the edges (*i.e.*, the variables in the set Σ of Definition 34). This is done with the IVAR keyword followed by a list of declarations of the form name_variable : type;.

The transition relation T of Definition 33 is defined by the TRANS keyword followed by a Boolean formula, built using the state and input variables as the atomic propositions and the classical symbols for the Boolean connectives, like & for the conjunction, | for the disjuction, ! for the negation, and so on and so forth.

The LTLSPEC keyword is used for specifying the LTL specification to check over the model. This is particularly interesting also for modeling symbolic automata: the LTL specification is exactly the accepting condition $\alpha(X)$ of Definition 34.

So far we have seen how to model each component of the definition of a symbolic automaton (Definition 34) and of a symbolic Kripke structure (Definition 33) except from the functions for labeling each state of a Kripke structure with a particular label $\sigma \in \Sigma$. The SMV language does not have constructs for modeling these functions. In fact, the labels on states, differently from the labels on the edges of automata (see IVAR keyword), are not directly specifiable in SMV. Instead, any LTL property that wants to use a label σ as a propositional atom can simply use the corresponding Boolean

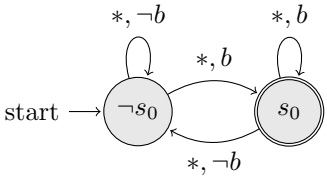


Figure 5.2: Büchi automaton for the language of infinite words over $\Sigma = 2^{\{a,b\}}$ containing infinitely many b. The $*$ symbol denotes any value for the 'a' letter, that is either a or $\neg a$.

formula ϕ_σ (see Definition 33) instead of such an atom.

Let us now present an example of how to model a symbolic automaton with the SMV language. Consider the Büchi automaton $\mathcal{A}$ over the alphabet $\Sigma = 2^{\{a,b\}}$ depicted in Fig. 5.2 accepting the words containing infinitely many b:

$$\mathcal{L}(\mathcal{A}) = \{\sigma \in (2^{\{a,b\}})^\omega \mid \exists^\omega i \,.\, b \in \sigma_i\}$$

The SMV file for the symbolic representation of $\mathcal{A}$ is the following:

```
MODULE main
IVAR
  a  : boolean;
  b  : boolean;
VAR
  s0 : boolean;
INIT
  !s0;
TRANS
  (!s0  &  !b  &  !s0')  |
  (!s0  &  b   &  s0')   |
  (s0   &  !b  &  !s0')  |
  (s0   &  b   &  s0');
LTLSPEC
  GF(s0);
```

Functional SMV

The SMV language provides also a set of keywords for specifying *deterministic symbolic automata* (recall Definition 40) or input-

deterministic state-transition systems[1]. We refer to this language as *Functional SMV*. Instead of using the INIT and TRANS keyword, SMV provides the ASSIGN keyword along with the init(·) and next(·) functions. In particular, after the ASSIGN keyword, for each state variables there is a pair of statements of this form:

```
init(state_var) := Boolean formula
                    over the state variables;
next(state_var) := Boolean formula
                    over the state variables;
```

The semantics of the previous functions is intuitive: $\mathsf{init}(\mathsf{v}) := \phi_I$ specifies the fact that the *initial value* for the state variable v is the evaluation (true or false) of the formula ϕ_I; similarly, $\mathsf{next}(\mathsf{v}) := \phi_\mathsf{T}$ specifies that the *next value* for v is the evaluation of the Boolean formula ϕ_T.

We now give an example of an SMV file using the ASSIGN statement. Consider the automaton depicted in Fig. 5.2. The automaton is (input-)deterministic. The SMV for modeling the deterministic symbolic automaton (Definition 40) is the following:

```
MODULE main
IVAR
  a  : boolean;
  b  : boolean;
VAR
  s0 : boolean;
ASSIGN
  init(s0) := FALSE;
  next(s0) := case
    !s0  &  !b  :  FALSE,
    !s0  &  b   :  TRUE,
    s0   &  !b  :  FALSE,
    s0   &  b   :  TRUE;
  esac;
LTLSPEC
  GF(s0);
```

[1]In the context of state-transition systems with labels on the edges rather than automata, the type of determinism in which we are interested (that is for each state and for each letter there is a unique successor state) is called *input-determinism*.

5.2.4 Binary Decision Diagrams, SAT and SMT

In Section 5.2.1, we have seen how to succinctly represent a state-transtion system (equiv. a Kripke structure) by means of Boolean formulas. Several algorithms for model checking that manipulate Boolean formulas instead of traversing the explicit representation of the graph have been proposed in the literature: they fall under the name of *symbolic model checking*.

Binary Decision Diagrams

The first algorithm for symbolic model checking was introduced in 1992 by McMillan *et al.* [37, 146]. We will not go into the details of this algorithm, since we will not make use of it throughout this thesis. Nevertheless, it is important to recall this groundbreaking result since, besides having paved the way for symbolic model checking, its limitations have inspired the introduction of other symbolic techniques (that we will use in this thesis) that try to overcome those limitations.

The technique proposed by McMillan *et al.* represents the Boolean formulas orginated from the symbolic representation of Kripke structures (Definition 33) by means of Binary Decision Diagrams (BDDs) [33, 3]. BDDs offer a canonical form for representing the set of all and only the models of a Boolean formula in a succinct way. They are basically binary trees with labels belonging to the alphabet $\{0, 1\}$ on the edges and on the leaves. The labels on the edges at a certain height corresponds to assignments to a given variable of the original Boolean formula: in this way, the path from the root to a given leaf corresponds to an assignment to the variables of the formula built using the labels on the edges in the path. If this leaf is labeled with 1, then it means that the assignment is a model of the formula, otherwise it means that it falsifies it.

BDDs are built in a bottom-up fashion, starting from the propositional atoms and providing operations for conjunctions and negations (and also for the quantifiers $\forall$ and $\exists$). In addition, there is a set of operators that is carefully designed in order to maintain the BDD as small as possible: this is at the core of the succinctness of BDDs. Roughly speaking, those operations exploits some *redundancies* between the models of a formula, specially when it comes to *don't care* variables (*i.e.*, variables whose assignment is irrelevant for the satisfaction of the formula).

It is possible to prove that BDDs are *canonical*, in the sense that, once fixed the order of the variables, there's a unique minimal BDD that can be built using the above-mentioned operations. In the average case, BDDs are a succinct representation for the set of models of a formula. Nevertheless, it is possible to prove that, for some orderings of the variables, the BDDs is exponential in the size of the formula. In addition, checking the optimal ordering (*i.e.*, the ordering for which the BDDs is the smallest one) is NP-complete [33]. This limitation brought to the introduction of other techniques. Two of them are Bounded Model Checking and K-Liveness, that we will see in the next part of this section.

The SAT problem and SAT-solvers

Instead of representing succinctly the set of models of a Boolean formula and then look for the existence of one of them, the alternative is to check directly for the satisfiability of the formula. The problem of checking the satisfiability of a Boolean formula is called the SAT problem. It was the first problem to be proved being NP-complete [61, 121].

A SAT-solver is an algorithm for solving the SAT problem. The DPLL algorithm (from the name of its inventors, Davis, Putnam, Logemann, Loveland [66, 65]) was one of the first algorithms to be proposed for the SAT problem, but still, even more than 80 years later, it stands at the basis of modern SAT-solvers.

A successful set of heuristics for SAT is CDDL (Conflict-Driven Clause Learning) for SAT. It refers to a set of techniques that aim at speed up the search of a model by learning from unsatisfiable assignments discovered so far. By pairing the DPPL and the CDCL algorithms together also to other clever heuristics, nowadays SAT-solvers are very efficient and optimized and can solve instances with millions of variables.

Finally, the success of modern SAT-solvers is also due to their *incremental interface*. This interface offers two methods, `push` and `pop`, for adding/removing a new subformula to/from the formula already stored in the SAT-solver, respectively. The goal is to recycle some work for the satisfiability of the new formula (the one obtained after the `push` or `pop`) from the work done for the satisfiability of the previous formula. This is achieved by implementing efficiently the `push` and `pop` operations, in such a way that the SAT-solver does

not recompute all its internal structures when these two operations are executed.

The SMT problem and SMT-solvers

The use of Boolean formulas for specifying systems (Definition 33) restricts the modeled system to be finite-state (*i.e.*, a Kripke structure), since the number of models of any Boolean formula (that in the symbolc representation are meant to represent states) is always finite.

Nevertheless, finite-state systems are not always enough for representing real-life designs. For example, a *timed system*, that is a system in which the real-time component plays a central role, is typically made of: (i) a finite-state discrete component; (ii) an infinite-state real-time component. Sometimes, the use of infinite state systems can be a choice guided by efficiency motivations. For example, a system can be finite-state but its Booleanization introduces to many variables that it is more convinient to consider it as really infinite-state.

As finite-state systems can be modeled by Boolean formulas (Definition 33), infinite-state systems can be modeled by Boolean formulas whose atoms are not necessary Boolean variables but can be arbitrary atomic formulas of a fixed theory. The problem of establishing the satisfiability of these formulas is called *Satisfiability Modulo Theory* (SMT). Formally, an SMT instance is a formula of first-order logic with a fixed theory, that is, with a fixed interpretation of some symbols. Therefore, the formula can feature existential and universal quantifiers ($\exists$ and $\forall$), Boolean connectives ($\vee$, $\neg$, ...), variables (x, y, ...), and atomic propositions belonging to the fixed theory ($x + 5 \leq y$, ...)[2].

If $\mathcal{T}$ is a theory, then with SMT($\mathcal{T}$) we refer to the set of all SMT instances where the theory $\mathcal{T}$ is fixed. Typically, the theories used in SMT are decidable fragments of first-order logic. For example, a non-comprehensive list of famous theories is the following:

- *Linear Real Arithmetic* (LRA), where atomic propositions are constrained to be linear formulas (like $x+y \leq 10$) and variables

[2]It is worth noticing that also in Boolean logic we have quantifiers, although usually they are not explicitly considered because they can be removed very easily by the Shannon expansion. Therefore, SMT instances are really a generalization of Boolean formulas where atoms are no more contrained to be Boolean.

are interpreted over the domain $\mathbb{R}$ of real numbers;

- *Linear Integer Arithmetic* (LIA): like above, but the domain of interpretation is the set $\mathbb{N}$ of natural numbers;
- *Difference Logic* (DL), where atomic propositions are constrained to be of type $x + y \leq c$, for some constant c, and the domain of interpretation is $\mathbb{R}$;
- *Bit-vectors* (BV): atoms are bit-vectors, that is, fixed-size strings of bits, and they are built using classical bitwise operators.

SMT-solvers are the set of techniques for deciding the satisfiability of an SMT-instance (*e.g.*, MathSAT [54], Z3 [76]). They combine efficient decision procedures developed for SAT-solvers (for tackling with the Boolean structure of the formula) with techniques for deciding the truth of atomic formulas at the theory level. This combination is tight: conflicts that are detected at theory-level can help pruning the state-space for the Boolean part.

5.2.5 Bounded Model Checking

Bounded Model Checking (BMC) is a symbolic model checking algorithm introduced in 1999 [16, 17]. The main idea behind BMC is the following. Say that we want to solve the LTL model checking problem $M \models \mathsf{A}\,\phi$, for a given symbolic Kripke structure M and an LTL formula ϕ. BMC looks for a path in M that falsifies ϕ; equivalently, it solves the following problem: $M \models E\neg\phi$, where E is the *exists a path* operator of CTL. Crucially, it encodes this problem into a sequence of SAT problems. Let us show how. The main cycle of BMC is the following:

1. let $n \in \mathbb{N}$ be a natural number for the length of a candidate model of $\neg\phi$; initially, $n = 0$;
2. BMC checks whether there exists at least a path of M of length n fulfilling $\neg\phi$; it solves this problem by encoding it into a SAT instance (Section 5.2.4); for this point, BMC exploits the fact that a *finite* path can still represent an infinite one if it contains a *loop-back*, that is a transition from the last state of the path to some previous state (see Fig. 5.3); in this way, it can discover all possible paths of M that falsifies ϕ, that can be either infinite or finite in the general case;

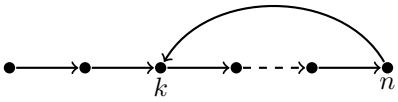


Figure 5.3: A finite path representing an infinite one.

3. it calls a SAT solver (Section 5.2.4) for solving the problem; if it returns a positive result, than this means that there is a path (of length n) in M that is a model of $\neg\phi$, and so BMC returns a *negative* result along with the path (counterexample); otherwise, it increments n, and repeats the cycle.

Let us show how BMC encodes the existence of a counterexample of length n of ϕ in M into a Boolean formula. This counterexample has two features:

1. it is a path of length n of M;
2. it is a model of $\neg\phi$.

Let us see how to encode the first feature. Let $M = (\Sigma, V, I, T, \{\phi_\sigma\}_{\sigma\in\Sigma})$ be a symbolic Kripke structure (Definition 33), where $I(V)$ and $T(V, V')$ are Boolean formulas for the set of initial states and the transition relation, respectively. The *unfolding of* T is the following Boolean formula, whose models represent all and only the paths of M of length n:

$$\langle\!\langle M \rangle\!\rangle_n \coloneqq I(V^0) \wedge \bigwedge_{i=0}^{n-1} T(V_i, V_{i+1})$$

where V^i is the set of variables obtained from V by adding the index i for specifying that they are meant to represent the i^{th} state of a path, and $V^0 = V$.

As for the second point, the satisfaction of $\neg\phi$ by the path (of length n) is encoded by BMC by distiguishing two cases. If there are loop-backs (in this case the path is called *lasso-shaped*), than the finite path represents actually an infinite path in M: for example, the finite path in Fig. 5.3 represents the infinite path that coincides with the finite one in the first n steps and then repeats infinitely often the suffix from k to n. In this case, standard LTL semantics

(Section 2.4) are applied. Conversely, if there are no loop-backs (in this case the path is called *loop-free*), then it means that the path is really finite in M: even if it is finite, the path can still be a violation of ϕ (for example, for all $\phi \in [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$). Take for example $\phi \coloneqq \mathsf{G}p$ and the path containing two states and no loop-backs, such that in the first state p holds while in the second $\neg p$ holds: the finite path is a model of $\neg\phi$. Therefore, in this case, the *finite semantics* of LTL are applied.

The formula L_n for detecting a loop-back from the last state n to one of the previous states is simple:

$$
\begin{aligned}
{}_kL_n &\coloneqq T(V^n, V^k) \\
L_n &\coloneqq \bigvee_{k=0}^{n} {}_kL_n
\end{aligned}
$$

The satisfaction of the formula $\neg\phi$ by the path of length n is encoded by: (i) distinguishing the case with or without loop-backs; and (ii) accordingly, expanding the formula with classical expansion rules (Table 5.1) in order to replicate the semantics along the path. We call $\langle\!\langle \phi \rangle\!\rangle_n^0$ the Boolean formula for the loop-free case of paths of length n, while ${}_k\langle\!\langle \phi \rangle\!\rangle_n^0$ the Boolean formula for the case of a lasso-shaped model with a loop-back from state n to state k (see Fig. 5.3). The final formula for iteration n, encoding all paths of length n in M that violates ϕ, is the following:

$$
\langle\!\langle M, \neg\phi \rangle\!\rangle_n \coloneqq \underbrace{\langle\!\langle M \rangle\!\rangle_n}_{\substack{\text{encoding} \\ \text{of the Kripke} \\ \text{structure}}} \wedge \Bigg(\underbrace{(\neg L_n \wedge \langle\!\langle \neg\phi \rangle\!\rangle_n^0)}_{\substack{\text{loop-free} \\ \text{models}}} \vee \underbrace{\bigvee_{k=0}^{n} ({}_kL_n \wedge {}_k\langle\!\langle \neg\phi \rangle\!\rangle_n^0)}_{\substack{\text{lasso-shaped} \\ \text{models}}} \Bigg) \tag{5.1}
$$

Exploiting Incrementality

Between one iteration and the next one, it is clear that only some parts of Eq. (5.1) change. In particular, the encoding $\langle\!\langle M \rangle\!\rangle_n$ of the paths of length n in M is always the same. Also some parts of the encodings $\langle\!\langle \neg\phi \rangle\!\rangle_n^0$ and ${}_k\langle\!\langle \neg\phi \rangle\!\rangle_n^0$ of the formula $\neg\phi$ can be saved between one iteration and the other.

This can be exploit by using the *incrementality* of modern SAT-solvers (Section 5.2.4), in particular by using the push/pop interface.

For example, at the beginning of iteration n, we can pop the parts of the Boolean formula that has to be removed from iteration $n-1$, and push the new parts. In this way, once we call the SAT-solver on the formula for the iteration n, it does not have to recompute all the internal structures but, instead, it can save some work from the previous iteration.

Completeness

Consider the case in which the property ϕ holds in all paths of M ($M \models \mathsf{A}\,\phi$). In this case, the algorithm of BMC described above does not terminate, since there are no counterexamples of ϕ of any length.

In order to guarantee termination, several techniques have been proposed for proving the exploration of *all paths*. For example, if we can prove that n is greater than the length of the longest loop-free path (the so-called *recurrence diameter* [17]), then we can stop incrementing n and terminate with a positive result. Nevertheless, the computation of the recurrence diamater in the particular case, and of other thresholds in the general case, is hard. For example, for the recurrence diamater, it requires to solve the satisfiability of a quantified Boolean formulas. For this reason, BMC is usually used as a *bug finder* (*i.e.*, for finding counterexamples to the property), rather than as a *prover* (*i.e.*, for proving the validity of ϕ over M).

5.2.6 K-Liveness

K-Liveness is a quite recent algorithm for symbolic LTL model checking [57]. Like BMC reduces the problem into a sequence of SAT problems and exploits efficient SAT-solvers for solving each of them, K-Liveness reduces the problem into a sequence of model checking problem of *safety properties* (Section 2.1.3) and exploits efficient model checkers for those properties. The gain is that a safety model checking problem is typically simpler than an LTL model checking problem. Moreover, nowadays, very efficient model checking algoritms for safety properties have been introduced: one among all is IC3 [28, 29], which proved to bring an extreme gain in performance againts existing algorithms.

The K-Liveness algorithm for the problem $M \models \mathsf{A}\,\phi$ consists in the following steps.

1. Build the Büchi automaton $\mathcal{A}(\neg\phi)$ for the LTL property $\neg\phi$; this can be done using any construction algorithm, for example the ones descibed in Section 5.1.2; K-Liveness represents this automaton symbolically (Definition 34); let α be the Boolean formula for the set of final states of this automaton.

2. Build the *product automaton* $M \times \mathcal{A}(\neg\phi)$ between M and $\mathcal{A}(\neg\phi)$ (Definition 36)[3]; this automaton accepts all and only the words that are (i) computations of M; and (ii) are models of $\neg\phi$; it holds that:

   $M \models \mathsf{A}\,\phi$
   $\mathcal{L}(M \times \mathcal{A}(\neg\phi)) = \varnothing$
   each path of $M \times \mathcal{A}(\neg\phi)$ contains finitely many final states
   $M \times \mathcal{A}(\neg\phi) \models \mathsf{A}(\mathsf{FG}\neg\alpha)$

3. *Bound and count* the number of times a path in $M \times \mathcal{A}(\neg\phi)$ satisfies the formula α. This points consists in a cycle that starts with $k = 0$, increments k if a path containing more than k states satisfying α is found, or returns a positive result otherwise (all paths of $M \times \mathcal{A}(\neg\phi)$ contains finitely many final states). It can be proved that, for finite-state systems, there exists a $k_{max} \in \mathbb{N}$ such that $M \times \mathcal{A}(\neg\phi) \models \mathsf{A}(\mathsf{FG}\neg\alpha)$ if and only if each path in $M \times \mathcal{A}(\neg\phi)$ contains at most k_{max} occurrences of a state satisfing α (*i.e.*, a final state). This means that there exists an iteration on which the cycle stops.

Let us see how K-Liveness performs the last step. Let $k \in \mathbb{N}$. First of all, a *counter for* α is introduced: a counter is a Kripke structure that starts with value 0 and increments the value any time the current state satisfies α. Let $\#(\alpha)$ be this counter. The problem of proving that any path in $M \times \mathcal{A}(\neg\phi)$ has at most k states in which α holds can be reformulated as $M \models \mathsf{A}(\mathsf{G}(\#(\alpha) \leq k))$. Moreover $\mathsf{G}(\#(\alpha) \leq k)$ is a safety property, and thus any safety model checking algorithm (like IC3) can be used: this is the point in which K-Liveness gains in performance. If this algorithm returns a positive result ($M \models \mathsf{A}(\mathsf{G}(\#(\alpha) \leq k))$) then it means that $M \models \mathsf{A}\,\phi$. Otherwise, the index k is incremented and the cycle is repeated.

[3] Although M is a Kripke structure and not an automaton, Definition 36 can be adapted to the product between a Kripke structure and an automaton with straightforward modifications.

Completeness

Since it is guaranteed that:

$$M \models \mathsf{A}\,\phi$$
$$\Leftrightarrow$$
$$\exists k_{max} \in \mathbb{N} \,.\, M \models \mathsf{A}(\mathsf{G}(\#(\alpha) \leq k_{max}))$$

the cycle eventually stops. However in partice, like in the case for BMC (Section 5.2.5), the threshold (k_{max}) is too expensive to compute. In order to avoid it, K-Liveness starts two parallel processes: (i) one trying to proving the property ϕ by the technique above (*i.e.*, by bounding and counting the visits to a state satisfying α); and (ii) the other trying to violate ϕ, for example using the BMC algorithm. The first who terminates stops also the entire procedure.

Efficiency

K-Liveness proved to be very efficient in pratice. This is not only due to the reduction of an LTL model checking problem to a sequence of safety problems, but also to at least two other aspects.

The first one is that it uses IC3 as the back-end for solving the safety model checking problems. The particular structure of IC3 (*frames*) allows K-Liveness to use a sort of push/pop interface like the one for SAT-solvers (Section 5.2.4) for IC3 as well, thus allowing to save some work between one iteration and the other.

As for the second aspect, Claessen and Sörensson (the authors of [57]) propose also to automatically extract from M a set of constraints, called *stabilizing constraints*, that help IC3 to shrink the state-space during the search. Stabilizaing constraints proved to be particularly beneficial in practice.

5.2.7 From LTL satisfiability to LTL model checking

Another successful approach for solving LTL satisfiability is to reduce the problem to LTL model checking. Recall that LTL model checking (Definition 32) is the problem of establishing, given a Kripke structure M and an LTL-formula ϕ, whether all the initialized paths in M are models of ϕ, in symbols:

$$M \models \mathsf{A}\,\phi$$

Recall also that *model checking* amounts to a *language inclusion* problem; if we consider M as an automaton over infinite words, then model checking amounts to finding whether the language $\mathcal{L}(M)$ of M is a *subset* of the language $\mathcal{L}(\phi)$ recognized by ϕ, that is:

$$\mathcal{L}(M) \subseteq \mathcal{L}(\phi)$$

which in turn is equivalent to:

$$\mathcal{L}(M) \cap \overline{\mathcal{L}(\phi)} = \varnothing \tag{5.2}$$

We have already seen that Eq. (5.2) can be solved algorithmically by: (i) taking the product $M \times \mathcal{A}(\neg\phi)$ between the Kripke structure M and the automaton $\mathcal{A}(\neg\phi)$ for $\neg\phi$; (ii) checking the emptiness of $M \times \mathcal{A}(\neg\phi)$.

Now, LTL satisfiability solves the problem of finding whether $\mathcal{L}(\phi) = \varnothing$. In order to reduce LTL satisfiability to LTL model checking, it suffices to:

1. define M as the *trivial* structure, meaning that $\mathcal{L}(M) = (2^{\Sigma})^{\omega}$;
2. model check the formula $\neg\phi$ againts M, *i.e.*, $M \models \mathsf{A}(\neg\phi)$.
3. if the algorithm of model checking returns *true*, then ϕ is unsatisfiable, otherwise ϕ is satisfiable.

Let us take a look on why this works. By Eq. (5.2), it holds that $M \models \mathsf{A}(\neg\phi)$ if and only if $\mathcal{L}(M) \cap \overline{\mathcal{L}(\neg\phi)} = \varnothing$; since $\mathcal{L}(M) = (2^{\Sigma})^{\omega}$, this means that $M \models \mathsf{A}(\neg\phi)$ implies that:

$$\overline{\mathcal{L}(\neg\phi)} = \varnothing$$
$$\mathcal{L}(\phi) = \varnothing$$

and thus ϕ is *unsatisfiable*. If instead $M \not\models \mathsf{A}\,\neg\phi$, then

$$\overline{\mathcal{L}(\neg\phi)} \neq \varnothing$$
$$\mathcal{L}(\phi) \neq \varnothing$$

and thus ϕ is *satisfiable*.

5.3 LTL Realizability and synthesis

In this section, we take a look at the definitions and the main properties of the *realizability* and *reactive synthesis* problems of logical specifications (mainly LTL+P, but also S1S). We will then describe some techniques that have been introduced for solving these problem, starting from the classical approach and going through more sophisticated methods. We finish this section by describing safety games and the realizability problem from Safety-LTL specifications

5.3.1 Definition of the problem

Realizability and reactive synthesis are in some sense more ambitious problems than model checking and satisfiability, since they aim to find whether a given temporal formula ϕ over two sets $\mathcal{U}$ and $\mathcal{C}$ of uncontrollable and controllable variables, respectively, is implementable and, if this is the case, to synthesize a possible *implementation*. Usually, realizability is modeled as a two-player game between Environment, who tries to violate the specification and Controller, who tries to fulfill it. In this setting, an implementation of the specification is represented by a *strategy*.

Definition 41 (Strategies)**.** *Let $\mathcal{U}$ and $\mathcal{C}$ be two disjoint sets of* input *(or* uncontrollable*) and* output *(or* controllable*) variables, respectively. A* strategy *g is a function $g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$. We define the* language *of the strategy g, denoted as $\mathcal{L}(g)$, as the set of all and only the sequences $\langle(\mathsf{U}_0 \cup \mathsf{C}_0), (\mathsf{U}_1 \cup \mathsf{C}_1), \ldots\rangle$ such that $\mathsf{U}_i \in 2^{\mathcal{U}}$ and $\mathsf{C}_i = g(\langle \mathsf{U}_0, \ldots, \mathsf{U}_i\rangle)$, for all $i \geq 0$.*

We define the *language denoted by a strategy* $g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ as the set of *infinite sequences* $\langle(\mathsf{U}_0 \cup \mathsf{C}_0), (\mathsf{U}_1 \cup \mathsf{C}_1), \ldots\rangle$ such that, for each $i \geq 0$, the U_i set is the set of variables chosen by the *Environment player* to be true at the i^{th} time point, while C_i is the set of variables chosen by the *Controller player* according to the strategy g given the history so far, that is the set $g(\langle \mathsf{U}_0, \ldots, \mathsf{U}_i\rangle)$.

Definition 42 (Languages of Strategies)**.** *Let $g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ be a strategy. We define the* language *of the strategy g, denoted as $\mathcal{L}(g)$, as the set of all and only the sequences $\langle(\mathsf{U}_0 \cup \mathsf{C}_0), (\mathsf{U}_1 \cup \mathsf{C}_1), \ldots\rangle$ such that $\mathsf{U}_i \in 2^{\mathcal{U}}$ and $\mathsf{C}_i = g(\langle \mathsf{U}_0, \ldots, \mathsf{U}_i\rangle)$, for all $i \geq 0$.*

Realizability can be formalized as the problem of establishing the *existence* of a strategy that *implements* a given formula, in the sense

that the language of the strategy (Definition 42) is contained in the language of the formula (Sections 2.1 and 2.4). Reactive synthesis is the problem of *computing* such a strategy.

Definition 43 (Realizability and Reactive Synthesis)**.** *Let ϕ be a temporal formula over the alphabet $\Sigma = \mathcal{U} \cup \mathcal{C}$, where $\mathcal{U}$ is the set of input variables, $\mathcal{C}$ the set of output variables, and $\mathcal{U} \cap \mathcal{C} = \varnothing$. We say that ϕ is* realizable *if and only if there exists a strategy $g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ such that $\mathcal{L}(g) \subseteq \mathcal{L}(\phi)$. If ϕ is realizable, the reactive synthesis problem is the problem of computing such a strategy.*

The games derived from realizability are *determined*, that is there exists exactly one winner (either Controller or Environment). We say that ϕ_1 and ϕ_2 are equirealizable when the realizability of ϕ_1 implies the realizability (possibly with different strategies) of ϕ_2, and *vice versa*. With *safety synthesis* we refer to the problem of establishing whether a safety specification (Section 2.1.3) is realizable.

5.3.2 Decidability and Complexity

The strategies which we are mainly interested in are the ones that can be represented finitely. In the literature, there are two main (and equivalent) representations for finite strategies, that is, *Mealy machines* and *Moore machines*. In this paper, we are mainly interested in the first ones.

Definition 44 (Mealy Machine)**.** *A* Mealy machine *is a tuple $M = (\Sigma_{\mathcal{U}}, \Sigma_{\mathcal{C}}, Q, q_0, \delta)$ such that: (i) $\Sigma_{\mathcal{U}}$ and $\Sigma_{\mathcal{C}}$ are the input and output alphabets, respectively; (ii) Q is the (finite) set of states and q_0 is the initial state; (iii) $\delta : Q \times \Sigma_{\mathcal{U}} \to \Sigma_{\mathcal{C}} \times Q$ is the* total *transition* function*. We say that an infinite word $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle \in (\Sigma_{\mathcal{U}} \cup \Sigma_{\mathcal{C}})^\omega$ is* accepted *by M iff there exists a* trace *$\langle (q_0, \sigma_0), (q_1, \sigma_1), \ldots \rangle \in (Q \times (\Sigma_{\mathcal{U}} \cup \Sigma_{\mathcal{C}}))^\omega$ such that $\delta(q_i, \sigma_i \cap \Sigma_{\mathcal{U}}) = (\sigma_i \cap \Sigma_{\mathcal{C}}, q^{i+1})$, for all $i \geq 0$. We define the language of M, written as $\mathcal{L}(M)$, as the set of all the infinite words accepted by M.*

A fundamental feature is the *Finite Model Property* for realizability of LTL+P [90, 128, 160], which ensures that each realizable LTL+P formula has at least a finitely representable strategy. Interestingly, the same also holds for S1S [36].

Proposition 12 (Finite Model Property of LTL+P [160])**.** *Let ϕ be an* LTL+P *formula and $n = |\phi|$. If ϕ is realizable by a strategy g,*

then there exists a Mealy machine M_g such that (i) M_g has at most $2^{2^{c \cdot n}}$ states, for some constant $c \in \mathbb{N}$, and (ii) $\mathcal{L}(M_g) \subseteq \mathcal{L}(\phi)$.

We remark that the constant c depends on the algorithms used for building the *nondeterministic Büchi automaton* for the language $\mathcal{L}(\phi)$ and on *Safra's determinazion algorithm* to obtain an equivalent deterministic Rabin automaton (see [160]). This means that the constant c can be effectively computed, obtaining a concrete upper bound for the size of the Mealy machine in Proposition 12.

Decidability and Complexity

The finite model property of LTL+P (and S1S) realizability is at the core of the decidability of the problem. In fact, the search for the existence (and the corresponding synthesis) of a finite strategy is decidable, since it amounts to search for a subgraph corresponding to a Mealy machine implementing the formula inside the graph corresponding the (Büchi) automaton of the initial formula.

Proposition 13 (Decidability of S1S and LTL+P realizability [36, 160, 164])**.** *The realizability problem of* S1S *and* LTL+P *is decidable.*

Despite being both decidable, the realizability problems of S1S and LTL+P have two very different complexities:

1. The complexity of S1S realizability is proved to have a nonelementary lowerbound [36], like S1S satisfiability. The intuition is that realizability (like satisfiability) requires the construction of a Büchi automaton that is language-equivalent to the initial formula. Nevertheless, the complementation of a Büchi automaton (necessary to deal with negations inside the formula) produces in the general case an automaton with exponentially many states and transitions. Since an S1S formula can contain an arbitrary number of negations, this gives rise to a tower of exponentials of unbounded height (*i.e.*, the nonelementary complexity).

2. LTL+P realizability is 2EXPTIME-complete [160, 164]. The intuition is that, as we will see, the two-player game corresponding to realizability has nice algorithms only when played over *deterministic* automata (also called *arenas*). It can be proved that the construction of a deterministic arena starting

from an LTL+P specification along with the game solving algorithm require at most doubly exponenial time in the size of the initial formula. Intuitively, one exponential is for the automata construction, while the other one is for its determinization (the game solving algorithm is usually polynomial in the size of the arena/automaton).

Proposition 14 (Complexity of S1S and LTL+P realizability)**.** *It holds that:*

- S1S *realizability has a nonelementary lowerbound.*
- LTL+P *realizability is* 2EXPTIME*-complete.*

5.3.3 The classical approach

The classical approach to LTL+P realizability is the one proposed by Pnueli and Rosner in [160, 164]. On a high level, given an LTL+P formula ϕ, this approach consists in the construction of a (nondeterministic) Büchi automaton for the formula ϕ and its determinization. Consequently, since the latter automaton is deterministic, it can be seen as an automaton that, instead of reading words (eq. linear orders), it reads/accepts trees. Finally, the realizability of ϕ amounts to the emptiness checking of this tree automaton, which can be performed effectively by an algorithm. All these steps have been implemented inside the Lily tool [118].

Before going into the details of the classical approach of LTL+P realizability, let us give some preliminaries.

Preliminaries

In Section 2.2, we have seen many types of automata reading finite or infinite words, for example NFAs (*Nondeterministic Finite Automata*) or NBAs (*Nondeterministic Büchi automata*). One of the common aspects between all those automata is that they read *words*, that is (finite or infinite) sequences of letters belonging to an alphabet Σ. Moreover, we have seen only two types of accepting conditions: the one of NFAs (reachability of a final state) and the one of NBAs (the visit of a final state infinitely many times).

Over the years, research focused on some generalizations of those kinds of automata in two directions: (i) either by changing its accepting condition; (ii) or by changing the type of structures the

automaton reads. As for the first point, in this part we will describe the *Rabin accepting condition*, while for the second point we will give an overview of automata reading trees instead of words. We start with the latter point.

Automata on (finite or infinite) trees Formally, a tree is a generalization of a word, in the sense that a word is a degenerate tree where each node has at most (or exactly, in case of infinite trees) one child. A tree is *finite* if and only if each of its branches is finite, otherwise is *infinite*. A Σ-labeled tree is a tree where each node is labeled by a letter in Σ.

An automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ reading trees has the same form of the other automata that we have seen in Section 2.2. What changes is the definition of run and accepting run. In the case of automata reading finite (resp. infinite) trees, a *run* is a finite (resp. infinite) Q-labeled tree, *i.e.*, a tree labeled by states in Q. The run is accepting if *all* the branches satisfy the accepting condition on the set of final states F. For example, we can define the *nondeterministic Büchi automata on trees*, which accept only the infinite trees that induce runs containing only branches with infinitely many occurrences of a final state.

The Rabin accepting condition and Safra's construction In Büchi automata (Definition 11), the acceptance condition requires each accepting run to visit at least one final state *infinitely many times*. Although being very powerful (*e.g.*, NBA are expressively equivalent to S1S), the Büchi acceptance condition makes NBA *not closed under determinization*. In fact, recall from Section 2.2.3 that there exists an ω-language that cannot be accepted by any DBA (*Deterministic Büchi Automata*).

In some applications, like for instance realizability and reactive synthesis, it is important to deal with a deterministic automaton. While for finite words the determinization can be carried out by the classical *subset construction* (Section 2.2.1), the situation is not so simple for the infinite case. In fact, subset construction does not work for Büchi automata [167].

This led to the search for acceptance conditions that would allow the closure by determinization. One of these is the *Rabin acceptance condition*. Differently from the Büchi acceptance condition, which consists in a *set of states* that an accepting run has to visit infinitely

often, the Rabin condition consists of a *set of pairs of subsets of states*, *i.e.*, $\{(U_1, V_1), \ldots, (U_n, V_n)\}$ where $U_i, V_i \subseteq Q$ for each $i \geq 0$, and Q is the set of states of the automaton. A run is accepting with respect to the Rabin condition if and only if there exists at least one pair (U_i, V_i) such that the set of states that occur infinitely often in the run *does* interset V_i but *does not* interset U_i. Automata with a Rabin acceptance condition are called Rabin automata. When reading infinite words, we call NRA the class of *Nondeterministic Rabin Automata* and DRA the class of *Deterministic Rabin Automata*.

There are two notable properties of Rabin automata: (i) they are *closed under determinization*; and (ii) they are expressively equivalent to NBAs. These two features are used when one wants to build a deterministic automaton starting from an NBA (for example in realizability of LTL+P formulas). In particular, one typically transforms the initial NBA to a language-equivalent DRA. *Safra's algorithm* [167] is an algorithm for performing such a determinization and it was used by Pnueli and Rosner [160, 164] in one of steps of the classical approach to LTL+P realizability[4]that we shall see in the paragraphs below.

Realizability and Determinism

Realizability has a strong connection with determinism. As we will see, the modern approach to realizability is to consider it as a two-player game between the Environment player and the Controller player. This game is played over an arena, which typically is the automaton corresponding to the starting formula. Therefore, the typical approach for solving the realizability of a formula ϕ is the following:

1. build the automaton for the formula ϕ;
2. solve a game over the arena represented by the automaton.

As noticed by N. Piterman in [157]:

In the context of games, the opponent may be able to choose between different options. Using a deterministic automaton we can follow the game step by step and monitor the goal of the game.

[4]Safra's algorithm is known to be very hard both to understand and to implement, since it deals with complex data structures. For this reason, in practice, one tries always to avoid the use of this algorithm.

Therefore, determinism can be considered as a prerequisit for applying simple game solving algorithms.

The algorithm

The algorithm used by the classical approach to LTL+P realizability [160, 164] consists in the following steps. Let ϕ be an LTL+P formula.

1. Build the NBA $\mathcal{A}(\phi)$ recognizing the ω-language $\mathcal{L}(\phi)$.
2. Use Safra's algorithm for building a DRA $\mathcal{A}'(\phi)$ equivalent to $\mathcal{A}(\phi)$.
3. since $\mathcal{A}'(\phi)$ is deterministic, it is simple to build a DRA $\mathcal{A}''(\phi)$ reading infinite trees such that: (i) each branch of each tree corresponds to an ω-word accepted by the DRA $\mathcal{A}'(\phi)$, and *vice versa*; (ii) each infinite tree represents a strategy (recall Definition 41), meaning that at each level it contains any possible choice of the Environment player and the choice of the Controller player given the history so far (that is the labels on the path from that node to the root). Therefore, each infinite tree of $\mathcal{A}''(\phi)$ corresponds to a strategy of Controller for realizing ϕ, and *vice versa*.
4. Check the emptiness of the DRA $\mathcal{A}''(\phi)$. This check is effective.

Pnueli and Rosner proved that this procedure runs in doubly exponential time, thus showing the optimality of this algorithm (LTL+P is 2EXPTIME-complete).

5.3.4 The classical approach to safety synthesis

Recall that *safety syntesis* is the problem of establishing whether a safety specification (for example of Safety-LTL) is realizable. Also in this case, the problem is modeled as a two-player game between Environment and Controller. These games are called *safety games*.

Let us take a closer look at safety games and at the classical approach for solving safety synthesis. Since safety specifications have a strong connection with languages over *finite words* (Section 2.1.3), and since in the general case reasoning over finite words is simpler than reasoning over infinite words, the classical approach for safety

synthesis exploits this feature for the determinization step: in particular, the classical *subset construction* (Section 2.2.1) can be applied. The classical approach consists in the following steps.

1. From the safety specification ϕ, build the NFA $\mathcal{A}$ (over *finite words*) for the formula $\neg\phi$.
2. Determinize $\mathcal{A}$ into an equivalent DFA $\mathcal{A}'$, using the classical subset construction.
3. If Controller player can force the play to *never* visit a final states of $\mathcal{A}'$, this means that there exists a strategy realizing ϕ; otherwise, since the game is determined, there exists a strategy for Environment to win the game, and ϕ is unrealizable.

In the third point above, the game played by Controller is called *safety game* (Controller has to force the game to visit only safe states, in this case states that are not final), while the (dual) game played by Environment is called *reachability game* (Environment has to force the game to eventually reach a target/final state).

Solving safety games

Safety games (or equivalently reachability games) are typically played over arenas represented by DSSA (Deterministic Symbolic Safety Automata, Definitions 37 and 40). Let $\mathcal{A} = (V = X \cup \Sigma, I, T, \mathsf{G}\alpha(\mathsf{X})))$ be a SSA, where the input alphabet Σ is partitioned into controllable and uncontrollable variables ($\Sigma = \mathcal{U} \cup \mathcal{C}$). The safety game is solved by computing the set of *winning states* of Controller player, *i.e.*, the states starting from which Controller can force the game to visit only safe states. If the (only) initial state (recall that the automaton is deterministic) is contained in the winning set, then Controller has a winning strategy; otherwise (the game is determined) Environment wins.

The computation of the winning states for Controller is performed by computing a fixpoint over the set of states of the automaton with the *strong precedessor* operator, which, given a set of states S, returns all and only the states of the automaton from which Controller can force *a step* of the game to visit only states in S. Let $S(X')$ be the Boolean formula over the variables in X' denoting exactly the set of states S. The strong predecessor operator

is defined as follows:

$$\mathsf{SPred}(S(X')) \coloneqq \forall u \subseteq \mathcal{U} \,.\, \exists c \subseteq \mathcal{C} \,.\, \exists X'(S(X') \wedge T(X \cup u \cup c, X'))$$

It is worth noting that $\mathsf{SPred}(S(X'))$ is a formula over the variables X. The set of winning states of Controller is the *greatest fixpoint* of $\mathsf{SPred}(S(X'))$ starting from the set of all the safe states (denoted by the formula $\alpha(X)$):

$$\mathit{Win} \coloneqq \nu S(X) \,.\, (\mathsf{SPred}(S(X')) \wedge \alpha(X))$$

The greatest fixpoint can be computed, for example, by iteratively computing the set of states from which Controller can force the game to visit only safe states in at most i steps:

$$\begin{aligned} \mathit{Win}_0 &\coloneqq \alpha(X) \\ \mathit{Win}_{i+1} &\coloneqq \mathit{Win}_i \cap \mathsf{SPred}(\mathit{Win}_i) \end{aligned}$$

When we found that $\mathit{Win}_i = \mathit{Win}_{i+1}$ or $\mathit{Win}_i = \varnothing$, we can stop because we have reached the fixpoint, reporting the realizability or the unrealizability of the game, respectively.

Safety-LTL realizability

Safety-LTL (Definition 15) is defined as LTL where the temporal operators are restricted to be X, G or R. Recall also that Safety-LTL captures exatly the LTL-definable safety languages (Theorem 20).

Safety-LTL realizability is studied by Zhu *et al.* in [201]. The proposed algorithm follows these steps:

1. Given a formula $\phi \in$ Safety-LTL, consider its negation $\neg\phi$ and turn it into S1S[FO] interpreted over *finite words*.
2. Exploit the MONA tool [113] for building the corresponding NFA and for its consequent determinization into an NFA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$. The DFA is represented semi-symbolically as a deterministic SSA, meaning that the states of the automaton are explicitly represented, while the set of labels on each edge is represented with a BDD.
3. Play the safety game (as describe above) over $\mathcal{A}$.

5.3.5 The Safraless approach

In Section 5.3.3, we have seen that the classical approach to LTL+P realizability requires the *determinization* of a NBA corresponding to the starting formula. Determinization is necessary to apply either (i) emptiness checking algorithms, or (ii) game solving algorithms. We have also seen that the classical approach performs a determinization of the NBA into a DRA by using Safra's algorithm.

Since Safra's algorithm is known to be very complicated (and thus also error-prone and less amenable to optimizations and symbolic algorithms[5]), research focused on finding alternatives to Safra's algorithm. *The Safraless approach* has been introduced by Kupferman and Vardi in [128] and refers to the set of techniques for circumventing the use of Safra's algorithm by using different types automata and related acceptance conditions.

Preliminaries

We start with some preliminaries. Let $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ be an automaton. So far, have seen that, even if $\delta : Q \times \Sigma \times Q$ is a relation (nondeterminism), the automaton's run is still a sequence. This holds both for deterministic and nondeterministic automata (both on finite or infinite words). *Universal automata* change this perspective by running over all directions of the transition relation. Formally, a run of a universal automaton is a (finite or infinite) Q-labeled tree (where Q is the set of states of the automaton) such that, if the transition relation contains the triples $(q, \sigma, q_1), \ldots, (q, \sigma, q_n)$, then each node labeled with q in the tree has n children labeled with $q_1, \ldots, q_n$. In some sense, universal automata resolve the nondeterminism by having runs that are trees. It is worth noting that in universal automata the acceptance condition can be of any type. For example, we can have Universal Finite Automata, as well as Universal Büchi Automata.

The *coBüchi* acceptance condition is defined as the dual of the Büchi condition. Instead of specifying the set of its accepting states (like a Büchi automaton), a *coBüchi automaton* $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ specifies the set F of rejecting states. A run of $\mathcal{A}$ is rejecting when

[5]To the best of our knowledge, the only implementation of Safra's algorithm is the one implemented inside the Lily tool [118], and it was written nearly 20 years after the introduction of Safra's algorithm [167].

the set of states that occur infinitely often in the run does *not* intersect F. The language of a *Nondeterministic coBüchi Automaton* (NCA) is the set of ω-words inducing at least one rejecting run. The language of a *Universal coBüchi Automaton* (UCA) is the set of ω-words inducing a run tree whose branches are all rejecting. It is not difficult to see that NBA are duals of UCA, meaning that a language is recognized by an NBA iff its complement is recognized by an UCA.

The Safraless approach to LTL+P Realizability

The Safraless approach to LTL+P realizability proposed by Kupferman and Vardi in [128] consists in the following steps. Let $\phi \in$ LTL+P.

1. Build the NBA $\mathcal{A}$ for $\neg\phi$ (the negation of the initial formula) with standard constructions, like the one described in [198, 197].

2. When considered as a UCA, it easy to see that, by duality, $\mathcal{A}$ recognizes exactly the language $\mathcal{L}(\phi)$.

3. Like in the classical approach, from the UCA obtained in the previous step, build a UCA reading infinite trees (representing strategies) instead of infinite words.

4. Check the emptiness of this last automaton. This check is effective [128].

By following these steps, the approach is able to avoid the use of Safra's algorithm.

Several Safraless algorithms for LTL+P realizability have been introduced following the original idea by Kupferman and Vardi. One example is *Bounded Synthesis* (that we will describe in the next section). Another example is reported in the paper [90] by E. Filiot *et al.* They describe a Safraless algorithm that coincides with the one by Kupferman and Vardi in the first three steps. However, they avoid the emptiness check by observing that, if there is a strategy realizing ϕ, than the strategy forces the UCA to visit its rejecting states a finite and *bounded* number of times (k_{max}). From this observation, they develop an algorithm that, for each number k ranging from 0 to k_{max}, checks the existence of a strategy forcing the game to visit rejecting states at most k times. Since this is a safety objective (we will formally define it later), it is in general simpler to solve.

If a subgame of this type is found to be winning, then this implies also the realizability of the starting formula. Otherwise, eventually the upperbound k_{max} will be reached, and this witnesses the unrealizability of the starting formula[6].

5.3.6 Bounded Synthesis

Bounded Synthesis [92] is a Safraless algorithm for the realizability problem of temporal formulas. We take a look at it when applied to LTL+P formulas. The main idea behind bounded synthesis is to reduce the problem to a sequence of safety synthesis subproblems (equiv. safety games, Section 5.3.4) and then to encode each of these into a contraint system in such a way to guarantee that a solution of it corresponds to a strategy realizing ϕ, and *vice versa*. The great progress that constraint solvers have had in the last decades ensures the efficiency of the approach.

On a lower level of detail, bounded synthesis considers an LTL+P formula ϕ and builds the corresponding UCA (Universal Büchi Automaton). It then builds a sequence of safety synthesis subproblem by bounding the following two quantities:

1. the number of visits of the play to the rejecting states of the automaton; and

2. the size of a candidate strategy/implementation (measured as the number of states of the corresponding Mealy machine, Definition 44).

While the first point reseambles the safety reduction of [90], the second point is the main novelty of bounded synthesis and allows for the realizability and synthesis also in contexts in which they are undecidable in the general case.

Each of the safety synthesis subproblems (safety games) is encoded into a constraint system. In the original paper [92], Finkbeiner and Schewe use SMT modulo LRA (*Linear Real Arithmetic*) and UI (*Uninterpreted Functions*) for the encoding (Section 5.2.4). More recently, several encodings for bounded synthesis have been proposed, for example into Boolean formulas, quantified Boolean formulas (QBF), and Dependency quantified Boolean formulas (DQBF) [88]. Among the different encodings, QBF seems the most efficient

[6]This algorthm has been implemented in a tool called Acacia [90].

in practice. Bounded synthesis has been implemented in several tools, for example UNBEAST [82, 81] (that uses BDDs for representing the safety games) and BOSY [89] (that implements all the above-mentioned encodings).

5.3.7 The parity games approach

Another successful approach for LTL+P realizability, that is often used also in practice (for example by the STRIX tool [150, 137]), is the one based on parity games.

The parity acceptance condition

We start by defining the *parity acceptance condition*. Let $\mathcal{A} = (\Sigma, Q, I, \delta, L)$ be an automaton, where $L : Q \to \{1, \ldots, |Q|\}$ is a function labeling each state with a natural number between 1 and $|Q|$. A run of $\mathcal{A}$ is *accepting* with respect to the parity condition iff the greatest between the labels of the states occurring infinitely often in the run is *even* (hence the name *parity automata*). An automaton with a parity acceptance condition is called *parity automaton*. We call NPA the set of *Nondeterministic Parity Automata* and DPA the set of *Deterministic Parity Automata*. The parity acceptance condition has proved to be very powerful. In fact, one can show that many other conditions, like for instance the Büchi or the Rabin ones, are special instances of the parity condition [128].

With *parity game* we refer to the game played over an arena represented by a parity automaton, where the objective is to force the play to fulfill the parity acceptance condition.

The algorithm

We take a look at the algorithm based on parity games described in [150, 137]. Since it avoids Safra's construction, it can be considered as a Safraless approach. Let $\phi \in$ LTL+P. The algorithm consists in the following steps.

1. Build the limit-deterministic Büchi automaton as described in [175]. These kind of automata are made of a nondeterministic component guessing the eventually true formulas, and of a deterministic component verifying this guess.

2. From the automaton obtained in the previous point, build the language-equivalent DPA (Deterministic Parity Automaton) as described in [87].

3. The DPA obtained in the previous point is considered as a parity game, and any algorithm for solving parity games can be used as backend

Parity games have been thorougly studied in the last years [169, 96], maybe due to several reasons: (i) modal μ-calculus model checking can be reduced to parity game solving [86]; and (ii) to the best of our knowledge, the exact theoretical complexity of the problem is still unkown so far (2021). It has been proved that the problem stands in the intersection $\mathsf{NP} \cap \mathsf{coNP}$ but no proof of the NP-completeness and no polynomial time algorithms have been given yet. An interesting result by M. Jurdzinski proves that parity game solving stands in the intersection $\mathsf{UP} \cap \mathsf{coUP}$ [119], where UP comprises the problems that can be solved in nondeterministic polynomial time by an *unambigous Turing machine* (*i.e.*, Turing machines having a unique polynomial-size certificate), and coUP is the dual class of UP.

Several optimizations are possible during the game solving process. For example, the STRIX tool [150, 137] implements several optimizations that allows to tackle the state-space explosion problem by avoiding to explore the whole set of states of the parity game.

5.3.8 GR(1) realizability

Recall from Definition 18, that GR(1) is defined as the set of formulas of type:

$$(\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

for any $m, n \in \mathbb{N}$, where $\alpha_i, \beta_j \in \mathsf{LTL}{+}\mathsf{P_P}$ for each $i \in \{1, \ldots, m\}$ and for each $j \in \{1, \ldots, n\}$.

In the context of realizability and synthesis, GR(1) has another definition (Bloem *et al.* [25]) which slightly differs from the original one. Take an alphabet $\Sigma = \mathcal{U} \cup \mathcal{C}$ where $\mathcal{U}$ is the set of uncontrollable variables, $\mathcal{C}$ is the set of controllable variables, and $\mathcal{U} \cap \mathcal{C} = \varnothing$. A

GR(1) formula is of type:

$$(\alpha \wedge \mathsf{G}\alpha' \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\beta \wedge \mathsf{G}\beta' \wedge \bigwedge_{i=1}^{n} \mathsf{GF}\beta_i) \tag{5.3}$$

for any $m, n \in \mathbb{N}$, satisfying the following constraints:

- α is a Boolean formula over $\mathcal{U}$.
- β is a Boolean formula over Σ.
- α' is a Boolean formula over Σ augmented with the X temporal operator, such that the formulas inside a X operator cannot contain variables in $\mathcal{C}$.
- β' is a Boolean formula over Σ augmented with the X temporal operator.
- each α_i and each β_i are pure past formulas over the alphabet Σ.

Intuitively, these contraints ensure that α and α' are formulas *controlled by the Environment player* (and the same holds also for β and β').

GR(1) is considered a good specification language because: (i) it provides constructs for the assumptions-guarantees paradigm, which is particularly useful in the context of realizability and synthesis, because typically a specification in this context is partitioned into assumptions about the environment and guarantees for the controller. (ii) it combines both safety and fairness (recurrence) formulas.

In [25], two types of realizability and synthesis are considered, the *strict* and the *standard* realizability, depending on how the implication is interpreted. Before going into the details, let us note from Definition 18 and Eq. (5.3) that each GR(1) formula is partitioned into six components:

- two Boolean formulas (α and β) constraining the initial time point;
- two pure past formulas (α' and β') that are meant to hold in each time point;
- two conjunctions of pure past formulas (α_i and β_j) that are meant to hold infinitely many times.

These formulas naturally represent two symbolic Büchi automata (Definition 38), since:

- α and β are the formulas for the set of initial states of the two automata;
- α' and β' are the formulas for the two transition relations;
- α_i and β_j corresponds to the final states of the two automata.

It is worth noticing that, although α, α' and each α_i are pure past formulas, they can be easily turned into Boolean formulas by adding new state variables to the automaton, in order to conform to Definition 38 (the same holds for β, β' and each β_j as well).

Standard and Strict realizability

In [25], *standard realizability* is defined as the realizability of the formula

$$(\alpha \wedge \mathsf{G}\alpha' \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\beta \wedge \mathsf{G}\beta' \wedge \bigwedge_{i=1}^{n} \mathsf{GF}\beta_i)$$

The typical problem of standard realizability is that, as soon as the Environment player violates one of its assumptions, the Controller player can behave arbitrarly, since the head of the implication is violated and thus the whole formula is trivially true. However, what one has typically in mind is that

Controller has to behave in conformance to its guarantees as long as Environment fulfills its assumptions.

For example, we would like the Controller to satisfy β even if Environment satisfies α but at some point it violates α'. In order to accomodate this more natural definition of the problem, Bloem *et al.* [25] consider the *strict realizability* problem, defined as the (standard) realizability problem of the following formula:

$$\begin{aligned}
&(\alpha \rightarrow \beta) \wedge \\
&(\mathsf{G}(\mathsf{H}(\alpha') \rightarrow \beta')) \wedge \\
&((\alpha \wedge \mathsf{G}\alpha' \wedge \bigwedge_{i=0}^{m} \mathsf{GF}\alpha_i) \rightarrow \bigwedge_{i=0}^{m} \mathsf{GF}\beta_i)
\end{aligned}$$

Solving the GR(1) game

We briefly describe the algorithm proposed in [25] for solving strict and standard realizability from GR(1) specifications. We start with strict realizability. The algorithm consists in two steps:

1. Build the arena for the game. This is done symbolically by considering the formulas for the initial states (α and β) and for the transition relations (α' and β').

2. A triple fixpoint is used to characterize the winning states of Controller player with respect to the fairness conditions $\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i$ and $\bigwedge_{i=1}^{n} \mathsf{GF}\beta_i$. An algorithm for computing the triple fixpoint is used for deciding the (un)realizability of the initial formula.

As pointed out in [25], standard GR(1) realizability can be reduced to strict GR(1) realizability with the introduction of additional variables.

This algorithm, together with some optimization, has been implemented inside the SLUGS tool [84].

5.3.9 Reactive Synthesis vs Parameter Synthesis

Given a temporal specification ϕ over a set of controllable and uncontrollable variables, reactive synthesis (Definition 43) refers to the problem of synthesizing a correct-by-construction controller (represented by means of a strategy, or a Mealy machine), that satisfies ϕ no matter what the values of the uncontrollable variables are.

In formal verification, another important problem that goes under the name of *"synthesis"* is *parameter syntesis*. Informally, a parametric system is a *partially specified system*: during design time, *not* all details may be known, and therefore some degrees of its working may be left uncontrained. Consider for example a real-time system. It may be the case that not all the real-time bounds are known at design time, for example due to uncertainties about the environment. In this case, the choice may be to consider the bounds as *parametric*, that is replacing the bounds with constants (parameters). Given a specification ϕ, *Parameter Synthesis* refers to the problem of synthesizing the set of values of the parameters (the *parameter region*) such that, when replaced to the parameters in the

system, they produce a *concrete* system that satisfies ϕ. We will see an application of parameter synthesis in Chapter 9.

Therefore, *reactive synthesis* must not be confused with *parameter synthesis*. As a matter of fact, the techniques for solving the former are intrinsically different from those for solving the latter.

CHAPTER

6

SATISFIABILITY OF LTL+P SPECIFICATIONS

In this chapter, we propose a satisfiability checking procedure for LTL+P formulas based on a *SAT encoding* of the one-pass and tree-shaped tableau, proposed by Reynolds [163] for LTL and extended to LTL+P and TPTL in [101]. We refer to Section 5.1.4 for the details of the tableau system.

The tableau tree is (symbolically) built in a breadth-first way, by means of Boolean formulas that encode all tableau branches up to a given depth k, which is increased at every step. The termination rules of the tableau system are encoded in Boolean formulas as well, in such a way that a successful assignment represents a branch of the tree of length k, which *directly* corresponds to a model of the original LTL+P formula. This breadth-first iterative deepening approach has been exploited in the past by *bounded satisfiability checking* and *bounded model checking* algorithms [56, 112] (recall Section 5.2.5), which share with us the advantage of leveraging the great progress

of SAT solvers in the last decades, and the *incrementality* of such solvers.

A common drawback of existing bounded satisfiability checking methods is the difficulty in identifying when to stop the search in the case of *unsatisfiable* formulae. In order to ensure termination, either a global upper bound has to be computed in advance, which is not always possible or feasible, or some other techniques are needed to identify when the search can be stopped. In our system, termination is guaranteed by a suitable encoding of the tableau's PRUNE rule. This rule was the main novelty of Reynolds' one-pass and tree-shaped system when it was originally proposed [163], has a clean model-theoretic interpretation [101], and the important role it plays in our encoding adds up to its interesting properties. The result is a simple and complete bounded satisfiability checking procedure based on a small and much simpler SAT encoding.

We implemented the proposed procedure and encoding in a tool, called BLACK[1]for (Bounded LTL sAtisfiability ChecKer), and we report the outcomes of our experimental evaluation, comparing it with state-of-the-art tools. The results are promising, consistently improving over the tableau explicit construction.

The modularity of Reynolds' tableau and of its encoding allows BLACK to support both *future* and *past* temporal operators, interpreted both on *finite* and *infinite* traces. On one side, *past* temporal operators do not add expressive power to the logic but do increase succinctness (recall Corollary 3 and Proposition 4) and allow for many properties to be expressed in a more natural way [136]. On the other side, while LTL+P has been historically defined as interpreted over infinite traces, the finite-trace semantics (recall Section 2.4.2) has recently seen much interest in the artificial intelligence and business modeling communities [73]. Independently from the set of temporal operators and the class of models considered, BLACK is able to output a model for satisfiable instances. The support for these features makes BLACK a quite flexible tool.

This chapter is structured as follows. Section 6.1 recalls the literature related to our work. In Section 6.2, we describe in details the algorithm and the SAT encoding of the tableau for LTL+P, including full proofs of soundness, completeness and termination. Section 6.3 shows how to extend the encoding for addressing the satisfiability of LTL+P under *finite words* interpretation and for extracting models

[1]BLACK can be downloaded from `https://github.com/black-sat/black`

of satisfiable formulas. Finally, Section 6.4 experimentally compares the tool with other state-of-the-art solvers, and Section 6.5 concludes with final remarks and a discussion of future developments.

6.1 Related work

In this part, we show a comparison between the encoding used by BLACK with respect to bounded model checking (Section 5.2.5). We also recall the importance of the finite semantics for LTL+P and we conclude with the role of SAT- and SMT-solvers for BLACK.

Bounded Model Checking

As we have already seen in Section 5.2.7, LTL+P satisfiability can be reduced to LTL+P model checking: to tell whether a formula is satisfiable, its negation is model-checked against the *complete transition system* (*i.e.*, a transition system generating all possible traces over an alphabet Σ), with any counterexample being a model of the original formula. Hence, any model checking technique can be seen as an alternative satisfiability checking technique. In this sense, the encoding presented here is similar in spirit to *bounded model checking* [17, 56] techniques (recall Section 5.2.5): a counterexample (here, a tableau branch) of length (tree depth) up to k, for increasing values of k is found by encoding the paths of the structure (the branches of the tree) up to length (depth) k into a SAT formula.

However, bounded model checking techniques are usually incomplete, since the computation of the diameter of the graph, which witnesses the exploration of *all paths*, is usually a very hard task (requiring for example to solve the satisfiability problem of a quantified Boolean formula [17]). Here, instead, our algorithm is complete thanks to the encoding of the PRUNE rule of Reynolds' tableau (see Section 5.1.4). In addition to that, the encoding for past operators, coming from the tableau rules, is much simpler than the *virtual unrollings* technique used to support past operators in bounded model checking approaches [18]. Indeed, support for past operators comes almost for free in our encoding, which allows BLACK to support this handy feature much efficiently. This was surprising at first, since support for past operators (sketched in *et al.* [107] and finalized in *et al.* [101]) is a bit more involved in the explicit construction of Reynolds' tableau.

Importance of finite semantics

As already pointed out in this thesis, although LTL+P has been historically defined over infinite traces, the finite-trace semantics has recently gained popularity in the artificial intelligence [73] and business process modeling fields [70]. Although the computational complexity of all the main problems remain the same, the manipulation of finite state automata on finite words instead of Büchi automata guarantees a notable speedup in practice. This led to much work revisiting, for example, model checking and synthesis [74], and the use of LTL on finite traces as specification language for non-markovian rewards in Markov Decision Processes [31], for restraining specifications in reinforcement learning applications [72], and for specifications of temporally extended goals in fully observable nondeterministic planning [30].

SAT- and SMT-solvers

The good performance of BLACK would not be possible without the use of efficient SAT solvers as the backend. The satisfiability problem for propositional logic is the canonical NP-complete problem and one of the most studied problems in computer science. For this reason, the efficiency of modern SAT solver has grown beyond the best expectations. BLACK supports different solvers as backend in order to be able to exploit the advantages of each. In addition to the classic but now outdated MiniSAT [80], we support CryptoMiniSAT [182], a modern, parallelized and very flexible SAT solver. In addition to that, we support two Satisfiability Modulo Theories (SMT) solvers, Z3 [76] and MathSAT [54]. Although BLACK does not make use of any SMT feature, the two SMT solvers proved to be very competitive backends also for purely propositional problems.

6.2 The SAT encoding of Reynolds' tableau

We refer to Section 5.1.4 for the description of the set of rules of the Reynolds' tableau. The BLACK satisfiability checker is based on an iterative procedure that symbolically explores the tableau tree breadth-first by means of a SAT encoding of the tableau branches up to a given depth k, for increasing values of k. The satisfiability checking procedure employed by BLACK is reported in Algorithm 1.

Algorithm 1 BLACK's main procedure

```
1:  procedure BLACK(φ)
2:      k ← 0
3:      while True do
4:          if ⟪φ⟫^k is UNSAT then
5:              return φ is UNSAT
6:          end if
7:          if |φ|^k is SAT then
8:              return φ is SAT
9:          end if
10:         if |φ|^k_T is UNSAT then
11:             return φ is UNSAT
12:         end if
13:         k ← k + 1
14:     end while
15: end procedure
```

The three formulas $\langle\!\langle \phi \rangle\!\rangle^k$, $|\phi|^k$, and $|\phi|^k_T$ encode different rules of the tableau.

Let us start with some notation. Let ϕ be an LTL+P formula in NNF over the alphabet Σ. We define the following sets of formulas:

$$\begin{aligned} \mathsf{XR} &= \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{tomorrow} \text{ formula}\} \\ \mathsf{YR} &= \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{yesterday} \text{ formula}\} \\ \mathsf{ZR} &= \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{weak yesterday} \text{ formula}\} \\ \mathsf{XEV} &= \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is an } \mathsf{X}\textit{-eventuality}\} \end{aligned}$$

The three encoding formulas are defined over an extended alphabet $\overline{\Sigma}$, which includes:

1. any proposition letter from the original alphabet Σ;
2. the set $\{p_\psi \mid \psi \in \mathsf{XR}, \mathsf{YR}, \mathsf{ZR}\}$, that is, the set of all the *grounded* X-, Y-, and Z-*requests*;
3. a *stepped* version p^k of all the proposition letters defined in items 1 and 2, with $k \in \mathbb{N}$ and p^0 identified as p.

Intuitively, different stepped versions of the same proposition letter p are used to represent the value of p at different states. Thus,

when p^i holds, it means that p holds at the i-th step node of the branch, *i.e.*, the i-th state of the model.

Moreover, given $\psi \in \mathcal{C}(\phi)$, we denote by ψ_G the formula in which all the X-, Y-, and Z-*requests* are replaced by their grounded version. Similarly, given $\psi \in \mathcal{C}(\phi)$, we denote by ψ^k the formula in which all proposition letters are replaced by their k stepped version. We write ψ_G^k to denote $(\psi_G)^k$.

The formula $\langle\!\langle \phi \rangle\!\rangle^k$ is called the *k-unraveling* of ϕ, and encodes the expansion of the tableau tree. To define it, we need to encode the expansion rules of Table 5.1.

Definition 45 (Stepped Normal Form)**.** *Given an* LTL+P *formula* ϕ *in NNF, its* stepped normal form*, denoted by* $\mathrm{snf}(\phi)$*, is defined as follows:*

$$
\begin{aligned}
\mathrm{snf}(\ell) &= \ell && \text{where } \ell \in \{p, \neg p\}, \text{ for } p \in \Sigma \\
\mathrm{snf}(\otimes \phi_1) &= \otimes \phi_1 && \text{where } \otimes \in \{\mathsf{X}, \mathsf{Y}, \mathsf{Z}\} \\
\mathrm{snf}(\phi_1 \otimes \phi_2) &= \mathrm{snf}(\phi_1) \otimes \mathrm{snf}(\phi_2) && \text{where } \otimes \in \{\wedge, \vee\} \\
snf(\phi_1 \,\mathsf{U}\, \phi_2) &= \mathrm{snf}(\phi_2) \vee (\mathrm{snf}(\phi_1) \wedge \mathsf{X}(\phi_1 \,\mathsf{U}\, \phi_2)) \\
\mathrm{snf}(\phi_1 \,\mathsf{R}\, \phi_2) &= \mathrm{snf}(\phi_2) \wedge (\mathrm{snf}(\phi_1) \vee \mathsf{X}(\phi_1 \,\mathsf{R}\, \phi_2)) \\
\mathrm{snf}(\phi_1 \,\mathsf{S}\, \phi_2) &= \mathrm{snf}(\phi_2) \vee (\mathrm{snf}(\phi_1) \wedge \mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)) \\
\mathrm{snf}(\phi_1 \,\mathsf{T}\, \phi_2) &= \mathrm{snf}(\phi_2) \wedge (\mathrm{snf}(\phi_1) \vee \mathsf{Z}(\phi_1 \,\mathsf{T}\, \phi_2))
\end{aligned}
$$

The stepped normal form is the extension to past operators of the *next normal form* used in [101]. It can be noted how it follows from the expansion rules of each operator in Table 5.1. We can now define the k-unraveling of ϕ recursively as follows:

$$
\begin{aligned}
\langle\!\langle \phi \rangle\!\rangle^0 &= \mathrm{snf}(\phi)_G \wedge \bigwedge_{\psi \in \mathsf{YR}} \neg \psi_G \wedge \bigwedge_{\psi \in \mathsf{ZR}} \psi_G \\
\langle\!\langle \phi \rangle\!\rangle^{k+1} &= \langle\!\langle \phi \rangle\!\rangle^k \wedge S_k \wedge Y_k \wedge Z_k
\end{aligned}
$$

where

$$
\begin{aligned}
S_k &\equiv \bigwedge_{\mathsf{X}\alpha \in \mathsf{XR}} \left((\mathsf{X}\alpha)_G^k \leftrightarrow \mathrm{snf}(\alpha)_G^{k+1} \right) \\
Y_k &\equiv \bigwedge_{\mathsf{Y}\alpha \in \mathsf{YR}} \left((\mathsf{Y}\alpha)_G^{k+1} \leftrightarrow \mathrm{snf}(\alpha)_G^{k} \right) \\
Z_k &\equiv \bigwedge_{\mathsf{Z}\alpha \in \mathsf{ZR}} \left((\mathsf{Z}\alpha)_G^{k+1} \leftrightarrow \mathrm{snf}(\alpha)_G^{k} \right)
\end{aligned}
$$

The S_k, Y_k and Z_k formulas encode, respectively, the STEP, YESTERDAY, and W-YESTERDAY rules of the tableau, while the base case of the 0-unraveling ensures that *yesterday* formulas are false and *weak yesterday* formulas are true at the first state. The CONTRADICTION rule of the tableau is implicitly encoded in the fact that only satisfying assignments of the formula are considered. Note that the FORECAST rule as well does not need to be explicitly encoded: the intrinsic nondeterminism of the SAT solving process accounts for the nondeterministic choices implemented by the rule.

Intuitively, if $\langle\!\langle \phi \rangle\!\rangle^k$ is unsatisfiable, all the branches of the tableau for ϕ are rejected before $k+1$ steps.

Lemma 13. *Let ϕ be an* LTL+P *formula. Then, $\langle\!\langle \phi \rangle\!\rangle^k$ is unsatisfiable if and only if all the branches of the complete tableau for ϕ are crossed by the CONTRADICTION or (W-)YESTERDAY rules and contain at most $k+1$ step nodes.*

Proof. We prove the contrapositive, *i.e.*, that $\langle\!\langle \phi \rangle\!\rangle^k$ is satisfiable if and only if the complete tableau for ϕ has at least a branch that is either accepted, crossed by PRUNE, or longer than $k+1$ step nodes. To do that we establish a connection between truth assignments of $\langle\!\langle \phi \rangle\!\rangle^k$ and suitable branches of the tableau.

From branches to assignments. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be a branch that is either accepted, crossed by PRUNE, or longer than $k+1$ step nodes. Let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. We define a truth assignment ν for $\langle\!\langle \phi \rangle\!\rangle^k$ as follows. Note that $\langle\!\langle \phi \rangle\!\rangle^k$ contains stepped propositions from p^0 until p^k for any given p, so we need at most $k+1$ step nodes from $\overline{u}$, which however can be shorter if it is accepted or crossed by the PRUNE rule. Hence, let us define $\ell = \min\{m, k\}$. Moreover, let us define p_U to be p if $p \in \Sigma$, and to be ψ if $p = \psi_G$ for some X-, Y-, or Z-request ψ, *i.e.*, $(\cdot)_U$ is the inverse of the $(\cdot)_G$ operation. Then, for $0 \leq i \leq \ell$, we set $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$. Then, we complete the assignments for positions $m < j \leq k+1$ (if any) as follows:

1. if the branch has been accepted by the EMPTY rule, all the other positions $j > m$ can be filled arbitrarily;

2. if the branch has been accepted by the LOOP or crossed by the PRUNE rule, then there is a position w such that $\Gamma(\pi_w) = \Gamma(\pi_m)$. Then we continue filling the truth assignment considering the successor of π_w as a successor of π_m.

It can be verified that the truth assignment so constructed satisfies $\langle\!\langle \phi \rangle\!\rangle^k$.

From assignments to branches. Let ν be a truth assignment for $\langle\!\langle \phi \rangle\!\rangle^k$. We use ν as a guide to navigate the tableau tree to find a suitable branch which is either accepted, crossed by PRUNE, or has more than $k+1$ step nodes. To do that we build a sequence of branch prefixes $\overline{u}_i = \langle u_0, \ldots, u_i \rangle$ where at each step we obtain $\overline{u}_{i+1}$ by choosing u_{i+1} among the children of u_i, until we find a leaf or we reach $k+1$ step nodes. During the descent, we build a partial function $J : \mathbb{N} \to \mathbb{N}$ that maps positions j in $\overline{u}_i$ to indexes $J(j)$ such that for all ψ it holds that $\psi \in \Gamma(u_j)$ if and only if $\nu \models \mathrm{snf}(\psi)_G^{J(j)}$, *i.e.*, we build a relationship between positions in the branch and steps in ν. As the base case, we put $\overline{u}_0 = \langle u_0 \rangle$ and $J(0) = 0$ so that the invariant holds since $\Gamma(u_0) = \{\phi\}$ and $\nu \models \mathrm{snf}(\phi)_G^0$ by the definition of $\langle\!\langle \phi \rangle\!\rangle^k$. Then, depending on the rule that was applied to u_i, we choose u_{i+1} among its children as follows:

1. if the STEP rule has been applied to u_i, then there is a unique child that we choose as u_{i+1}, and we define $J(i+1) = J(i)+1$. Now, for all $\mathsf{X}\alpha \in \Gamma(u_i)$, we have $\alpha \in \Gamma(u_{i+1})$ by construction of the tableau. Note that $\mathrm{snf}(\mathsf{X}\alpha) = \mathsf{X}\alpha$, hence we know by construction that $\nu \models (\mathsf{X}\alpha)_{\mathsf{G}}^{J(j)}$. Then, by definition of $\langle\!\langle \phi \rangle\!\rangle^k$, we know that $\nu \models \mathrm{snf}(\alpha)_G^{J(j)+1}$, *i.e.*, $\nu \models \mathrm{snf}(\alpha)_G^{J(i+1)}$. On the other direction, if $\nu \models \mathrm{snf}(\alpha)_G^{J(i+1)}$, then by definition of $\langle\!\langle \phi \rangle\!\rangle^k$ we have $\nu \models (\mathsf{X}\alpha)_G^{J(i)}$, hence $\nu \models \mathrm{snf}(\mathsf{X}\alpha)_G^{J(i)}$, hence $\mathsf{X}\alpha \in \Gamma(u_i)$, so by construction of the tableau we have $\alpha \in \Gamma(u_{i+1})$. Hence the invariant holds.

2. if the FORECAST rule has been applied to u_i, then there are n children $\{u_i^1, \ldots, u_i^n\}$ such that $\Gamma(u_i) \subseteq \Gamma(u_i^m)$ for all $1 \leq m \leq n$. Now, we set $J(i+1) = J(i)$ and we choose u_{i+1} as a child u_i^m with a label $\Gamma(u_i^m)$ such that for any ψ we have $\psi \in \Gamma(u_{i+1})$ if and only if $\nu \models \mathrm{snf}(\psi)_G^{J(i+1)}$. Note that at least one such child exists, because at least one child has the same label as u_i. Thus the invariant holds by construction.

3. if an *expansion rule* has been applied to u_i, then there are one or two children. In both cases, we set $J(i+1) = J(i)$. Then:

   (a) if there is one child, then it is chosen as u_{i+1}. In this case, the rule is the CONJUNCTION rule and has been applied

to a formula $\psi \equiv \psi_1 \wedge \psi_2$, hence $\psi_1, \psi_2 \in \Gamma(u_{i+1})$. By construction we know that $\nu \models \mathrm{snf}(\psi)_G^{J(i)}$, hence $\nu \models \mathrm{snf}(\psi)_G^{J(i+1)}$. Now, note that $\mathrm{snf}(\psi_1 \wedge \psi_2) = \mathrm{snf}(\psi_1) \wedge \mathrm{snf}(\psi_2)$, so it holds that $\nu \models \mathrm{snf}(\psi_1)_G^{J(i+1)}$ and $\nu \models \mathrm{snf}(\psi_2)_G^{J(i+1)}$. On the other direction, if $\nu \models \mathrm{snf}(\psi_1)_G^{J(i+1)}$ and $\nu \models \mathrm{snf}(\psi_2)_G^{J(i+1)}$ we know that $\nu \models \mathrm{snf}(\psi_1 \wedge \psi_2)_G^{J(i+1)}$ hence $\nu \models \mathrm{snf}(\psi_1 \wedge \psi_2)_G^{J(i)}$, hence by construction we have $\psi_1 \wedge \psi_2 \in \Gamma(u_i)$ and so we have $\psi_1, \psi_2 \in \Gamma(u_i)$, hence the invariant holds.

(b) if there are two children u_i' and u_i'', then let us suppose the rule applied is the DISJUNCTION rule. Similar arguments will hold for the other rules. In this case, the rule has been applied to a formula $\psi \equiv \psi_1 \vee \psi_2$, hence $\psi_1 \in \Gamma(u_i')$ and $\psi_2 \in \Gamma(u_i'')$. We know $\nu \models \mathrm{snf}(\psi)_G^{J(i)}$, hence $\nu \models \mathrm{snf}(\psi)_G^{J(i+1)}$. Since $\mathrm{snf}(\psi_1 \vee \psi_2) = \mathrm{snf}(\psi_1) \vee \mathrm{snf}(\psi_2)$, it holds that either $\nu \models \mathrm{snf}(\psi_1)_G^{J(i+1)}$ or $\nu \models \mathrm{snf}(\psi_2)_G^{J(i+1)}$. Now, we choose u_{i+1} accordingly, so to respect the invariant. Note that if both nodes are eligible, which one is chosen does not matter. The other direction of the invariant also holds, since if either $\nu \models \mathrm{snf}(\psi_1)_G^{J(i+1)}$ or $\nu \models \mathrm{snf}(\psi_2)_G^{J(i+1)}$, then $\nu \models \mathrm{snf}(\psi_1)_G^{J(i)}$ or $\nu \models \mathrm{snf}(\psi_2)_G^{J(i)}$, so $\nu \models \mathrm{snf}(\psi_1 \vee \psi_2)_G^{J(i)}$, hence $\psi_1 \vee \psi_2 \in \Gamma(u_i)$, hence either $\psi_1 \in \Gamma(u_{i+1})$ or $\psi_2 \in \Gamma(u_{i+1})$.

Let $\overline{u} = \langle u_0, \ldots, u_i \rangle$ be the branch prefix constructed as above, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_n \rangle$ be the sequence of its step nodes. As mentioned, the descent stops when π_n is a leaf or when $n = k + 1$. Note in any case that $u_i = \pi_n$. In case we find a leaf, note that it cannot have been crossed by the CONTRADICTION rule. Otherwise, we would have $\{p, \neg p\} \subseteq \Gamma(u_i)$, which would mean $\nu \models p^{J(i)}$ and $\nu \models \neg p^{J(i)}$, which is not possible. Moreover, it cannot have been crossed by the YESTERDAY rule, since that would mean there is some $\mathsf{Y}\alpha \in \Gamma(\pi_n)$ with $\alpha \notin \Gamma^*(\pi_{n-1})$. But, we know that $\nu \models \mathrm{snf}(\mathsf{Y}\alpha)_G^{J(i)}$, hence $\nu \models (\mathsf{Y}\alpha)_G^{J(i)}$ since $\mathrm{snf}(\mathsf{Y}\alpha) = \mathsf{Y}\alpha$. Then, by definition of $\langle\!\langle \phi \rangle\!\rangle^k$, we know that $\nu \models \mathrm{snf}(\alpha)_G^{J(i)-1}$. Since $u_i = \pi_n$ is a step node, $J(i)-1 = J(j)$ for some j such that $u_j = \pi_{n-1}$, hence $\nu \models \mathrm{snf}(\alpha)_G^{J(j)}$, and by construction we know that $\alpha \in \Gamma(u_j)$, which conflicts with

the hypothesis that the YESTERDAY rule crossed the branch. With a similar argument, we can see that it cannot have been crossed by the W-YESTERDAY rule neither. Hence we found a branch which is either longer than $k+1$ step nodes, or have been accepted, or have been crossed by the PRUNE rule. □

The formula $|\phi|^k$ is called the *base encoding* of ϕ and, in addition to the k-unraveling, includes the encoding of the EMPTY and LOOP rules, *i.e.*, the rules that can accept branches. The formula is defined as:

$$|\phi|^k \equiv \langle\!\langle \phi \rangle\!\rangle^k \wedge (E_k \vee L_k)$$

where the E_k formula encodes the EMPTY rule and is defined as follows:

$$E_k \equiv \bigwedge_{\psi \in \mathsf{XR}} \neg \psi_G^k$$

and the L_k formula encodes the LOOP rule and is defined as:

$$L_k \equiv \bigvee_{l=0}^{k-1} ({}_lR_k \wedge {}_lF_k)$$

where

$${}_lR_k \equiv \bigwedge_{\psi \in \mathsf{XR} \cup \mathsf{YR} \cup \mathsf{ZR}} \left(\psi_G^l \leftrightarrow \psi_G^k \right)$$

$${}_lF_k \equiv \bigwedge_{\substack{\psi \in \mathsf{XEV} \\ \psi \equiv \mathsf{X}(\psi_1 \mathsf{U} \psi_2)}} \left(\psi_G^k \rightarrow \bigvee_{i=l+1}^{k} \mathrm{snf}(\psi_2)_G^i \right)$$

Intuitively, ${}_lR_k$ encodes the presence of two nodes whose labels contain the same requests for the next and the previous nodes, while ${}_lF_k$ checks that all the X-eventualities are fulfilled between those nodes. It can be proved that $|\phi|^k$ correctly encodes tableau trees where at least one branch is accepted in $k+1$ steps.

Lemma 14. *Let ϕ be an* LTL+P *formula. If the complete tableau for ϕ contains an accepted branch of $k+1$ step nodes, then $|\phi|^k$ is satisfiable.*

Proof. Suppose that the complete tableau for ϕ contains an accepted branch of $k+1$ step nodes, so let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be such a branch,

and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_k \rangle$ be the sequence of its step nodes. Then, by Lemma 13, $\langle\!\langle \phi \rangle\!\rangle^k$ is satisfiable. We can then build a truth assignment ν in the same way as in the proof of Lemma 13, such that $\nu \models \langle\!\langle \phi \rangle\!\rangle^k$. Remember that this means we set $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$ for all $0 \leq i \leq k$. So now we have to prove that ν satisfies either E_k or L_k. We will need an auxiliary fact, that is, that $\psi \in \Gamma^*(\pi_i)$ if and only if $\nu \models \mathrm{snf}(\psi)^i_G$. That can be done by induction on the structure of ψ, exploiting the definition of the expansion rules of the tableau.

Now, we distinguish two cases depending on which rule accepted the branch:

1. if the branch was accepted by the EMPTY rule, then $\Gamma(\pi_k) = \varnothing$, hence, in particular $\Gamma(\pi_k)$ does not contain any X-request. Hence by definition of ν we have that $\nu \models \neg\psi^k_G$ for any $\psi \in \mathsf{XR}$, so E_k is satisfied;

2. if the branch was accepted by the LOOP rule, then we have a node π_l such that $\Gamma(\pi_l) = \Gamma(\pi_k)$, hence by definition of ν we have $\nu \models \psi^l_G$ if and only if $\nu \models \psi^k_G$ for any $\psi \in \mathsf{XR} \cup \mathsf{YR} \cup \mathsf{ZR}$, so ${}_lR_k$ is satisfied. Moreover, we know that for any X-eventuality $\psi \equiv \mathsf{X}(\psi_1 \mathsf{U} \psi_2)$ requested in $\Gamma(\pi_k)$, ψ has been fulfilled between π_l and π_k, *i.e.*, there is a $l < j \leq k$ such that $\psi_2 \in \Gamma^*(\pi_j)$. Hence we know that $\nu \models \mathrm{snf}(\psi_2)^j_G$, hence ${}_lF_k$ is satisfied. Then, ${}_lR_k \wedge_l F_k$ is satisfied for at least one l, so L_k is satisfied.

□

Lemma 15. *Let ϕ be an* LTL+P *formula. If $|\phi|^k$ is satisfiable then the complete tableau for ϕ contains an accepted branch.*

Proof. Suppose that $|\phi|^k$ is satisfiable, hence we have a truth assignment ν such that $\nu \models |\phi|^k$. Then, $\langle\!\langle \phi \rangle\!\rangle^k$ is satisfiable, and we know from Lemma 13 that the complete tableau for ϕ has a branch that is either accepted, crossed by PRUNE, or longer than $k+1$ step nodes. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be the branch prefix found as shown in the proof of Lemma 13, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. By construction we have a function $J : \mathbb{N} \to \mathbb{N}$ fulfilling the invariant that $\psi \in \Gamma(u_i)$ if and only if $\nu \models \mathrm{snf}(\psi)^{J(i)}_G$. We now show that indeed $\overline{u}$ is accepted or is the prefix of an accepted branch. Since $|\phi|^k$ is satisfiable, either E_k or L_k are satisfiable as well:

1. if E_k is satisfiable, we know that $\nu \models \neg\psi_G^k$ for each $\psi \in \mathsf{XR}$. Since ψ is an X-request, $\mathrm{snf}(\psi) \equiv \psi$, so $\nu \not\models \mathrm{snf}(\psi)_G^k$. Here, $k = J(j)$ for some j, and from the invariant we know that $\psi \notin \Gamma(u_j)$. Hence, u_j does not contain any X-request, so its successor u_{j+1} has an empty label, triggering the EMPTY rule that accepts the branch.

2. if L_k is satisfiable, so are ${}_lR_k$ and ${}_lF_k$ for some $0 \leq l < k$. Hence from ${}_lR_k$ we know that $\nu \models \psi_G^l$ if and only if $\nu \models \psi_G^k$ for all $\psi \in \mathsf{XR} \cup \mathsf{YR} \cup \mathsf{ZR}$, that is $\nu \models \mathrm{snf}(\psi)_G^l$ if and only if $\nu \models \mathrm{snf}(\psi)_G^k$ because ψ is an X-, Y-, or Z-request. Here, $l = J(i)$ and $k = J(j)$ for some i and some j. Since the value of the function J increments at each step node, we can assume *w.l.o.g.* that u_i and u_j are step nodes, and by the invariant we know $\psi \in \Gamma(u_i)$ if and only if $\psi \in \Gamma(u_j)$, *i.e.*, u_i and u_j have the same X-, Y-, and Z-request. Similarly, the fact that $\nu \models {}_lF_k$ tells us that all the X-eventualities requested in u_i are fulfilled between u_{i+1} and u_j. The LOOP rule requires two identical labels in order to trigger, but u_i and u_j only have the same requests. However, since they have the same X-requests, we know that $\Gamma(u_{i+1}) = \Gamma(u_{j+1})$. Then, there is a step node $u_{j'}$, grandchild of u_j, such that $\Gamma(u_j) = \Gamma(u_{j'})$ and the segment of the branch between u_{i+1} and u_j is equal to the segment between u_{j+1} and $u_{j'}$, hence all the X-eventualities requested in u_i and u_j, fulfilled between u_{i+1} and u_j, are fulfilled between u_{j+1} and $u_{j'}$ as well, and the LOOP rule can apply to $u_{j'}$, accepting the branch.

□

Lastly, the formula $|\phi|_T^k$, called the *termination encoding*, encodes the PRUNE rule. The formula is defined as follows:

$$|\phi|_T^k \equiv \langle\!\langle \phi \rangle\!\rangle^k \wedge \bigwedge_{i=0}^{k} \neg P^i$$

where

$$P^k \equiv \bigvee_{l=0}^{k-2} \bigvee_{j=l+1}^{k-1} \left({}_lR_j \wedge {}_jR_k \wedge {}_lP_j^k \right)$$

$${}_lP_j^k \equiv \bigwedge_{\substack{\psi \in \mathsf{XEV} \\ \psi \equiv \mathsf{X}(\psi_1 \mathsf{U} \psi_2)}} \left(\psi_G^k \wedge \bigvee_{i=j+1}^{k} \mathrm{snf}(\psi_2)_G^i \rightarrow \bigvee_{i=l+1}^{j} \mathrm{snf}(\psi_2)_G^i \right)$$

It can be proved that $|\phi|_T^k$ is unsatisfiable if the tableau for ϕ contains only rejected branches.

Lemma 16. *Let ϕ be an* LTL+P *formula. If $|\phi|_T^k$ is unsatisfiable, then the complete tableau for ϕ contains only rejected branches.*

Proof. We prove the contrapositive, *i.e.*, that if the complete tableau for ϕ contains an accepted branch, then $|\phi|_T^k$ is satisfiable. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be such a branch, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. By Lemma 13, we know $\langle\!\langle \phi \rangle\!\rangle^k$ is satisfiable, thus we can obtain a truth assignment ν such that $\nu \models \langle\!\langle \phi \rangle\!\rangle^k$. We can build ν as in the proof of Lemma 13, *i.e.*, such that $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$ for all $0 \leq i \leq k$. Similarly to the proof of Lemma 14, we highlight the fact that $\psi \in \Gamma^*(\pi_i)$ if and only if $\nu \models \mathrm{snf}(\psi)_G^i$. Now, since the branch is accepted, the PRUNE rule cannot be applied to it. This means that either a) there are no three nodes π_u, π_v, π_w such that $\Gamma(\pi_u) = \Gamma(\pi_v) = \Gamma(\pi_w)$, or b) these three nodes exist but there is an X-eventuality ψ requested in $\Gamma(\pi_w)$ that is fulfilled between π_u and π_v and not between π_v and π_w. In case a) this means ${}_uR_v \wedge {}_vR_w$ does not hold for any u and v. In case b), ${}_uR_v \wedge {}_vR_w$ holds but ${}_uP_v^w$ does not. In any case, it follows that $\neg P^i$ holds for any $0 \leq i \leq k$, hence $|\phi|_T^k$ is satisfied. □

Together with the soundness and completeness results for the underlying tableau (Proposition 11), the above Lemmas allow us to prove the soundness and completeness of the procedure of Algorithm 1.

Theorem 35 (Soundness and completeness)**.** *Let ϕ be an* LTL+P *formula. The* BLACK *algorithm answers* satisfiable *on ϕ if and only if ϕ is satisfiable.*

Proof. ($\rightarrow$) Suppose the BLACK algorithm answers *satisfiable* on the formula ϕ. Then, it means there is a $k \geq 0$ such that $|\phi|^k$ is satisfiable. By Lemma 15, the complete tableau for ϕ has an accepting branch. By the soundness of the tableau, then ϕ is satisfiable.

($\leftarrow$) Now suppose the formula ϕ is satisfiable. By the completeness of the tableau, the complete tableau for ϕ has an accepting branch. Let us suppose such a branch has $k+1$ step nodes for some $k \geq 0$. Then, we want to show that the BLACK algorithm eventually answers *satisfiable*. Let $i < k$ be any earlier iteration of the main loop of the algorithm. We have that by Lemma 13, $\langle\!\langle \phi \rangle\!\rangle^i$ is satisfiable because there is a branch longer than $i+1$ step nodes. Similarly, by Lemma 16, $|\phi|^i_T$ is satisfiable because not all the branches of the tableau are rejected. Hence, the algorithm does not answer *unsatisfiable* at step i. Arrived at step k, $|\phi|^k$ is satisfiable by Lemma 14 because the tableau has an accepted branch of $k+1$ step nodes, hence the algorithm answers *satisfiable*. □

6.3 Extensions for finite traces and models extraction

In this section, we present two extensions of the encoding shown in Section 6.2, one for dealing with LTL+P interpreted over finite traces and the other allowing the extraction of a model in the case of satisfiable formulas, that works both for infinite and finite semantics.

6.3.1 Extension for LTL under finite traces

The encoding presented above can be very easily adapted to look for finite models of the formula. Since the tableau rules as recalled in Section 5.1.4 support infinite models, we now have to present the needed changes to Reynolds' tableau in order to support the search for finite models. Then, the corresponding changes to the encoding will follow easily.

With respect to the tableau rules as recalled in Section 5.1.4, we have to:

1. remove the LOOP rule, since we do not want to accept infinite periodic models anymore;

2. change the expansion rule for the *release* operator (see Table 5.1) in such a way that if $\phi \equiv \alpha \,\mathsf{R}\, \beta$, then $\Gamma_2(\phi) = \{\beta, \widetilde{\mathsf{X}}(\alpha \,\mathsf{R}\, \beta)\}$.

Moreover, we have to change the STEP and EMPTY rules as follows. Intuitively, the presence of only *weak-tomorrow* requests does not force the creation of a new state, since any formula of type $\widetilde{\mathsf{X}}\alpha$ is true at the last state of a finite model. Therefore, one the one hand, we change the STEP rule to propagate *weak tomorrow* requests as well, and on the other hand we change the EMPTY rule to accept nodes without *tomorrow* requests (but possibly with *weak-tomorrow* requests), instead of nodes with empty labels. So, if we have a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$, the rule is defined as follows:

STEP A child u_{n+1} is added to u_n, with:

$$\Gamma(u_{n+1}) = \{\alpha \mid \mathsf{X}\alpha \in \Gamma(u_n) \text{ or } \widetilde{\mathsf{X}}\alpha \in \Gamma(u_n)\}$$

EMPTY if $\Gamma(u_n)$ does not contain *tomorrow* formulas, then $\overline{u}$ is *accepted*.

Note that this variation of the EMPTY rule is equivalent to the original one for infinite models that we recalled in Section 5.1.4: in fact, since in the infinite case the *tomorrow* and the *weak-tomorrow* operators coincide, asking for a node with an empty label amounts to asking for a parent node with no *(weak-)tomorrow* formulas in its label. However, in the finite words setting, the *tomorrow* and the *weak-tomorrow* operators do *not* coincide anymore. In particular, any formula of type $\widetilde{\mathsf{X}}\alpha$ is true at the last state of a finite model, while any formula of type $\mathsf{X}\alpha$ is false in such a state. Interestingly, this difference does not affect the definition of the EMPTY rule. In fact, by asking for a label devoid of *tomorrow* formulas only, we are leaving open the possibility for last states (*i.e.*, leaves) to have labels containing any *weak-tomorrow* formula.

Hence, as far as the encoding is concerned, the only required changes are:

1. removing the L_k formula from the definition of $|\phi|^k$;

2. changing the definition of *stepped normal form* to reflect the change in the expansion rule of the *release* operator;

3. changing the definition of the *k-unraveling* to include *weak tomorrow* formula in the encoding of the STEP rule. To do this, we define $\widetilde{\mathsf{X}}\mathsf{R}$ as follows:

$$\widetilde{\mathsf{X}}\mathsf{R} = \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{weak tomorrow} \text{ formula}\}$$

Then, we change the S_k formula in the definition of $\langle\!\langle \phi \rangle\!\rangle^k$ as follows:

$$S_k \equiv \bigwedge_{\mathsf{X}\alpha \in \mathsf{XR}} \left((\mathsf{X}\alpha)^k_G \leftrightarrow \mathrm{snf}(\alpha)^{k+1}_G \right) \wedge \bigwedge_{\widetilde{\mathsf{X}}\alpha \in \widetilde{\mathsf{X}}\mathsf{R}} \left((\widetilde{\mathsf{X}}\alpha)^k_G \leftrightarrow \mathrm{snf}(\alpha)^{k+1}_G \right)$$

The encoding of the EMPTY rule, *i.e.*, the E_k formula in the definition of $|\phi|^k$, remains the same.

6.3.2 Extraction of models

The encoding and algorithm presented in this chapter can be used not only to decide the satisfiability of LTL+P formulas, but also to extract a model for satisfiable ones.

To do that, one has first of all to ask the SAT solver, from the assignment to $|\phi|^k$, the values of the propositions p^t, which tell the truth value of the proposition p for each time step t. Then, one has to extract the starting state of the loop of the periodic model identified by the LOOP rule, if any.

To do that, a few propositions $\ell_{l,k}$ are introduced, for some k and some $l < k$, and the formula L_k defined to build the base encoding $|\phi|^k$ is modified as follows:

$$L_k \equiv \bigwedge_{l=0}^{k-1} (\ell_{l,k} \leftrightarrow ({}_lR_k \wedge {}_lF_k)) \wedge \bigvee_{l=0}^{k-1} \ell_{l,k}$$

In this way, for a satisfiable formula, the first $\ell_{l,k}$ to be true will tell us that the loop starts at time $t = l$. If no such proposition is true, the branch was closed by the EMPTY rule and the model can be regarded as looping through its last state (if we are looking for an infinite model).

Then, another correction is necessary. A difference of this encoding w.r.t. the original tableau by Reynolds is that, while the LOOP rule asks for two nodes with the exact same label, the encoding of the rule only looks for the same X-requests. From the proof of

Lemma 15, it can be seen that this change does not compromise correctness, and is even an advantage, since it leads the encoding to stop earlier. However, when one wants to extract a model from the assignment of $|\phi|^k$, if the loop has been identified from state l to k, one has to take care of past requests ($\mathsf{Y}\alpha$) coming from state $l+1$, that are for sure satisfied by state l, but have to be satisfied by state k as well. The original LOOP rule does take care of this detail by requiring the whole label to be equal. Here, where we only look for the same X-requests, we have to take care of it explicitly. The formula ${}_lR_k$ then becomes:

$$
\begin{aligned}
{}_lR_k \equiv & \bigwedge_{\psi \in \mathsf{XR} \cup \mathsf{YR} \cup \mathsf{ZR}} \left(\psi_G^l \leftrightarrow \psi_G^k\right) \wedge \\
& \bigwedge_{\mathsf{Y}\alpha \in \mathsf{YR}} (\mathsf{Y}\alpha)_G^{l+1} \leftrightarrow \mathrm{snf}(\alpha)_G^k \wedge \\
& \bigwedge_{\mathsf{Z}\alpha \in \mathsf{YR}} (\mathsf{Z}\alpha)_G^{l+1} \leftrightarrow \mathrm{snf}(\alpha)_G^k
\end{aligned}
$$

6.4 Experimental evaluation

In this section, we describe the experimental evaluation of BLACK against other state-of-the-art tools for the satisfiability of LTL and LTL+P formulas under both infinite and finite trace semantics. Benchmarks consist in a set of input formulas (see later for a detailed account) over which the different tools have been run to measure solving speed. All the benchmarks have been run on a 16-core AMD EPYC 7281 processor with 64GB of RAM, with a timeout of five minutes and a memory limit of 3GB for each formula. All the benchmark formulas and the supporting scripts are available in BLACK's source code repository. In all the tests described here, BLACK has been run using the MathSAT backend [54], which in internal tests appeared to perform better than the other backends, overall.

6.4.1 LTL over infinite traces

To evaluate BLACK's performance on LTL over infinite traces, we compared it with the following tools:

1. the NUXMV [38] model checker, both in SBMC and K-Liveness

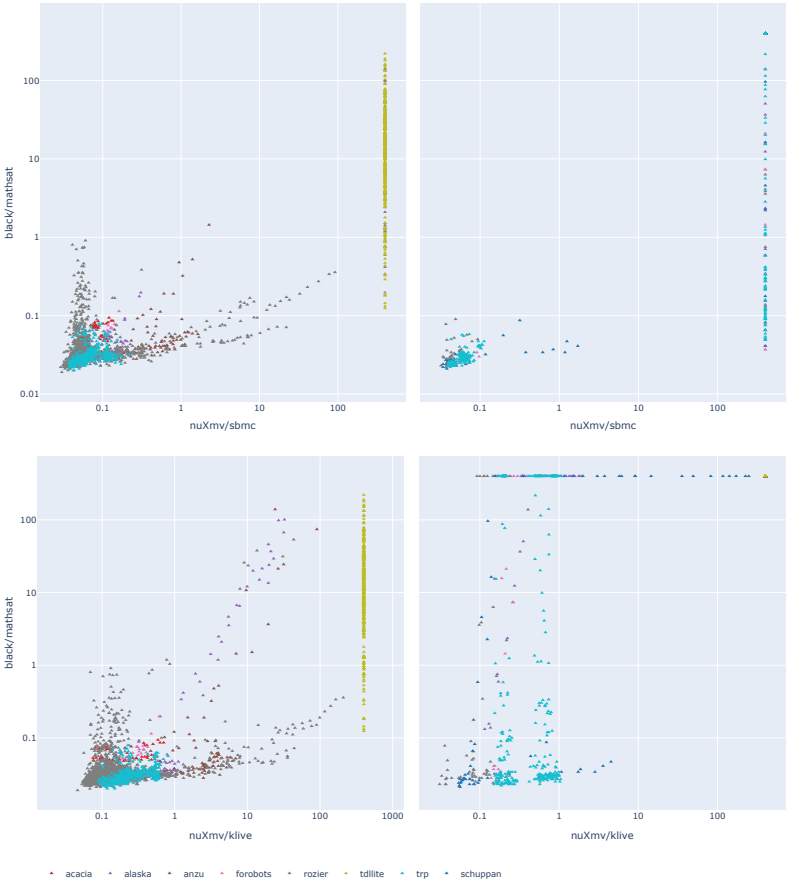


Figure 6.1: Scatter plots of BLACK versus NUXMV in SBMC (top) and K-live (bottom) over LTL on infinite traces. SAT instances on the left, UNSAT instances on the right.

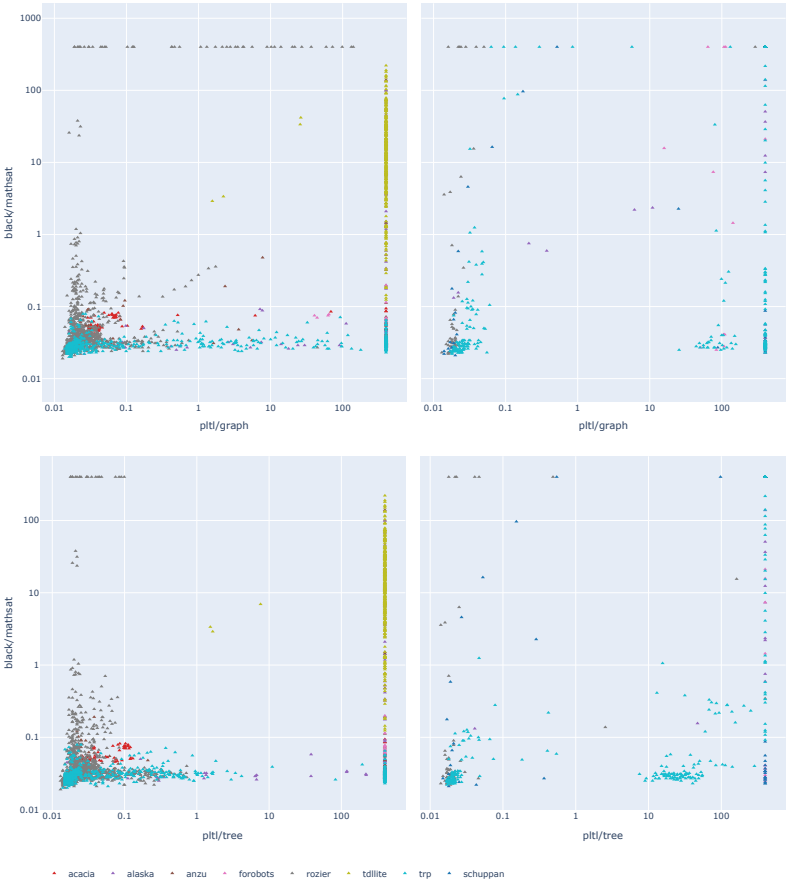


Figure 6.2: Scatter plots of BLACK versus pltl in graph (top) and tree (bottom) over LTL on infinite traces. SAT instances on the left, UNSAT instances on the right.

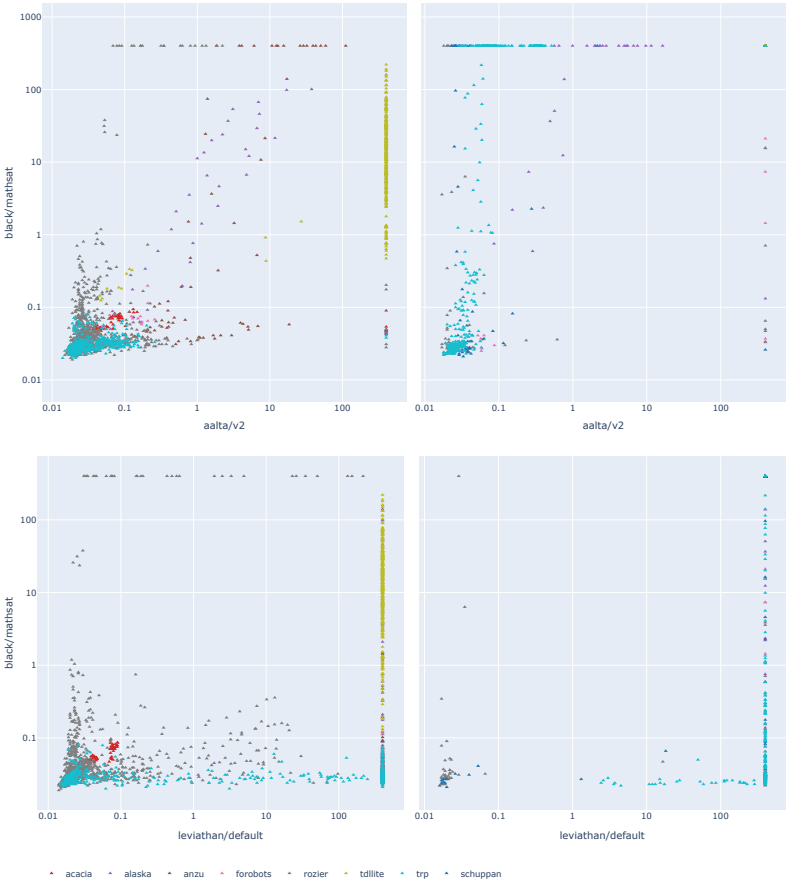


Figure 6.3: Scatter plots of BLACK versus Aalta (top) and Leviathan (bottom) over LTL on infinite traces. SAT instances on the left, UNSAT instances on the right.

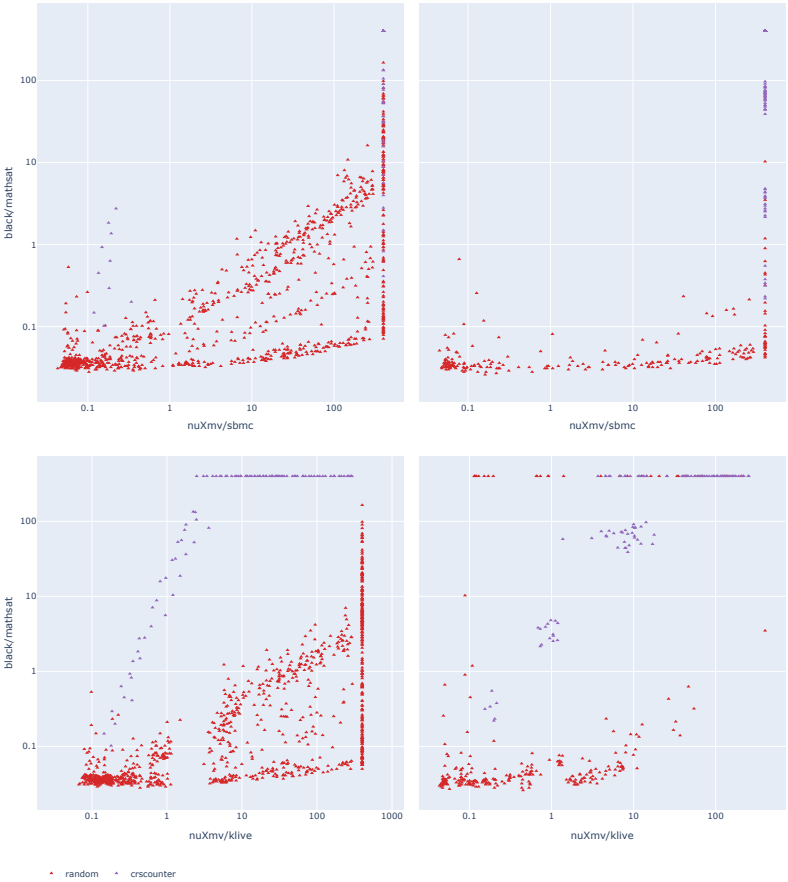


Figure 6.4: Scatter plots of BLACK versus NUXMV in SBMC (top) and K-Live (bottom) modes, over LTL+P formulas. SAT instances on the left, UNSAT instances on the right.

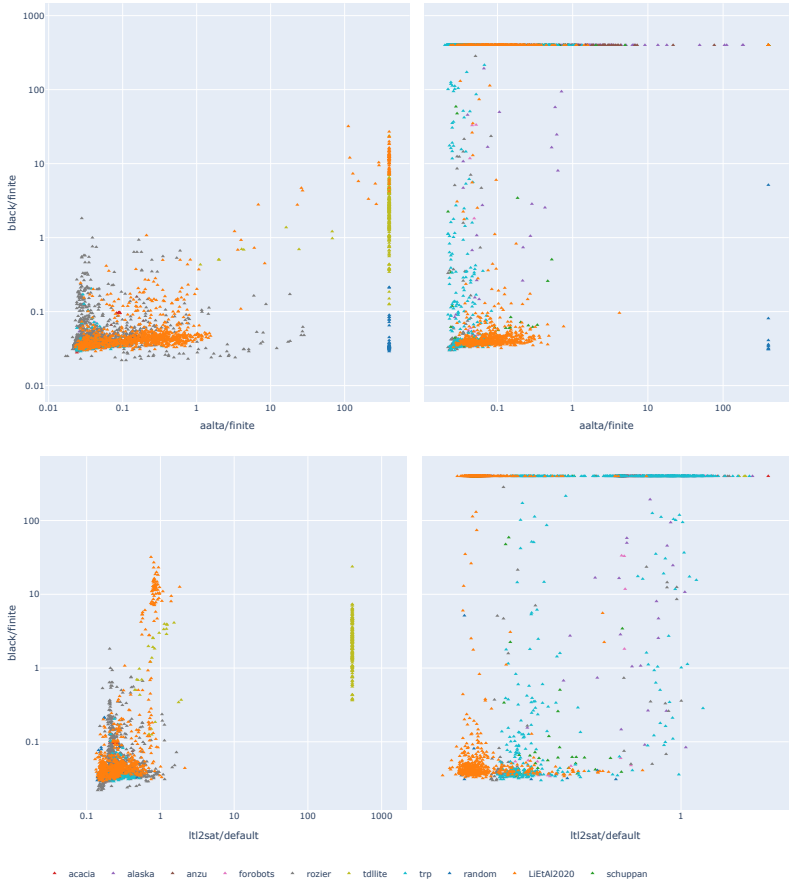


Figure 6.5: Scatter plots of BLACK versus Aaltaf (top) and LTL2SAT (bottom) over LTL on finite traces. SAT instances on the left, UNSAT instances on the right.

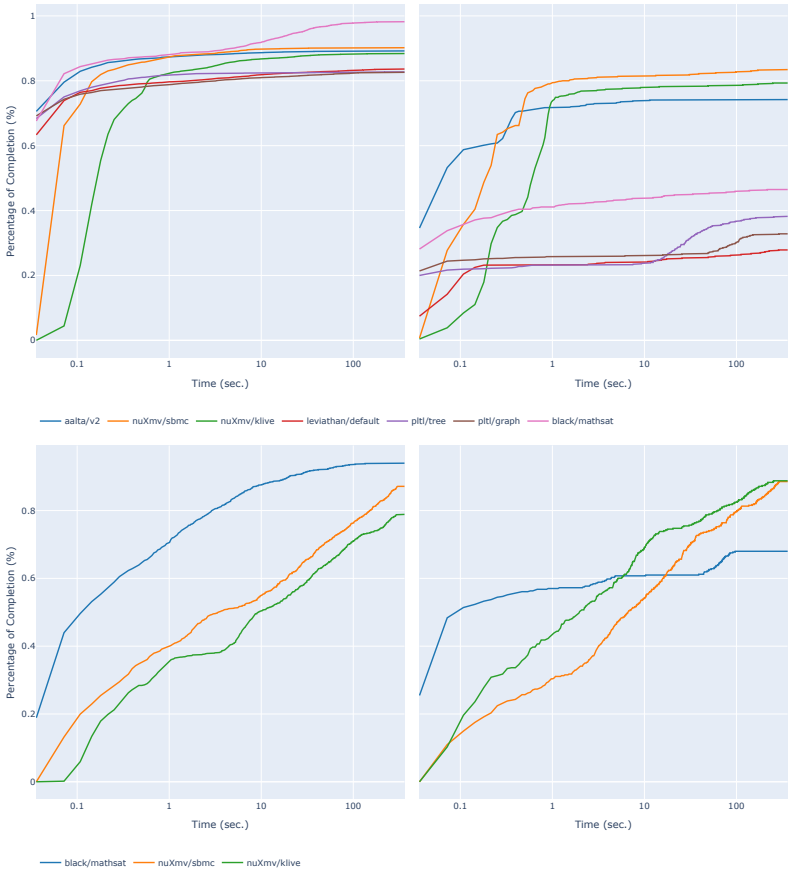


Figure 6.6: Survival plot for LTL over infinite traces (top row) and LTL+P (bottom row). SAT instances on the left, UNSAT instances on the right.

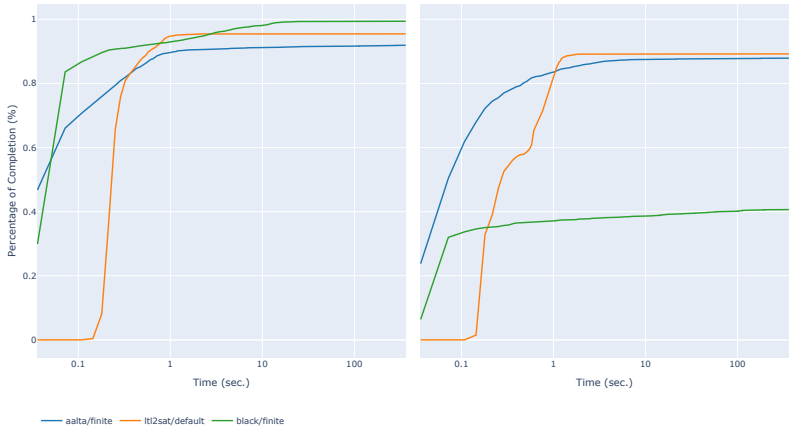


Figure 6.7: Survival plot for LTL over finite traces. SAT instances on the left, UNSAT instances on the right.

modes; we used this model checker for implementing the reduction from LTL+P satisfiability to LTL+P model checking described in Section 5.2.7.

2. Aalta [133], a tool based on an explicit graph-shaped tableau built with the help of a SAT solver;

3. pltl, a tableau-based tool that implements both a graph-shaped tableau [1] and a tree-shaped tableau *à la* Schwendimann [172];

4. Leviathan [13], an implementation of the explicit construction of Reynolds' tableau [163].

Of these, nuXmv in SBMC mode and Leviathan are the most similar to BLACK, in different ways. The former implements an iterative model checking procedure that looks for counter-examples of the specification of length at most k for increasing values of k, with a completeness check that ensures termination for unsatisfiable formulas [112]. The latter is the first implementation of Reynolds' tableau, which is the tableau underlying BLACK's SAT encoding and algorithm, but which Leviathan constructs explicitly.

These tools have been tested on a total of 4181 formulas which have been collected from two different sources:

1. the formulas collected by Schuppan and Darmawan [171], which include the `acacia`, `alaska`, `anzu`, `forobots`, `rozier`, `trp` and `schuppan` sets;

2. a set of formulas coming from the LTL encoding of the temporal description logic TDL-lite, as described by Tahrat *et al.* [187], which are referred to in the plots as the `tdllite` set.

The results are shown as scatter plots reported in Figs. 6.1, 6.2 and 6.3 and the survival plot reported in Fig. 6.7. In the comparison with nuXmv, it can be seen that BLACK outperforms it in SBMC mode, which is the most similar approach to BLACK. With respect to nuXmv in K-Liveness mode, BLACK performs better over satisfiable instances and suffers over unsatisfiable instances. This is a pattern that repeats also in the comparison with Aalta. This suffering in unsatisfiable instances can be explained with the cubic growth in size of the termination encoding $|\phi|_T^k$ at increasing values of k. On the other hand, in the comparison with the tableau-based tools Leviathan and pltl, it can be seen that BLACK performs considerably better both on satisfiable and unsatisfiable instances. It is worth to note that BLACK is the only tool that manages to solve the formulas in the `tdllite` set.

6.4.2 LTL+P

To evaluate BLACK over LTL+P formulas, we compared it with nuXmv [38] which, as far as we know, is the only other tool available that directly supports past operators. As there are no readily available benchmark sets for LTL+P in the literature, we came up with our own. The benchmark formulas consist in two sets:

1. the `random` set, which consists in randomly generated formulas of varying size, generated with an extension of the algorithm proposed by Tauriainen and Heljanko [188];

2. the `crscounter` set, which is an adaptation to the satisfiability problem of a benchmark set for model checking provided by Cimatti *et al.* [56].

The second family, in particular, needs some explanations. In the original benchmark set for model checking, a Kripke structure called $Counter(N)$, where N is a power of two, is introduced. $Counter(N)$

works as follows: it starts at $c = 0$, counts up to $c = N$, jumps back to $c = N/2$, and then loops, counting up to $c = N$ and jumping back to $c = N/2$, forever. Afterwards, they evaluated, on top of that Kripke structure, some parametric properties of the form:

$$P(i) \equiv \neg \mathsf{F}\big(\mathsf{O}((c = \tfrac{N}{2}) \wedge \mathsf{O}((c = \tfrac{N}{2} + 1) \wedge \ldots \wedge \mathsf{O}(c = \tfrac{N}{2} + i)\ldots))\big).$$

The value i identifies the number of nested *once* operators, while the structure of such properties requires that the loop of length $N/2$ in the model is traversed backwards several times in order to reach a counterexample.

Since these benchmarks were introduced in the context of model checking, we made a reduction from the model checking problem to the satisfiability checking one for LTL+P: we built the LTL+P formulas $\phi_{Counter(N)}$ and $\phi_{P(i)}$ encoding the above elements. In this way, $\neg(\phi_{Counter(N)} \rightarrow \phi_{P(i)})$ is UNSAT if and only if $Counter(N) \models P(i)$. With this framework, we were able to obtain both SAT ($i \leq \frac{N}{2}$) and UNSAT ($i > \frac{N}{2}$) instances. Moreover, this family of formulas stresses the ability to process past operators and find short counterexamples, and thus, it specifically challenges BLACK's performance.

The results are shown in the scatter plots of Fig. 6.4, and the survival plot of Fig. 6.7. As it can be seen, again BLACK outperforms NUXMV in SBMC mode, and is competitive with NUXMV in K-Liveness mode. The performance gain is more visible in satisfiable instances, as in the future-only formulas of the previous sections. In general, BLACK's performance is better on random formulas.

6.4.3 LTL over finite traces

We compared BLACK against two state-of-the-art tools for LTL over finite traces, namely Aaltaf [132], which implements an algorithm called *Conflict-Driven* LTL *under finite traces Satisfiability Checking* where a SAT-aided explicit tableau construction is paired with the extraction of unsatisfiable cores to prune the search space, and LTL2SAT [93], a tool which employs a SAT-based reduction but with specific heuristics for particular classes of formulas.

We compared the tools against the following sets of formulas:

1. all the formula sets from the infinite traces case as described above, but interpreted over finite traces;

2. a set of formulas particularly crafted for finite traces, referred to as `LiEtAl2020` in the plots, taken from Li *et al.* [132].

The results are shown in the scatter plots of Fig. 6.5, and the survival plot of Fig. 6.7. The finite traces setting is the one where BLACK suffers the most. As usual, the performance is better on satisfiable instances. In this case, BLACK performs better overall against Aaltaf than against LTL2SAT. As in the infinite trace setting, it is worth to note that only BLACK is capable of solving the formulas from the `tdllite` set.

6.5 Conclusions

In this chapter, we present BLACK, a new satisfiability checking tool for LTL and LTL+P under both infinite and finite traces semantics, based on a SAT encoding of the one-pass tree-shaped tableau by Reynolds [163]. Regarding its performance, extensive experimental evaluations show that BLACK, is competitive with other state-of-the-art tools in most circumstances, especially on satisfiable instances.

Many future developments are possible. From the point of view of performance, it would be interesting to find heuristics to optimize the application of the PRUNE rule of the tableau in order to speed up the execution on unsatisfiable instances. In particular, a linear-size encoding would arguably provide a great speed-up in such cases.

Reynolds' tableau has been extended to other logics beyond LTL+P, such as TPTL. A similar SAT (or SMT) encoding for the TPTL tableau would allow BLACK to support this real-time logic as well. Similar extensions are being investigated for first-order extensions of LTL+P, based on the work by Kontchakov *et al.* [124].

As far as the tool itself is concerned, many improvements are possible, such as the support of more SAT backends, the improvement of the efficiency of the current CNF translation, and the integration of more input and/or output formats. Moreover, the modular structure of the tool is not at all tied to the specific algorithm and encoding presented here, hence the implementation of different satisfiability checking approaches is possible and would provide a nice portfolio-based solver able to cope with even more application scenarios.

CHAPTER

7

REALIZABILITY OF $\mathsf{LTL}_{\mathsf{EBR}}$+P SPECIFICATIONS

Reactive synthesis is a key technique for the design of correct-by-construction systems, which has been thoroughly investigated in the last decades. In Section 5.3, we have seen that it consists of the synthesis of a controller that reacts to environment's inputs satisfying a given temporal logic specification. Common approaches are based on the explicit construction of automata and on their determinization, which limits their scalability.

In Section 3.1, we introduced $\mathsf{LTL}_{\mathsf{EBR}}$+P (Extended Bounded Response LTL+P), a safety fragment of LTL+P that captures exactly the safety properties definable in LTL+P.

In this chapter, we focus on realizability from $\mathsf{LTL}_{\mathsf{EBR}}$+P specifications. We show that reactive synthesis from $\mathsf{LTL}_{\mathsf{EBR}}$+P specifications can be reduced to solving a safety game over a deterministic

symbolic automaton built directly from the specification. We prove the correctness of the approach and study the complexity of the fragment showing that (i) it is singly exponential in time (in contrast to that of LTL, which is doubly exponential in time); and that (ii) the proposed solution is optimal. Finally, we evaluate it on various benchmarks showing better performance of other approaches for general LTL or larger safety fragments.

7.1 Overview

In the previous chapters, we have seen that, since the dawn of computer science, synthesizing correct-by-construction systems starting from a specification is recognized as an important and difficult task. A practical algorithm to solve this task would be a big improvement in declarative programming, since it would allow the programmer to write only the specification of the program, freeing her from possible design or implementation bugs, which, in many cases, are due to an imperative style of programming. In the context of model-based design and formal verification, the possibility of synthesizing a controller that complies with the specification, for all possible behaviors of the environment, would be of great importance as well, as all the design effort would be directed to improve the quality of the specification of the controller.

In this chapter, we focus on the *realizability* and *reactive sythesis* problems for $\mathsf{LTL}_{\mathsf{EBR}}$+P (Section 3.1). First, we show that formulas of $\mathsf{LTL}_{\mathsf{EBR}}$+P can be turned into *deterministic symbolic automata* over infinite words (Definitions 37 and 40) by means of a translation carried out in a completely symbolic way. Such a transformation is performed in two steps: (i) a *pastification* of the *bounded future* subformulas by making use of techniques similar to those exploited for MTL [139, 138], and (ii) the construction of *deterministic monitors* for the universal temporal operators as well as for the past modalities of $\mathsf{LTL}_{\mathsf{EBR}}$+P. These two steps allow the entire procedure to be carried out without ever producing any explicit automaton. Then, we use existing algorithms for safety synthesis to solve the game on the deterministic symbolic automaton. Finally, we prove that the entire procedure works in EXPTIME.

Next, we focus on complexity issues for $\mathsf{LTL}_{\mathsf{EBR}}$+P by showing that (i) its satisfiability problem is PSPACE-hard, and (ii) its real-

izability problem is EXPTIME-complete. In the proof of the first hardness result, we make use of a reduction from the *corridor tiling problem*, while in the proof of the second one, we exploit a reduction from the *corridor tiling game* (EXPTIME-membership immediately follows from our algorithm). Our complexity result for $\mathsf{LTL}_{\mathsf{EBR}}$+P realizability shows that this problem has a lower worst-case complexity than realizability from LTL (which is 2EXPTIME-complete). Moreover, it follows that our algorithm is *optimal* with respect to the theoretical complexity of $\mathsf{LTL}_{\mathsf{EBR}}$+P realizability.

Finally, we implement the proposed solution in a tool, called EBR-LTL-SYNTH, and compare its performance against state-of-the-art synthesizers for full LTL and for Safety-LTL over a set of $\mathsf{LTL}_{\mathsf{EBR}}$+P formulas. Despite the high (EXPTIME) complexity, the outcomes of the experimental evaluation are quite encouraging and many cases can be solved efficiently.

The rest of the chapter is organized as follows. In Section 7.2, we define the compilation of $\mathsf{LTL}_{\mathsf{EBR}}$+P formulas into symbolic safety automata; then, in Section 7.3, we show how to exploit the resulting automata to solve the realizability and synthesis problems for $\mathsf{LTL}_{\mathsf{EBR}}$+P and we contrast the devised solution with existing algorithms, pointing out the main differences. In addition, we study the complexity of the realizability and satisfiability checking problems for $\mathsf{LTL}_{\mathsf{EBR}}$. We illustrate and discuss the outcomes of the experimental evaluation in Section 7.4. In Section 7.5, we provide an assessment of the work, and outline some research directions. The proof of all the results are reported either in the main body of this chapter or in Section A.1.

7.2 From $\mathsf{LTL}_{\mathsf{EBR}}$+P to deterministic SSA

This section outlines a procedure to turn every $\mathsf{LTL}_{\mathsf{EBR}}$+P formula into a deterministic symbolic safety automaton (SSA) on infinite words (see Definitions 37 and 40) that recognizes the same language.

We recall from Section 3.1 that any $\mathsf{LTL}_{\mathsf{EBR}}$+P formula χ belongs

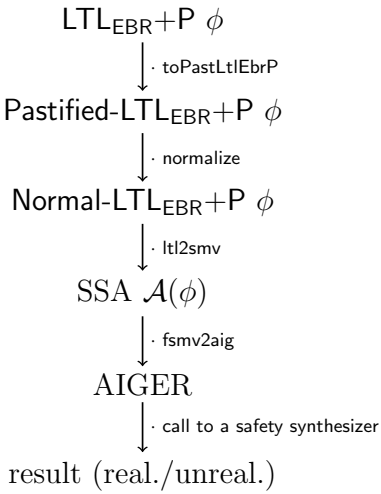


Figure 7.1: The overall procedure.

to the following syntax:

$$\eta := p \mid \neg\eta \mid \eta_1 \vee \eta_2 \mid \mathsf{Y}\eta \mid \eta_1 \,\mathsf{S}\, \eta_2 \qquad \text{Pure Past Layer}$$

$$\psi := \eta \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{X}\psi \mid \psi_1 \,\mathsf{U}^{[a,b]}\, \psi_2 \qquad \text{Bounded Future Layer}$$

$$\phi := \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid \psi \,\mathsf{R}\, \phi \qquad \text{Future Layer}$$

$$\chi := \phi \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2 \qquad \text{Boolean Layer}$$

$\mathsf{LTL{+}P_{BF}}$ (*bounded future* LTL+P) is the logic obtained from the grammar of $\mathsf{LTL_{EBR}}$+P by starting from the Bounded Future Layer, and similarly $\mathsf{LTL{+}P_P}$ (*pure past* LTL+P) is the logic obtained by considering only the pure past layer. Finally, we recall that we consider bounded operators as *shortcuts* for the equivalent expansion using the *next* operator (*e.g.*, $\mathsf{F}^{[a,b]}\phi \equiv \bigvee_{i=a}^{b} \mathsf{X}^i\phi$, where $\mathsf{X}^i = \mathsf{X}_{(1)} \ldots \mathsf{X}_{(i)}$). In particular, this means that, when considering the *dimension* $|\phi|$ (with ϕ either a $\mathsf{LTL{+}P_{BF}}$ or an $\mathsf{LTL_{EBR}}$+P formula), we are referring to the size of the formula where all the bounded operators have been expanded.

In doing the main transformation from $\mathsf{LTL_{EBR}}$+P formulas to language-equivalent deterministic symbolic safety automata, we apply a few transformation steps on the formula, summarized in Fig. 7.1, to simplify its syntactic structure and turn it into a form amenable

to direct transformation into a deterministic SSA. We define two syntactic restrictions of $\mathsf{LTL_{EBR}}$+P that are the targets of the transformation steps.

Definition 46 (Pastified-$\mathsf{LTL_{EBR}}$+P)**.** *A* Pastified-$\mathsf{LTL_{EBR}}$+P *formula χ is inductively defined as follows:*

$$\psi := p \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{Y}\psi \mid \psi_1 \,\mathsf{S}\, \psi_2$$
$$\phi := \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid (\mathsf{X}^i\psi)\,\mathsf{R}\,\phi$$
$$\chi := \phi \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2$$

Definition 47 (Normal-$\mathsf{LTL_{EBR}}$+P)**.** *The* normal form *of* $\mathsf{LTL_{EBR}}$+P *formulas is inductively defined as follows:*

$$\psi := p \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{Y}\psi \mid \psi_1 \,\mathsf{S}\, \psi_2$$
$$\phi := \psi \mid \mathsf{G}\psi \mid \psi_1 \,\mathsf{R}\, \psi_2$$
$$\lambda := \phi \mid \mathsf{X}\lambda$$
$$\chi := \lambda \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2$$

Normal-$\mathsf{LTL_{EBR}}$+P formulas do not contain nested occurrences of unbounded temporal operators, whose operands can be only pure past formulas, and each of these is prefixed by an arbitrary number of *next* operators.

The transformation of $\mathsf{LTL_{EBR}}$+P formulas into deterministic SSAs consists of three steps, which are depicted in Fig. 7.1: (i) a translation from $\mathsf{LTL_{EBR}}$+P to Pastified-$\mathsf{LTL_{EBR}}$+P; (ii) a translation from Pastified-$\mathsf{LTL_{EBR}}$+P to its normal form; (iii) a transformation of Normal-$\mathsf{LTL_{EBR}}$+P formulas into deterministic SSA. Once a deterministic SSA $\mathcal{A}(\phi)$ for the original $\mathsf{LTL_{EBR}}$+P formula ϕ over $\mathcal{C} \cup \mathcal{U}$ has been obtained, we solve the safety game $\langle \mathcal{A}(\phi), \mathcal{C}, \mathcal{U} \rangle$, *i.e.*, we check the existence of a winning strategy for Controller in the automaton, by applying an existing safety synthesis algorithms (see Section 5.3.4).

7.2.1 From $\mathsf{LTL_{EBR}}$+P to Pastified-$\mathsf{LTL_{EBR}}$+P

Let ϕ be an $\mathsf{LTL_{EBR}}$+P formula. The first step consists in translating each $\mathsf{LTL}{+}\mathsf{P_{BF}}$ subformula of ϕ into an *equivalent* one, which is of the form $X^d\psi$, with $\psi \in \mathsf{LTL}{+}\mathsf{P_P}$ and $d \in \mathbb{N}$. We refer to this process as *pastification* [139, 138]. As we will see, since "the past has already

happened", pure past formulas can be represented by deterministic monitors.

In order to pastify each LTL+$\mathsf{P_{BF}}$ subformula of ϕ, we adapt to $\mathsf{LTL_{EBR}}$+P a technique developed by Maler *et al.* for $\mathsf{MTL}-\mathsf{B}$ [139, 138]. Intuitively, for each model of a LTL+$\mathsf{P_{BF}}$ formula ϕ, there exists a furthermost time point d (the *temporal depth* of ϕ) such that the subsequent states cannot be constrained by ϕ in any way. The *pastification* of ϕ is a formula that uses only past operators and that is equivalent to ϕ when interpreted at time point d instead of at the origin.

We now define the *temporal depth* and the *pastification* of a formula. Since we consider bounded operators as shortcuts, in those definitions it would suffice to consider only the cases for atomic propositions, Boolean operators, and the *next* temporal operator. However, for sake of completeness, we include also the case for the *bounded until* operator (in gray color).

Definition 48 (Temporal Depth [139])**.** *Let ϕ be an* LTL+$\mathsf{P_{BF}}$ *formula. The* temporal depth *of ϕ, denoted as $D(\phi)$, is inductively defined as follows:*

- $D(\psi) = 0$*, for all* $\psi \in \mathsf{LTL}$+$\mathsf{P_P}$
- $D(\neg\phi_1) = D(\phi_1)$
- $D(\phi_1 \vee \phi_2) = \max\{D(\phi_1), D(\phi_2)\}$
- $D(\mathsf{X}\phi_1) = 1 + D(\phi_1)$
- $D(\phi_1 \,\mathsf{U}^{[a,b]}\, \phi_2) = b + \max\{D(\phi_1), D(\phi_2)\}$

It is easy to see that, for any LTL+$\mathsf{P_{BF}}$ formula ϕ, it holds that $D(\phi) \leq |\phi|$.

Definition 49 (Pastification [139])**.** *Let ϕ be an* LTL+$\mathsf{P_{BF}}$ *formula and $d \geq D(\phi)$. The* pastification of ϕ at d *is the formula* $\Pi(\phi, d)$ *inductively defined as follows:*

- $\Pi(\psi, d) = \mathsf{Y}^d\psi$*, where* $\psi \in \mathsf{LTL}$+$\mathsf{P_P}$
- $\Pi(\neg\phi, d) = \neg\Pi(\phi, d)$
- $\Pi(\phi_1 \vee \phi_2, d) = \Pi(\phi_1, d) \wedge \Pi(\phi_2, d)$
- $\Pi(\mathsf{X}\phi, d) = \Pi(\phi, d-1)$

- $\Pi(\phi_1 \mathsf{U}^{[a,b]} \phi_2, d) = \bigvee_{t=0}^{b-a} (\mathsf{Y}^t(\Pi(\phi_2, d-b) \wedge \mathsf{H}^{b-t-1} \mathsf{Y} \Pi(\phi_1, d-b)))$

Note that from Definition 49 we can derive that $\Pi(\mathsf{F}^{[a,b]}\phi, d) \equiv \Pi(\top\, \mathsf{U}^{[a,b]}\, \phi, d) \equiv \bigvee_{t=0}^{b-a} \mathsf{Y}^t \Pi(\phi, d-b)$.

Proposition 15 (Soundness of pastification)**.** *Let φ be a $\mathsf{LTL}{+}\mathsf{P_{BF}}$ formula. For all state sequences $\sigma \in (2^\Sigma)^\omega$, all $i \in \mathbb{N}$, and all $d \geq D(\phi)$, it holds that:*

$$\sigma, i \models \varphi \;\Leftrightarrow\; \sigma, i \models \mathsf{X}^d \Pi(\varphi, d)$$

Proof. See Section A.1. □

Our definition of pastification differs from the original one [139, 138] in two directions. Firstly, the input of our pastification can be an arbitrary $\mathsf{LTL}{+}\mathsf{P_{BF}}$ formula, that is, one belonging to the two last layers of Definition 21. In constrast, in [139], the input formulas of the pastification algorithm are *pure future* formulas with only bounded operators, which are a special case of $\mathsf{LTL}{+}\mathsf{P_{BF}}$ formulas (they correspond to $\mathsf{LTL}{+}\mathsf{P_{BF}}$ formulas without *past* operators). We can deal with general $\mathsf{LTL}{+}\mathsf{P_{BF}}$ formulas by noting that $\mathsf{LTL}{+}\mathsf{P_P}$ subformulas do *not* need to be pastified (see the base case of Definition 49).

Secondly, in [139, 138] the pastification of a *bounded until* operator produces a novel temporal operator $\mathcal{P}^{[a,b]}$, called *bounded precedes*, which allows the output formula to be linear in size with respect to the input one even when the bounded operators are considered as primitives, in particular: $\Pi(\phi_1\, \mathsf{U}^{[a,b]}\, \phi_2, d) = \Pi(\phi_1, d-b)\mathcal{P}^{[a,b]}\Pi(\phi_2, d-b)$. On the contrary, since we consider the *bounded until* as a shortcut, we pastify the *bounded until* (and also the *bounded eventually*) operator by using only primitive *past* operators, like *yesterday* and *historically*, but still maintaining linear the size of the pastified formula.

From now on, let $\mathsf{pastify}(\phi)$ be the formula $\mathsf{X}^{D(\phi)}\Pi(\phi, D(\phi))$, that is the *pastification of ϕ at the temporal depth of ϕ*, prefixed by as many nested *next* operators as its temporal depth. As an example, if $\phi \coloneqq \mathsf{F}^{[0,k_1]}(q \wedge \mathsf{F}^{[0,k_2]}p)$, then $\mathsf{pastify}(\phi) \coloneqq \mathsf{X}^{k_1+k_2} \bigvee_{i=0}^{k_1} \mathsf{Y}^i(\mathsf{Y}^{k_2}q \wedge \bigvee_{j=0}^{k_2} \mathsf{Y}^j p)$. We state the following complexity result about pastification.

Proposition 16 (Size of pastification)**.** *Let ϕ be a $\mathsf{LTL}{+}\mathsf{P_{BF}}$ formula. Then, $\mathsf{pastify}(\phi)$ is a formula of size $\mathcal{O}(n)$, where $n = |\phi|$.*

Given an $\mathsf{LTL_{EBR}}$+P formula ϕ, we pastify each of its LTL+$\mathsf{P_{BF}}$ subformulas with the pastify operator: we call this step toPastLtlEbrP. Once it has been completed, the resulting formula belongs to Pastified-$\mathsf{LTL_{EBR}}$+P (see Definition 46).

Optimization

The toPastLtlEbrP algorithm can be improved by observing that there are LTL+$\mathsf{P_{BF}}$ formulas that already belong to Pastified-$\mathsf{LTL_{EBR}}$+P. One example is the formula $p \wedge \mathsf{XXX}q$. Obviously, for this kind of formulas there is no need for the algorithm to pastify them. Consider the previous example. Without the proposed trick, the algorithm would have produced the formula $\mathsf{XXX}(\mathsf{YYY}p \wedge q)$, while, by simply noting that the formula already belongs to Pastified-$\mathsf{LTL_{EBR}}$+P, it does not need to pastify anything, returning $p \wedge \mathsf{XXX}q$.

Proposition 17. *For each* $\mathsf{LTL_{EBR}}$+P *formula* ϕ*, there is an equivalent* Pastified-$\mathsf{LTL_{EBR}}$+P *formula* ϕ' *of size* $\mathcal{O}(n)$*, where* $n = |\phi|$*.*

Proof. Since the pastify operator replaces any subformula of size n with another one of size $\mathcal{O}(n)$, the size of $\mathsf{pastify}(\phi)$ is again $\mathcal{O}(n)$. Moreover $\mathsf{pastify}(\phi) \in$ Pastified-$\mathsf{LTL_{EBR}}$+P. □

7.2.2 From Pastified-$\mathsf{LTL_{EBR}}$+P to Normal-$\mathsf{LTL_{EBR}}$+P

The second step is the normalization of the Pastified-$\mathsf{LTL_{EBR}}$+P formula obtained from the previous step, in order to produce an equivalent formula in *normal form* (Definition 47). Normal-$\mathsf{LTL_{EBR}}$+P formulas are Boolean combinations of formulas of the form $\mathsf{X}^i\psi_1$, $\mathsf{X}^i\mathsf{G}\psi_1$, and $\mathsf{X}^i(\psi_1 \,\mathsf{R}\, \psi_2)$, where ψ_1 and ψ_2 are pure past formulas. Compared to general Pastified-$\mathsf{LTL_{EBR}}$+P formulas, those in normal form do not admit neither nested unbounded operators nor *next* operators in front of the left-hand argument of a *release*.

The algorithm computing the normalization of a Pastified-$\mathsf{LTL_{EBR}}$+P formula works by applying the following set of rewriting rules on the

formula obtained from the step described in the previous subsection:

$$R_1 : \mathsf{X}^i(\psi_1 \wedge \psi_2) \rightsquigarrow \mathsf{X}^i\psi_1 \wedge \mathsf{X}^i\psi_2$$

$$R_2 : \psi \,\mathsf{R}\, (\psi_1 \wedge \psi_2) \rightsquigarrow \psi \,\mathsf{R}\, \psi_1 \wedge \psi \,\mathsf{R}\, \psi_2$$

$$R_3 : (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\psi_2) \rightsquigarrow \begin{cases} \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2)) & \text{if } i > j \\ \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathsf{R}\, \psi_2) & \text{otherwise} \end{cases}$$

$$R_4 : (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j(\psi_2 \,\mathsf{R}\, \psi_3)) \rightsquigarrow \begin{cases} \mathsf{X}^i(\psi_1 \,\mathsf{R}\, ((\mathsf{Y}^{i-j}\psi_2) \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_3))) & \text{if } i > j \\ \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathsf{R}\, (\psi_2 \,\mathsf{R}\, \psi_3)) & \text{otherwise} \end{cases}$$

$$R_5 : \mathsf{GX}^i\mathsf{G}\psi \rightsquigarrow \mathsf{X}^i\mathsf{G}\psi$$

$$R_6 : \mathsf{GX}^i(\psi_1 \,\mathsf{R}\, \psi_2) \rightsquigarrow \mathsf{X}^i\mathsf{G}\psi_2$$

$$R_7 : (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\mathsf{G}\psi_2) \rightsquigarrow \begin{cases} \mathsf{X}^i\mathsf{G}\mathsf{Y}^{i-j}\psi_2 & \text{if } i > j \\ \mathsf{X}^j\mathsf{G}\psi_2 & \text{otherwise} \end{cases}$$

$$R_{flat} : \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\psi_2 \,\mathsf{R}\, (\ldots (\psi_{n-1} \,\mathsf{R}\, \psi_n) \ldots))) \rightsquigarrow \mathsf{X}^i((\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \ldots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i\top) \ldots)) \,\mathsf{R}\, \psi_n) \qquad \text{for any } n \geq 3$$

where ψ, ψ_1, ψ_2, and ψ_3 are pure past formulas. It is worth noting that, so far, we do not have rules (preserving equivalence) to deal with the following cases: (i) $(\phi_1 \wedge \phi_2) \,\mathsf{R}\, (\phi)$, (ii) $(\mathsf{G}\phi_1) \,\mathsf{R}\, (\phi)$ or (iii) $(\phi_1 \,\mathsf{R}\, \phi_2) \,\mathsf{R}\, (\phi)$. This is why, in Definition 21, we restricted the left-hand argument of each *release* operator to be a $\mathsf{LTL{+}P_{BF}}$ formula.

Definition 50 (Normalization)**.** *Given a* Pastified-$\mathsf{LTL_{EBR}}$+P *formula ϕ, we define* normalize(ϕ) *as the formula* flatten(applyR1R7(ϕ))*, where* applyR1R7 *is the algorithm in Fig. 7.2 and* flatten *is the algorithm in Fig. 7.4.*

We state the following results about the correctness and complexity of normalize(ϕ).

Lemma 17 (Soundness of normalize($\cdot$))**.** *For any* Pastified-$\mathsf{LTL_{EBR}}$+P *formula ϕ, it holds that ϕ and* normalize(ϕ) *are equivalent and* normalize(ϕ) *is a* Normal-$\mathsf{LTL_{EBR}}$+P *formula.*

Lemma 18 (Complexity of normalize($\cdot$))**.** *For any* Pastified-$\mathsf{LTL_{EBR}}$+P *formula ϕ,* normalize(ϕ) *can be built in $\mathcal{O}(n)$ time, and the size of* normalize(ϕ) *is $\mathcal{O}(n)$, where $n = |\phi|$.*

```
/*
* Input: φ ∈ Pastified-LTL_EBR+P
* Output: φ ∈ Pastified-LTL_EBR+P
* Notation:
*     φ, φ_1, ..., φ_n ∈ LTL_EBR+P
*     ψ, ψ_1, ψ_2, ψ_3 ∈ LTL+P_P
*     p ∈ Σ
*/
define applyR1R7(φ){
  switch(φ){
    // Base case = LTL+P_P formulae
    case p :
    case ¬ψ :
    case Yψ_1 :
    case ψ_1 S ψ_2 :
      return φ

    // And/Or Operators
    case φ_1 ∧ φ_2 :
      return applyR1R7(φ_1) ∧
             applyR1R7(φ_2)

    case φ_1 ∨ φ_2 :
      return applyR1R7(φ_1) ∨
             applyR1R7(φ_2)

    // Next Rewriting Rules
    case Xφ_1 :
      φ_1 ← applyR1R7(φ_1)
      switch(φ_1){
        case φ_2 ∧ ··· ∧ φ_n : // rule R_1
          return Xφ_2 ∧ ··· ∧ Xφ_n
        default :
          return Xφ_1
      }

    // Globally Rewriting Rules
    case Gφ_1 :
      φ_1 ← applyR1R7(φ_1)
      switch(φ_1){
        case φ_2 ∧ ··· ∧ φ_n : // rule R_2
          φ_2 ← resolve_globally(φ_2)
          ...
          φ_n ← resolve_globally(φ_n)
          return φ_2 ∧ ··· ∧ φ_n
        default:
          φ_1 ← resolve_globally(φ_1)
          return φ_1
      }

    // Release Rewriting Rules
    case ψ R φ_1 :
      φ_1 ← applyR1R7(φ_1)
      switch(φ_1){
        case φ_2 ∧ ··· ∧ φ_n : // rule R_2
          φ_2 ← resolve_release(ψ,φ_2)
          ...
          φ_n ← resolve_release(ψ,φ_n)
          return φ_2 ∧ ··· ∧ φ_n
        default :
          φ_1 ← resolve_release(ψ,φ_1)
          return φ_1
      }

    default :
      unreachable_code()
  }
}
```

Figure 7.2: The applyR1R7 algorithm (part I).

```
define resolve_globally(φ){
  switch(φ){
    case X^i ψ : // rule R_3 (2nd case)
      return X^i G ψ
    case X^i G ψ : // rule R_5
      return X^i G ψ
    case X^i (ψ R ψ_1) : // rule R_6
      return X^i G ψ_1
    default :
      return G ψ
  }
}

define resolve_release(X^i ψ_1, φ){
  switch(φ){
    case X^j ψ_2 : // rule R_3
      if(i > j)
        return X^i (ψ_1 R (Y^{i-j} ψ_2))
      else
        return X^j ((Y^{j-i} ψ_1) R ψ_2)
    case X^j G ψ_2 : // rule R_7
      if(i > j)
        return X^i G(Y^{i-j} ψ_2)
      else
        return X^j G ψ_2
    case X^j (ψ_2 R φ_3) : // rule R_4
      if(i > j)
        return X^i (ψ_1 R ((Y^{i-j} ψ_2) R (Y^{i-j} ψ_3)))
      else
        return X^j ((Y^{j-i} ψ_1) R (ψ_2 R ψ_3))
    default :
      return (X^i ψ_1) R φ
  }
}
```

Figure 7.3: The applyR1R7 algorithm (part II).

```
define flatten(φ){
  switch(φ){
    case φ_1 ∧ φ_2 :
      return flatten(φ_1) ∧ flatten(φ_2)

    case φ_1 ∨ φ_2 :
      return flatten(φ_1) ∨ flatten(φ_2)

    // rule R_flat
    case X^i(ψ_1 R (ψ_2 R (...(ψ_{n-1} R ψ_n)...))) :
      return X^i((ψ_{n-1} ∧ O(ψ_{n-2} ∧ ... O(ψ_1 ∧ Y^i ⊤)...)) R ψ_n)

    default :
      return φ
  }
}
```

Figure 7.4: The flatten algorithm.

7.2.3 From Normal-$\mathsf{LTL_{EBR}}$+P to deterministic SSA

The particular shape of Normal-$\mathsf{LTL_{EBR}}$+P formulas makes it possible to encode the specification into deterministic symbolic safety automata (SSA, recall Definitions 37 and 40). The key observation is that $\mathsf{LTL}{+}\mathsf{P_P}$ formulas can be encoded into deterministic automata: since these formulas talk exclusively about the past, their truth can be evaluated at any single step depending only on previous steps, without making any guess about the future ("the past already happened"). But $\mathsf{LTL}{+}\mathsf{P_P}$ formulas are not the only ones that can be encoded deterministically. Consider, for instance, the formula $\phi \equiv \mathsf{X}p \vee \mathsf{X}q$. At a first glance, it may seem that ϕ needs a non-deterministic automaton to be encoded, which at the first state makes a choice about whether p or q will hold in the next state. Nevertheless, this formula is equivalent to $\mathsf{X}(p \vee q)$ and it corresponds to the *deterministic* automaton that, once arrived in its second state by reading any proposition symbol, proceeds to an accepting state by reading either p or q, or goes to a sink (*error*) state otherwise.

Pastified-$\mathsf{LTL_{EBR}}$+P in its normal form combines pure past formulas into a broader language that can still be turned into symbolic deterministic automata, extending the above intuition and exploiting the *monitorability* of *universal* temporal operators.

Monitoring is a technique coming from *runtime verification* [131]. Consider the formula $\mathsf{G}\alpha$. By observing a state sequence, at each step we can decide if a *violation* has occurred; indeed, if α is false at the current step, then the value of $\mathsf{G}\alpha$ is certainly false for each of the previous steps. More generally, universal temporal formulas, such as $\mathsf{G}\phi$ and $\phi_1 \,\mathsf{R}\, \phi_2$, are *monitorable*, meaning that a violation of them can be decided on the basis of the observation of a *finite* number of steps. In particular, reporting an error in the next state can be done by considering only the current values. This means that any universal temporal operator can be monitored by adding a Boolean *error variable* with a *deterministic* transition relation.

Therefore, despite not being able to evaluate the truth of a formula such as $\mathsf{G}\alpha$, as it can be done in the case of past operators, we can nevertheless state in the accepting condition that an error state can never be reached. In this way, if the trace is accepting, that is, an error state can never be reached, then we know that there are no violations, *e.g.*, for $\mathsf{G}\alpha$, we have forced α to be true in every state. Otherwise, if the trace is not accepting, that is, an error state

is reachable, we know that there is a (finite) violation and that the temporal formula was falsified at some step. We therefore introduce an *error bit* for each $\mathsf{X}^i\psi_1$, $\mathsf{X}^i\mathsf{G}\psi_1$, and $\mathsf{X}^i(\psi_1 \,\mathsf{R}\, \psi_2)$ of a normal Pastified-$\mathsf{LTL_{EBR}}$+P formula.

Let ϕ be a Normal-$\mathsf{LTL_{EBR}}$+P formula over the alphabet $\Sigma = \mathcal{C} \cup \mathcal{U}$. We define the deterministic SSA $\mathcal{A}(\phi) = (V, I, T, S)$ as follows:

- *Variables.* The set of *state variables* of the automaton is defined as $X = X_P \cup X_F \cup X_C$, where:

$$X_P = \{v_\alpha \mid \alpha \text{ is an } \mathsf{LTL{+}P_P} \text{ subformula of } \phi\}$$

$$X_F = \left\{ error_\varphi \,\middle|\, \begin{array}{l} \varphi \text{ is subformula of } \phi \text{ of the form} \\ \mathsf{X}^i\psi,\ \mathsf{X}^i\mathsf{G}\psi, \text{ or } \mathsf{X}^i(\psi_1 \,\mathsf{R}\, \psi_2) \end{array} \right\}$$

$$X_C = \left\{ counter_i \,\middle|\, \begin{array}{l} i \in \{0, \ldots, \log_2 d\} \\ d \text{ max. among all } \mathsf{X}^d\psi \text{ in } \phi. \end{array} \right\}$$

  Intuitively, variables in X_P track the truth value of all the pure past subformulas, variables in X_F implement the above-described monitoring mechanism, and variables in X_C are used to encode a binary counter used to monitor nested *tomorrow* operators. In particular, for n nested *tomorrow* operators, a counter with $\log_2(n)$ bits is needed.

- *Initial state.* All the state variables, including the counter bits, are initially false, that is, $I(X) = \bigwedge_{x \in X} \neg x$.

- *Transition relation.* $T(X, \Sigma, X')$ is the conjunction of the transition *functions* of the binary counter and the monitors of each subformula of ϕ, as will be defined later. Notice that each conjunct is of the form $x' \leftrightarrow \beta_x(X \cup \Sigma)$, and thus it corresponds to a deterministic transition relation.

- *Safety condition.* $S(X)$ is a Boolean formula obtained from ϕ by replacing each formula $\varphi \in X_F$ by $\neg error_\varphi$, *i.e.*, $S(X) = \phi[\varphi / \neg error_\varphi]$.

We now define the monitors for the binary counter, used to handle nested *tomorrow* operators, any formula $\psi \in \mathsf{LTL{+}P_P}$, and any Normal-$\mathsf{LTL_{EBR}}$+P formula of one of the forms $\mathsf{X}^i\psi_1$, $\mathsf{X}^i\mathsf{G}\psi_1$, and $\mathsf{X}^i(\psi_1 \,\mathsf{R}\, \psi_2)$. We give the definition of the monitors using the SMV language (recall Section 5.2.3), as it provides useful shorthands (like

the *switch-case* primitive). Each of the following SMV statement corresponds to the Boolean formula that defines transition functions of our monitors.

We define a counter with $\log_2(n)$ bits that starts with value 0, increments at each successive state and, once it reaches the value n, it keeps its value. The monitor for the counter is defined as follows:

$$\texttt{next}(\texttt{counter}_0) := (\bigwedge_{j=1}^{\log_2(n)} \texttt{counter}_j) \vee \neg\texttt{counter}_0$$
$$\texttt{next}(\texttt{counter}_i) := (\bigwedge_{j=1}^{\log_2(n)} \texttt{counter}_j) \vee ((\bigwedge_{j=0}^{i-1} \texttt{counter}_j) \leftrightarrow \neg\texttt{counter}_i)$$

If ψ is a formula in the *pure past* layer, its monitor is defined as follows:

$$\texttt{next}(v_{\mathsf{Y}\alpha}) := v_\alpha \wedge \texttt{counter} > 0$$
$$\texttt{next}(v_{\mathsf{Z}\alpha}) := v_\alpha \vee \texttt{counter} \leq 0$$
$$\texttt{DEFINE}$$
$$v_{\alpha\mathsf{S}\beta} := v_\beta \vee (v_\alpha \wedge v_{\mathsf{Y}(\alpha\mathsf{S}\beta)})$$
$$v_{\alpha\mathsf{T}\beta} := (v_\alpha \wedge v_\beta) \vee (v_\beta \wedge v_{\mathsf{Z}(\alpha\mathsf{T}\beta)})$$

If ψ is a propositional atom, a negation, or a disjunction of pure past formulas, we define its monitor as follows:

$$\texttt{DEFINE}$$
$$v_p := p$$
$$v_{\neg\alpha} := \neg v_\alpha$$
$$v_{\alpha\vee\beta} := v_\alpha \vee v_\beta$$

For each formula ϕ of type $\mathsf{X}^i\psi$, where ψ is a pure past formula, we introduce a new error bit $error_\phi$. Its monitor is defined as follows:

$$\texttt{next}(error_{\mathsf{X}^i\psi}) := \texttt{case}$$
$$error_{\mathsf{X}^i\psi} : \texttt{TRUE};$$
$$counter = i \wedge \neg v_\psi : \texttt{TRUE};$$
$$\texttt{TRUE} : \texttt{FALSE};$$
$$\texttt{esac}$$

If $\phi := \mathsf{X}^i\mathsf{G}\psi$, where ψ is a pure past formula, we introduce a new error bit $error_\phi$, and we define its monitor as follows:

$$\texttt{next}(error_{\mathsf{X}^i\mathsf{G}\psi}) := \texttt{case}$$
$$counter < i : \texttt{FALSE};$$
$$\neg error_{\mathsf{X}^i\mathsf{G}\psi} \wedge v_\psi : \texttt{FALSE};$$
$$\texttt{TRUE} : \texttt{TRUE};$$
$$\texttt{esac}$$

The same for $\phi := \mathsf{X}^i(\psi_1 \,\mathsf{R}\, \psi_2)$:

```
next(error_{X^i(ψ1 R ψ2)}) := case
    counter < i : FALSE;
    ¬error_{X^i(ψ1 R ψ2)} ∧ v^i_{ψ1^p} : FALSE;
    ¬error_{X^i(ψ1 R ψ2)} ∧ v_{ψ1} ∧ v_{ψ2} : FALSE;
    ¬error_{X^i(ψ1 R ψ2)} ∧ v_{ψ2} : FALSE;
    TRUE : TRUE;
  esac

next(v^i_{ψ1^p}) := case
    counter < i : FALSE;
    v_{ψ1} : TRUE;
    v^i_{ψ1^p} : TRUE;
    TRUE : FALSE;
  esac
```

In Fig. 7.5, we describe the execution of all the steps described so far on a simple formula.

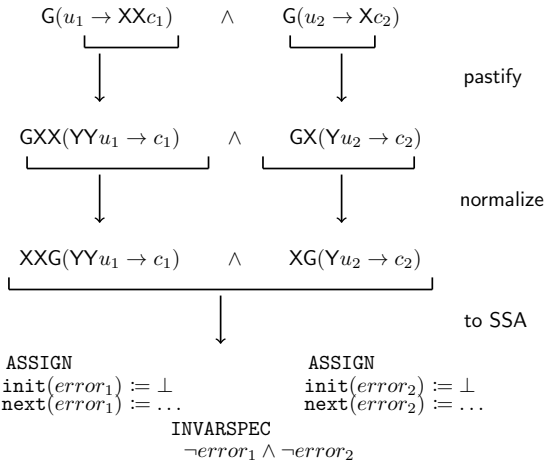


Figure 7.5: The execution of the sequence of steps: a simple example.

Before giving the correctness and complexity results for the construction of deterministic SSAa we have just described, we prove the following lemma.

Lemma 19. *For each* $\mathsf{Normal\text{-}LTL_{EBR}}$+P *formula* ϕ*, for each* LTL+$\mathsf{P_P}$ *formula* $\alpha \in \mathsf{LTL}$+$\mathsf{P_P}$ *that is a subformula of* ϕ*, for each state sequence* σ*, and for each* $i \geq 0$*,* $\sigma(i) \models \alpha$ *iff* $\tau(i) \models v_\alpha$*, where* τ *is the trace of* $\mathcal{A}(\phi)$ *induced by* σ*.*

Proof. We prove the lemma by induction on the structure of α. For the base case, $\sigma(i) \models p \in \Sigma$ iff $\tau(i) \models v_p$; since by definition of its monitor $v_p \leftrightarrow p$, we have that $\sigma(i) \models p$ iff $\tau(i) \models p$; since τ is induced by σ, this is always true.

For the inductive step, consider first $\alpha \vee \beta$. If $\sigma(i) \models \alpha \vee \beta$, then either $\sigma(i) \models \alpha$ or $\sigma(i) \models \beta$; by inductive hypothesis, either $\tau(i) \models v_\alpha$ or $\tau(i) \models v_\beta$; finally, by the definition of the monitor for disjunction, we have that $\tau(i) \models v_{\alpha \vee \beta}$. The opposite case and the case for $\neg\alpha$ can be proven similarly.

Consider the case for $\mathsf{Y}\alpha$. If $\sigma(i) \models \mathsf{Y}\alpha$, then $\sigma(i-1) \models \alpha$ and $i > 0$. By inductive hypothesis $\tau(i-1) \models v_\alpha$ and $i > 0$; by definition of the monitor for $\mathsf{Y}\alpha$, $\tau(i) \models v_{\mathsf{Y}\alpha}$.

Finally, we prove the case for $\alpha\,\mathsf{S}\,\beta$. If $\sigma(i) \models \alpha\,\mathsf{S}\,\beta$, then either $\sigma(i) \models \beta$ or $\sigma(i) \models \alpha \wedge \mathsf{Y}(\alpha\,\mathsf{S}\,\beta)$; by inductive hypothesis, either $\tau(i) \models v_\beta$ or $\tau(i) \models v_\alpha \wedge v_{\mathsf{Y}(\alpha \mathsf{S} \beta)}$; by definition of the monitor for $\alpha\,\mathsf{S}\,\beta$, we have that $\tau(i) \models v_{\alpha S \beta}$. The opposite direction can be proven in the similar way. □

Proposition 18. *Let ϕ be a* Normal-$\mathsf{LTL_{EBR}}$+P *formula, with $|\phi| = n$. Then, there exists a deterministic SSA of size $\mathcal{O}(n)$ that accepts the same language.*

Proof. Let ϕ be a Normal-$\mathsf{LTL_{EBR}}$+P formula over the alphabet Σ and let $\mathcal{A}(\phi) = (X \cup \Sigma, I(X), T(X, \Sigma, X'), S(X))$ be the deterministic symbolic safety automaton as previously defined.

Soundness. We first prove that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A}(\phi))$. In particular we prove that $\forall\sigma \in \mathcal{L}(\phi).\sigma \models \phi$ iff $\tau(i) \models S(X)\ \forall i \geq 0$, where τ is the trace induced by σ in $\mathcal{A}(\phi)$. Recall that $S(X) = \phi[\varphi/\neg error_\varphi]$. We proceed by induction on the structure of ϕ.

For the base case we consider $\phi = \mathsf{X}^i\mathsf{G}\alpha$ where $\alpha \in \mathsf{LTL{+}P_P}$(the cases for $\mathsf{X}^i\alpha$ and $\mathsf{X}^i(\alpha\mathsf{R}\beta)$ are similar). If $\sigma \models \mathsf{X}^i\mathsf{G}\alpha$ then $\sigma(i) \models \mathsf{G}\alpha$, that is $\sigma(j) \models \alpha\ \forall j \geq i$. By Lemma 19, $\tau(j) \models v_\alpha\ \forall j \geq i$. The following points hold:

1. given the first condition in the monitor for $\mathsf{X}^i\mathsf{G}\alpha$, we have that $\tau(j) \models \neg error_\phi\ \forall 0 \leq j < i$;
2. given the previous point and the fact that $\tau(j) \models v_\alpha\ \forall j \geq i$, by the second condition of the monitor we have that $\tau(j) \models \neg error_\phi\ \forall j \geq i$.

By these two points, it follows that $\tau(j) \models \neg error_\phi \ \forall j \geq 0$. Vice versa, if $\tau(j) \models \neg error_\phi \ \forall j \geq 0$, then by definition of the monitor we have that $\tau(j) \models v_\alpha \ \forall j \geq i$. By Lemma 19, $\sigma(j) \models \alpha \ \forall j \geq i$, that is $\sigma \models \mathsf{X}^i\mathsf{G}\alpha$.

For the inductive step, consider first $\phi = \phi_1 \wedge \phi_2$. If $\sigma \models \phi$, then $\sigma \models \phi_1$ and $\sigma \models \phi_2$. By inductive hypothesis, $\tau(i) \models \phi_1[\varphi/\neg error_\varphi] \ \forall i \geq 0$ and $\tau(i) \models \phi_2[\varphi/\neg error_\varphi] \ \forall i \geq 0$, that is $\tau(i) \models (\phi_1 \wedge \phi_2)[\varphi/\neg error_\varphi] \ \forall i \geq 0$. The opposite direction can be proven in the same way.

Finally, consider the case $\phi = \phi_1 \vee \phi_2$. If $\sigma \models \phi$, then by inductive hypothesis either $\tau(i) \models \phi_1[\varphi/\neg error_\varphi] \ \forall i \geq 0$ or $\tau(i) \models \phi_2[\varphi/\neg error_\varphi] \ \forall i \geq 0$; thus $\tau(i) \models (\phi_1 \vee \phi_2)[\varphi/\neg error_\varphi] \ \forall i \geq 0$. For the opposite direction, assume that $\tau(i) \models (\phi_1 \vee \phi_2)[\varphi/\neg error_\varphi] \ \forall i \geq 0$; since each $error_\varphi$ is monotone (once set to true, it remains true forever), it holds that either $\tau(i) \models \phi_1[\varphi/\neg error_\varphi] \ \forall i \geq 0$ or $\tau(i) \models \phi_2[\varphi/\neg error_\varphi] \ \forall i \geq 0$. By inductive hypothesis, either $\sigma \models \phi_1$ or $\sigma \models \phi_2$, that is $\sigma \models \phi_1 \vee \phi_2$.

Complexity. Let $n = |\phi|$; it holds that:

- $|X| = |M_P| + |M_F| \in \mathcal{O}(n)$, since $|M_P| + |M_F| \leq n$;
- $|I(X)|, |T(X, \Sigma, X')| \in \mathcal{O}(n)$, since they are both summations over the variables in X;
- $|S(X)| \in \mathcal{O}(n)$, since $S(X)$ is obtained from ϕ by replacing each subformula in M_F with a variable.

Overall, we have that the size of $\mathcal{A}(\phi)$ is $\mathcal{O}(n)$. □

Theorem 36. *Let ϕ be an $\mathsf{LTL_{EBR}{+}P}$ formula and let $n = |\phi|$. Then, there exists a deterministic SSA of size $\mathcal{O}(n)$ that accepts the same language.*

Proof. Let ϕ be an $\mathsf{LTL_{EBR}{+}P}$ formula of size n. By Proposition 17, we can build an equivalent $\mathsf{Pastified\text{-}LTL_{EBR}{+}P}$ formula ϕ' of size $\mathcal{O}(n)$; by Lemma 18, from ϕ' we can obtain an equivalent $\mathsf{Normal\text{-}LTL_{EBR}{+}P}$ formula ϕ'' of linear size with respect to $|\phi'|$. Finally, by Proposition 18, the size of the deterministic symbolic safety automaton $\mathcal{A}(\phi'')$ is linear in $|\phi'|$, hence $|\mathcal{A}(\phi'')| \in \mathcal{O}(n)$. □

7.3 An optimal algorithm for $\mathsf{LTL_{EBR}}$+P realizability

7.3.1 Solving the game on the deterministic SSA

Once we have obtained the deterministic SSA $\mathcal{A}(\phi)$ for an $\mathsf{LTL_{EBR}}$+P formula ϕ with the steps described in the previous sections, we can use $\mathcal{A}(\phi)$ as the arena of a two-player game between Controller and Environment in order to solve the realizability (and synthesis) problem for ϕ.

Let us focus on the *safety game* represented by the deterministic symbolic safety automaton $\mathcal{A} = (V = X \cup \Sigma, I, T, \mathsf{G}\alpha(\mathsf{X})))$ with $\Sigma = \mathcal{U} \cup \mathcal{C}$ and $\mathcal{U} \cap \mathcal{C} = \varnothing$ (recall Section 5.3.4). Safety games have been extensively studied, as their reachability objective makes the problem simpler than considering ω-regular objectives, such as, for instance, Büchi and Rabin conditions.

The aim of Controller is to choose an infinite sequence of values for the *controllable* variables in such a way that, no matter what values for the *uncontrollable* variables are chosen by Environment, *the* trace induced by the play in $\mathcal{A}(\phi)$ is *safe*, that is, it visits only states s such that $s \models S(X)$ (recall Definition 37). Since in our case $\mathcal{A}(\phi)$ recognizes exactly the language of ϕ, the play satisfies ϕ, and thus Controller has a winning strategy for ϕ.

Since the organization of the SYNTCOMP [117], many optimized tools have been proposed in the literature to solve safety games. For this reason, we chose to use a safety synthesizer as a black box. The majority of these tools accept as input a symbolic arena described in terms of and-inverter graphs (or AIGER format [19]), so we provide a simple utility to obtain the AIGER representation of *Functional* SMV modules (recall Section 5.2.3), that is, SMV modules with the transition relation expressed only in terms of ASSIGN statements, such as the ones resulting from our encoding. The AIGER model is then given as input to the chosen safety synthesizer, completing the process outlined in Fig. 7.1.

The next theorem states the complexity of the procedure.

Theorem 37. *The realizability problem for* $\mathsf{LTL_{EBR}}$+P *belongs to* EXPTIME.

Proof. Since it is easy to see that the time complexity of all the steps matches their space complexity, we have an algorithm to turn

an $\mathsf{LTL_{EBR}}$+P formula ϕ into an equivalent deterministic SSA $\mathcal{A}(\phi)$ whose time complexity is $\mathcal{O}(n)$, where $n = |\phi|$. Since $\mathcal{A}(\phi)$ is symbolically represented, it can be turned into an explicit automaton $\mathcal{A}'(\phi)$ of size at most exponential in the size of $\mathcal{A}(\phi)$, that is, $|\mathcal{A}'(\phi)| \in \mathcal{O}(2^n)$. Finally, the time complexity of reachability games is *linear* in the size of the arena [68], and thus the overall time complexity of the realizability problem for $\mathsf{LTL_{EBR}}$+P is EXPTIME. □

It is interesting to briefly compare the proposed procedure for realizability to the one used by the SSYFT tool for Safety LTL specifications [201]. In that tool, the negation of the initial formula is first translated into first-order logic over finite words and then transformed into deterministic automata using the tool MONA [113], which uses the classical subset construction to determinize automata over finite words. Finally, SSYFT uses the classical backward fixpoint iteration to compute the set of winning states over the DFA. It is worth to notice that the way MONA represents automata is *not* fully symbolic: the set of states is explicitly represented, while it uses a BDD for each pair of states in order to represent symbolically the transitions between the two corresponding states. In contrast to subset construction, our solution performs the pastification of $\mathsf{LTL{+}P_{BF}}$ formulas. Most importantly, our construction of deterministic monitors is carried out in a fully symbolic way.

7.3.2 Complexity of $\mathsf{LTL_{EBR}}$+P

In the following, we prove that the realizability problem for $\mathsf{LTL_{EBR}}$+P is EXPTIME-complete by showing its EXPTIME-hardness.

In fact, we first prove that the problem of establishing whether or not an $\mathsf{LTL_{EBR}}$+P formula is *satisfiable* is PSPACE-hard. As we will see, such a preliminary result, besides being interesting by itself, allows us to set up a machinery that is crucial for the following hardness proof. Then, we demonstrate that deciding *realizability* for $\mathsf{LTL_{EBR}}$+P, that is, proving the existence of a winning strategy for controller, is EXPTIME-hard.

We start with the satisfiability problem, and provide a reduction from the *corridor tiling problem*, which is known to be PSPACE-complete [195, 152]. As a first step, we define the notions of tiling structure and of tiling.

Definition 51 (Tiling Structure)**.** *A* tiling structure *is a tuple* $\overline{\mathcal{T}} = \langle \overline{T}, t_\perp, t_\top, \overline{H}, \overline{V}, n \rangle$*, where* $\overline{T}$ *is a finite set of elements, called* tiles*,* $t_\perp, t_\top \in \overline{T}$ *are the* bottom tile *and the* top tile*, respectively,* $\overline{H}, \overline{V} \subseteq \overline{T} \times \overline{T}$ *are the* horizontal *and the* vertical relations*, respectively, and* $n \in \mathbb{N}$ *is a natural number encoded in* unary*.*

Definition 52 (Tiling)**.** *Let* $\overline{\mathcal{T}} = \langle \overline{T}, t_\perp, t_\top, \overline{H}, \overline{V}, n \rangle$ *be a* tiling structure*. A* tiling *for* $\overline{\mathcal{T}}$ *is a function* $f : \mathbb{N} \times [0, n) \to \overline{T}$ *such that associates a tile in* $\overline{T}$ *with every position of the infinite discrete corridor of height* n *in such a way that:*

1. *the horizontal relation is satisfied:*

$$\forall x \in \mathbb{N}\ \forall y \in [0, n)\ .\ f(x, y) \overline{H} f(x + 1, y);$$

2. *the vertical relation is satisfied:*

$$\forall x \in \mathbb{N}\ \forall y \in [0, n - 1)\ .\ f(x, y) \overline{V} f(x, y + 1);$$

3. *the (infinite) bottom row of the corridor is tiled only with* $t_\perp$*:*

$$\forall x \in \mathbb{N}\ .\ f(x, 0) = t_\perp;$$

4. *the (infinite) top row of the corridor is tiled only with* $t_\top$*:*

$$\forall x \in \mathbb{N}\ .\ f(x, n - 1) = t_\top.$$

Given a tiling structure $\overline{\mathcal{T}}$ (Definition 51), the *corridor tiling problem* is the problem of checking whether there exists a tiling (Definition 52) for $\overline{\mathcal{T}}$. This problem is known to be PSPACE-complete [195, 152].

We prove the PSPACE-hardness of the satisfiability problem for $\mathsf{LTL_{EBR}}$+P by reducing the corridor tiling problem to it. The proof basically defines a correspondence between tilings for the tiling structure $\overline{\mathcal{T}}$ (see Definition 51) and models, that is, infinite sequences of states, of an $\mathsf{LTL_{EBR}}$+P formula built starting from $\overline{\mathcal{T}}$.

Theorem 38. *The satisfiability problem of* $\mathsf{LTL_{EBR}}$+P *is* PSPACE*-hard.*

Proof. Given a tiling structure $\overline{\mathcal{T}} = \langle \overline{T}, t_\perp, t_\top, \overline{H}, \overline{V}, n \rangle$, we build a formula $\phi_{\overline{\mathcal{T}}}$ of $\mathsf{LTL_{EBR}}$+P such that there is a tiling f for $\overline{\mathcal{T}}$ *if and only if* there is a model σ for $\phi_{\overline{\mathcal{T}}}$.

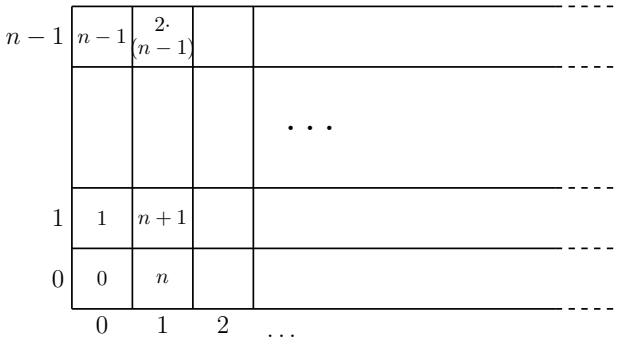


Figure 7.6: Correspondence between a tiling f of $\overline{\mathcal{T}}$ and a model σ of $\phi_{\overline{\mathcal{T}}}$.

The correspondence between a tiling f for $\overline{\mathcal{T}}$ and a model σ for $\phi_{\overline{\mathcal{T}}}$ is shown in Fig. 7.6. The first state of σ corresponds to the bottom-left tile of the corridor, that is, the tile at coordinates $(0, 0)$; the second state of σ corresponds to the tile at coordinates $(0, 1)$; the n-th state of σ corresponds to the tile at coordinates $(1, 0)$, and so on and so forth. Formally, for all $i \in \mathbb{N}$, σ_i is in correspondence with $f(x, y)$ where $x = \lfloor \frac{i}{n} \rfloor$ and $y = (i \mod n)$.

We will define $\phi_{\overline{\mathcal{T}}}$ as a formula over the set of atomic proposition $\mathcal{AP} = \overline{T} \cup \{b_\bot, b_\top\}$, where $\overline{T}$ is the set of tiles of the tiling structure $\overline{\mathcal{T}}$, and $b_\bot$ and $b_\top$ are two special variables for the bottom and top border of the corridor. We build it step-by-step by introducing all its components (conjuncts) separately:

- *mutual exclusion*: each state of σ can contain at most one tile:

$$\mathsf{G}\Big(\bigwedge_{t,t' \in \overline{T}, t' \neq t} \neg(t \wedge t')\Big)$$

- *no empty cells*: each state must contain at least one tile:

$$\mathsf{G}\Big(\bigvee_{t \in \overline{T}} t\Big)$$

- *horizontal constraint*: for each $i \in \mathbb{N}$, the tile at state i must be in horizontal relation with the tile at state $i + n$, that is,

the one to its right in the corridor:

$$\mathsf{G}(\bigvee_{(t,t')\in\overline{H}} (t \wedge \mathsf{X}^n t'))$$

- *top border*: the variable $b_\top$ is true at all and only the positions of σ corresponding to the top border of the corridor:

$$\mathsf{G}^{[0,n-2]}\neg b_\top \wedge \mathsf{X}^{n-1} b_\top \wedge \mathsf{G}(b_\top \leftrightarrow \mathsf{X}^n b_\top)$$

- *bottom border*: the variable $b_\bot$ is true at all and only the positions of σ corresponding to the bottom border of the corridor:

$$b_\bot \wedge \mathsf{G}(b_\top \leftrightarrow \mathsf{X} b_\bot)$$

- *vertical constraint*: for each $i \in [0, n-1)$, that is, for all the tiles except for the ones in the top border, the tile at state i must be in vertical relation with the tile at state $i+1$, that is, the one above it in the corridor:

$$\mathsf{G}(\neg b_\top \rightarrow \bigvee_{(t,t')\in\overline{V}} (t \wedge \mathsf{X} t'))$$

- *top row constraint*: all the states of σ corresponding to the top border of the corridor must satisfy $t_\top$:

$$\mathsf{G}(b_\top \rightarrow t_\top)$$

- *bottom row constraint*: all the states of σ corresponding to the bottom border of the corridor must satisfy $t_\bot$:

$$\mathsf{G}(b_\bot \rightarrow t_\bot)$$

The formula $\phi_{\overline{\mathcal{T}}}$ is the conjunction of all the above formulas. It is easy to see that $\overline{\mathcal{T}}$ has a tiling f if and only if $\phi_{\overline{\mathcal{T}}}$ has a model σ, that is, if and only if $\phi_{\overline{\mathcal{T}}}$ is satisfiable. Moreover, since by Definition 51 the number n is encoded in *unary*, and since the bounded operators of $\mathsf{LTL_{EBR}}$+P are only shortcuts (*e.g.*, $\mathsf{X}^n p$ has dimension n), the size of $\phi_{\overline{\mathcal{T}}}$ is polynomial in the size of $\overline{\mathcal{T}}$. Therefore, the proposed reduction works in polynomial time. Since the corridor tiling problem is PSPACE-complete, we can conclude that the satisfiability problem of $\mathsf{LTL_{EBR}}$+P is PSPACE-hard. □

We now prove that the realizability problem of $\mathsf{LTL}_{\mathsf{EBR}}$+P is EXPTIME-hard by means of a reduction from the *corridor tiling game*, which is the two-players variant of the tiling problem of Definition 52. Tiling games extend tiling problems by considering two players: *Constructor*, whose goal is to build a tiling for the tiling structure $\overline{\mathcal{T}}$, and *Saboteur*, trying to prevent this from happening.

The two players play one at a time (with *Constructor* being the first one to play), choose a tile from $\overline{T}$ and position it on the tiling structure $\overline{\mathcal{T}}$ in a precise order: the first position is the one at coordinates $(0, 0)$ (see Fig. 7.6), the second position is the one at coordinates $(0, 1)$, and so on and so forth. When a column is entirely tiled, the game proceeds on the next column. If there is no tile fitting the next position or the definition of tiling (Definition 52) is violated, then *Saboteur* wins. Otherwise, that is, in the case the game goes on forever, *Constructor* wins. The problem is EXPTIME-complete [41].

From an instance of the corridor tiling game, we will build a corresponding $\mathsf{LTL}_{\mathsf{EBR}}$+P formula which is realizable if and only if the tiling game is won by *Constructor*. In doing that, we will make use of the definition of $\phi_{\overline{\mathcal{T}}}$ from the proof of the PSPACE-hardness of the satisfiability problem of $\mathsf{LTL}_{\mathsf{EBR}}$+P.

The simplicity of the reduction comes from the similarities between the exponential corridor tiling game and the $\mathsf{LTL}_{\mathsf{EBR}}$+P realizability problem. In particular, besides the connection between tilings for a tiling structure $\overline{\mathcal{T}}$ and models for an $\mathsf{LTL}_{\mathsf{EBR}}$+P formula, that we already established in the previous proof, there is a straightforward correspondence between *Constructor* player and *Controller* player, as well as between *Saboteur* player and *Environment* player. Nevertheless, there are some issues that have to be solved in order to design a correct reduction between the two problems:

1. in tiling games, the two players choose the tile to position from the *same* set of tiles $\overline{T}$, while in the realizability problem, *Environment* and *Controller* player can choose the value only for their own set of *uncontrollable* and *controllable* variables, respectively;

2. in tiling games, exactly one player can decide which tile to place in a given cell, while in the realizability problem each state is determined by the choices of both players; since in the proof of Theorem 38 we established a correspondence between

cells in the corridor and states of a model of the $\mathsf{LTL_{EBR}}$+P formula, this mismatch is present, and must be solved;

3. the player who moves first is different in the two problems: in tiling games, *Constructor* moves first, while in the realizability problem *Environment* moves first.

We now prove the EXPTIME-hardness of the realizability problem of $\mathsf{LTL_{EBR}}$+P by dealing with all the above issues.

Theorem 39. *The realizability problem of* $\mathsf{LTL_{EBR}}$+P *is* EXPTIME*-hard.*

Proof. Given a tiling structure $\overline{\mathcal{T}} = \langle \overline{T}, t_\bot, t_\top, \overline{H}, \overline{V}, n \rangle$, we build a formula $\phi^{game}_{\overline{\mathcal{T}}}$ of $\mathsf{LTL_{EBR}}$+P such that *Constructor* wins the corridor tiling game in $\overline{\mathcal{T}}$ *if and only if* there exists a strategy of *Controller* (Definition 41) for $\phi^{game}_{\overline{\mathcal{T}}}$.

As anticipated, we reuse the $\mathsf{LTL_{EBR}}$+P formula $\phi_{\overline{\mathcal{T}}}$ introduced in the proof of Theorem 38. The formula $\phi^{game}_{\overline{\mathcal{T}}}$ is defined over the following sets of variables:

- $\mathcal{U} \coloneqq \{t^u \mid t \in \overline{T}\}$ is the set of uncontrollable variables, and
- $\mathcal{C} \coloneqq \{t^c \mid t \in \overline{T}\} \cup \{b_\top, b_\bot\}$ is the set of controllable variables.

In such a way, we solve the first of the above mentioned problems: each player has its own set of variables, which corresponds to a copy of the same set of tiles $\overline{T}$. Note also that $b_\top$ and $b_\bot$ are set to be *controllable*. This is because we want formulas for the *top* and *bottom border* to be *non-blocking* for the realizability of $\phi^{game}_{\overline{\mathcal{T}}}$, that is, we want $\phi^{game}_{\overline{\mathcal{T}}}$ to be realizable *if and only if* there exist some values $v_\top$ and $v_\bot$ such that $\phi^{game}_{\overline{\mathcal{T}}}[b_\top \mapsto v_\top, b_\bot \mapsto v_\bot]$ is realizable.

As it happened with $\phi_{\overline{\mathcal{T}}}$, we build $\phi^{game}_{\overline{\mathcal{T}}}$ step-by-step.

- The tiling generated by the two players during the play has to be a correct one (Definition 52). Here, we exploit formula $\phi_{\overline{\mathcal{T}}}$ that we introduced in the proof of Theorem 38. Since now we have not a single set of variables, but rather two disjoint sets $\mathcal{U}$ and $\mathcal{C}$, we impose the tiling to consist of tiles chosen either by Controller or by Environment by means of the following formula:

$$\phi_{\overline{\mathcal{T}}}[t \mapsto t^c \vee t^u]$$

- Next, for each cell of the corridor, we force that exactly one player can decide the tile to place on it and, at the next round, the other player does the same, thus addressing the second of the above-mentioned issues:

$$\mathsf{G}(\bigvee_{t\in T} t^u \leftrightarrow \mathsf{X}\bigvee_{t\in \overline{T}} t^c) \wedge \qquad \text{alternation between players}$$

$$\mathsf{G}(\bigvee_{t\in T} t^u \leftrightarrow \bigwedge_{t\in \overline{T}} \neg t^c) \qquad \text{mutual exclusion between players}$$

- Finally, environment player is the first who moves: $\bigvee_{t\in \overline{T}} t^u$.

We define $\phi^{game}_{\overline{\mathcal{T}}}$ as the conjunction of the three formulas above. It is easy to see that $\phi^{game}_{\overline{\mathcal{T}}}$ is an $\mathsf{LTL_{EBR}}$+P formula and that it is realizable if and only if *Constructor* wins the corridor tiling game. As it happens with $\phi_{\overline{\mathcal{T}}}$, $\phi^{game}_{\overline{\mathcal{T}}}$ is polynomial in the size of the tiling structure $\overline{\mathcal{T}}$. Since the corridor tiling problem is EXPTIME-complete, this proves the EXPTIME-hardness of the realizability from $\mathsf{LTL_{EBR}}$+P specifications. □

Given the EXPTIME-membership of the $\mathsf{LTL_{EBR}}$+P realizability problem, we obtain two results: (i) the $\mathsf{LTL_{EBR}}$+P realizability problem is EXPTIME-complete, and (ii) the algorithm described in Section 7.3.1 is *optimal*. This shows that $\mathsf{LTL_{EBR}}$+P is a syntactical (safety) fragment of LTL with a lower complexity for the realizability problem.

7.4 Experimental Evaluation

We implemented the proposed procedure (see Fig. 7.1) in a tool called EBR-LTL-SYNTH.[1] The transformation from $\mathsf{LTL_{EBR}}$+P to deterministic SSA together with the translation to AIGER has been implemented inside the NUXMV model checker [38]. As the backend for solving the safety game, we have chosen the SAT-based tool DEMIURGE [26].

We tested our tool on a set of scalable benchmarks divided in four categories (the propositional atoms starting with the letter c are controllable, while those starting with the letter u are uncontrollable):

[1] `https://es-static.fbk.eu/tools/ebr-ltl-synth/`

1. the first category is generated by the realizable formula:

$$\mathsf{G}(c_0 \wedge \mathsf{XG}(c_1 \wedge \cdots \wedge \mathsf{X}^n\mathsf{G}(c_n \vee u)\dots))$$

2. the second category is generated by the realizable formula:

$$\mathsf{G}((c_0 \vee u_0) \wedge \mathsf{XG}((c_1 \vee u_1) \wedge \cdots \wedge \mathsf{X}^n\mathsf{G}((c_n \vee u_n))\dots))$$

3. the third category is generated by the unrealizable formula:

$$\mathsf{G}(c) \wedge \bigvee_{i=1}^{n} \mathsf{G}(\bigwedge_{j=0}^{i} u_j)$$

4. the fourth category is generated by the unrealizable formula:

$$c \wedge \bigwedge_{i=1}^{n} \mathsf{X}^i(u_i \vee u_{i+1})$$

Each category contains the respective scalable formula for $n \in [1, 200]$, for a total of 800 benchmarks, half of which are realizable and the other half are unrealizable. We remark that the two unrealizable categories are already in normal form, while this is not true for the realizable categories.

We set a timeout of 180 seconds for each benchmark. We compared EBR-LTL-SYNTH with LTLSYNT [116], STRIX [137] (the version of SYNTCOMP 2020) and SSYFT [201]. The first two tools solve the realizability and synthesis problems for full LTL and are based on a translation to parity games. LTLSYNT uses SPOT [78] for efficient translation and manipulation of automata. STRIX implements several optimizations like specification splitting, that enables to split the initial formula in safety, co-safety, Büchi, and co-Büchi subformulas and speeds up the process of solving of the game. On the contrary, SSYFT solves the realizability problem for specifications written in Safety LTL (see Section 7.3.1 for a brief description of the SSYFT tool).

For realizability, we tested all the tools in their sequential configurations. LTLSYNT has two sequential configurations, which differ on whether the split of actions into Controller's and Environment's ones is performed before or after the determinization. STRIX has two sequential modes as well, depending on the kind of search on

the arena (depth-first for the first configuration and with a priority queue for the second). ssyft and ebr-ltl-synth have only one configuration.

Figure 7.7 shows the outcomes of the comparison between ebr-ltl-synth and the best configuration of ltlsynt: it can be clearly seen that, for both realizable and unrealizable formulas, ltlsynt presents an exponential blow-up in the solving time that is avoided by ebr-ltl-synth. Figure 7.8 compares ebr-ltl-synth with the best configuration of strix: while for the majority of realizable formulas strix reaches a timeout, it is interesting to note that for the unrealizable benchmarks the difference between the solving time of the two tools is linear, mostly showing a 10x improvement in favor of ebr-ltl-synth. The survival plots for the set of realizable and unrealizable scalable benchmarks are shown in Figs. 7.9 and 7.10, respectively. The better performance of ebr-ltl-synth with respect to ltlsynt and strix is most likely due to the fact that the symbolic representation of safety automata used by ebr-ltl-synth is more succinct than the representation of the arenas for parity games used by the other two tools. Moreover, ltlsynt and strix are tools for full LTL realizability and they are based on parity games: in the general case, the *parity objective* (that ltlsynt and strix have to guarantee during the game) is more complex than the *safety objective*, which is the one used by ebr-ltl-synth.

The outcomes of the comparison between ebr-ltl-synth and ssyft are shown in Fig. 7.11. The three lines near the sides of the figure correspond to *timeouts* (the solid black line), *memouts* for unrealizable benchmarks and *memouts* for realizable benchmarks (the dotted lines). It can be noticed that ssyft reaches a memory out for the vast majority of benchmarks. For instance, on both the realizable categories, ssyft reaches the first memout with $n = 7$. As for the unrealizable benchmarks, on the third category, ssyft reaches the first memout with $n = 36$, while for the fourth category with $n = 59$. This is due to mona, which is not able to build the (explicit) DFA for the (negation of the) initial specification[2]. This is an important hint about the use of *fully symbolic* techniques for the representation of automata, like the one of ebr-ltl-synth, as in many cases they can avoid an exponential blowup of the automata' state space. The survival plot between ebr-ltl-synth and ssyft

[2]We point out that in some cases, like in the fourth category for $n \geq 60$, mona's memouts are due to its parser.

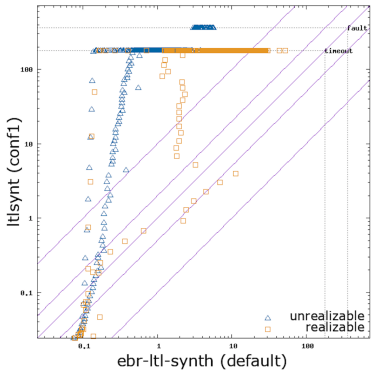


Figure 7.7: EBR-LTL-SYNTH vs LTLSYNT (first conf.) on all scalable benchmarks.

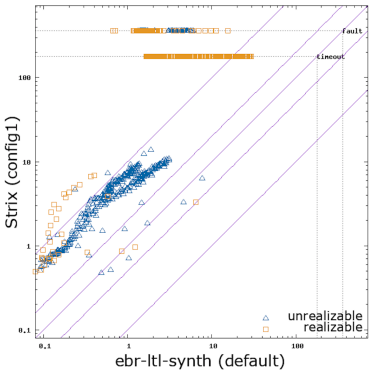


Figure 7.8: EBR-LTL-SYNTH vs STRIX on all scalable benchmarks.

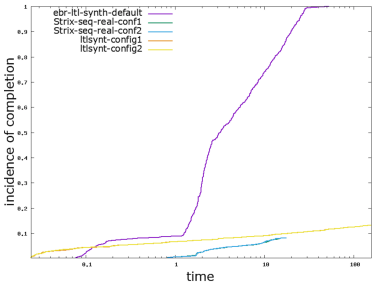


Figure 7.9: Survival plot for realizable scalable benchmarks.

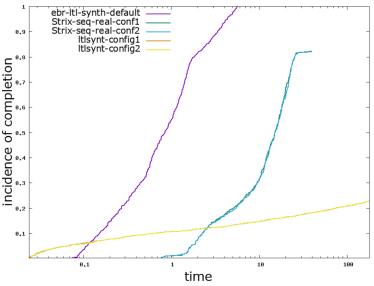


Figure 7.10: Survival plot for unrealizable scalable benchmarks.

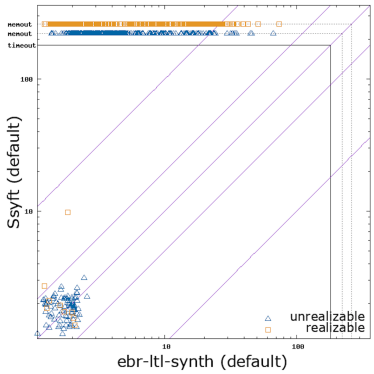


Figure 7.11: EBR-LTL-SYNTH vs SSYFT on scalable benchmarks.

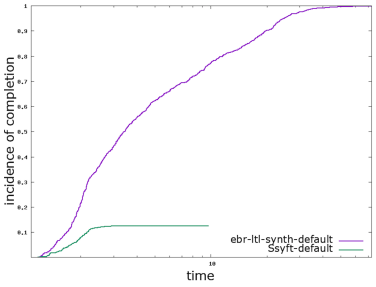


Figure 7.12: Survival plot for EBR-LTL-SYNTH and SSYFT on scalable benchmarks.

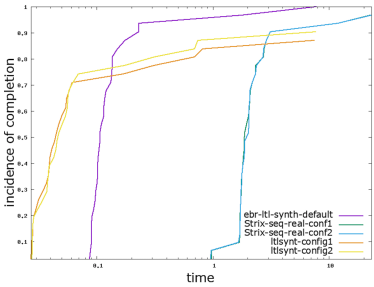


Figure 7.13: Survival plot for SYNTCOMP benchmarks.

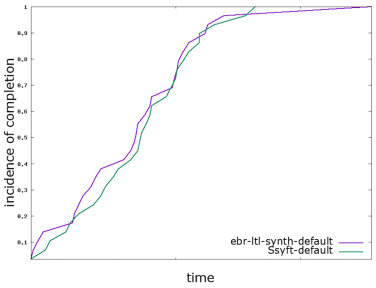


Figure 7.14: Survival plot for EBR-LTL-SYNTH and SSYFT on SYNTCOMP benchmarks.

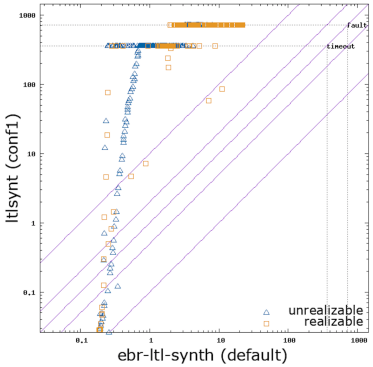


Figure 7.15: EBR-LTL-SYNTH vs LTLSYNT on normalized scalable benchmarks.

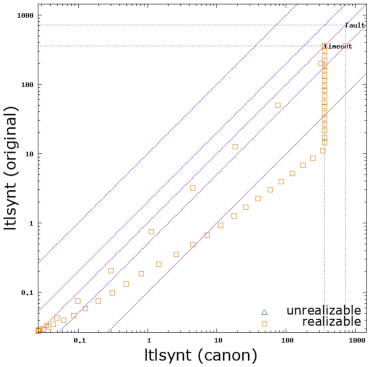


Figure 7.16: LTLSYNT on original scalable benchmarks vs LTLSYNT on normalized scalable benchmarks

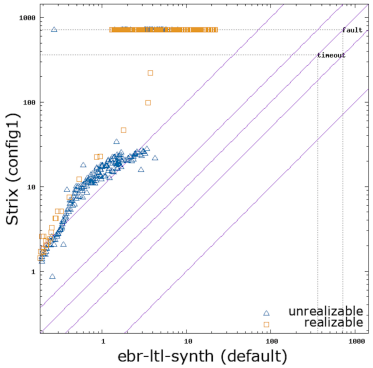


Figure 7.17: EBR-LTL-SYNTH vs STRIX on normalized scalable benchmarks.

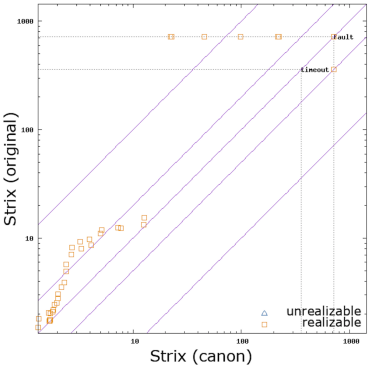


Figure 7.18: STRIX on original scalable benchmarks vs STRIX on normalized scalable benchmarks

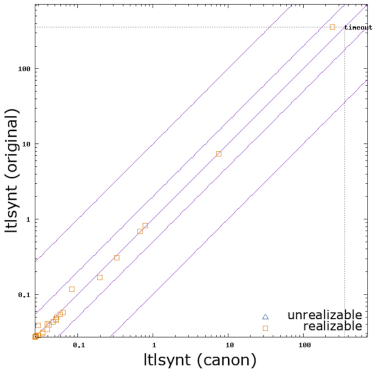


Figure 7.19: LTLSYNT on original SYNTCOMP benchmarks vs LTLSYNT on normalized SYNTCOMP benchmarks.

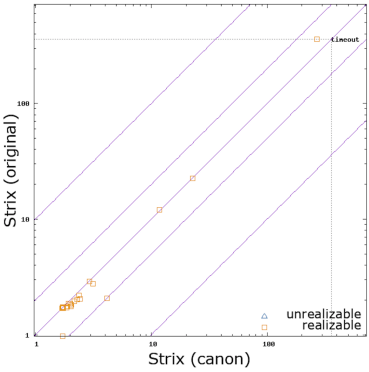


Figure 7.20: STRIX on original scalable benchmarks vs STRIX on normalized scalable benchmarks

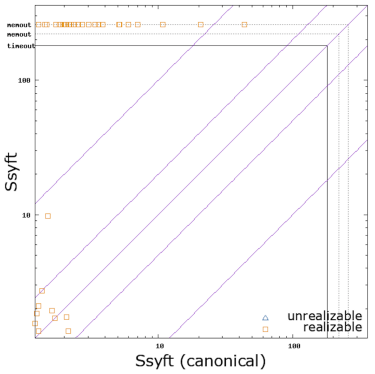


Figure 7.21: SSYFT on original scalable benchmarks vs SSYFT on normalized scalable benchmarks

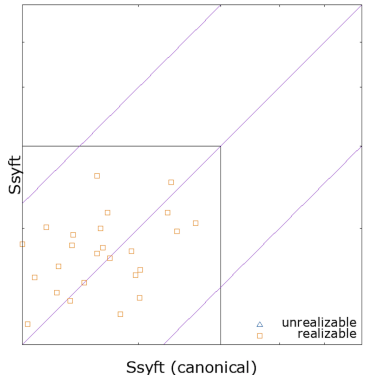


Figure 7.22: SSYFT on original SYNTCOMP benchmarks vs SSYFT on normalized SYNTCOMP benchmarks

is shown in Fig. 7.12[3].

In addition to the previous scalable benchmarks, we run the four tools on the benchmarks of the 2020 edition of the Reactive Synthesis Competition (SYNTCOMP 2020[4]) that belong to $\mathsf{LTL}_{\mathsf{EBR}}$+P, which consists of 29 benchmarks. The survival plot showing the comparison with LTLSYNT and STRIX is shown in Fig. 7.13, while the comparison with SSYFT is shown in Fig. 7.14. It is interesting to see that, on the SYNTCOMP benchmarks, the results of EBR-LTL-SYNTH and SSYFT are comparable: likely, this could depend on the simple structure of the considered SYNTCOMP benchmarks, that have only few temporal operators, in most cases *globally* (G) operators.

As for the synthesis problem, once a specification is found to be realizable, all the tools except for SSYFT produce a strategy as a witness: this strategy is in the form of an and-inverter graph whose input bits are only the starting uncontrollable variables. Often, such a strategy can be minimized by using logic synthesis tools (like ABC [32]) as black-box. In the particular case of EBR-LTL-SYNTH, LTLSYNT and STRIX, they all use a separate logic synthesizer as black box, with different configurations to minimize the strategy. Therefore, we do not compare the size of the resulting strategies, since such a comparison would add nothing about the methods implemented by the tools but would rather compare their backends. The only exception is STRIX, which, before applying the minimization, extracts the strategy in form of an *incompletely specified* Mealy machine, in the attempt to synthesize circuits of smaller size [137, 2].

7.4.1 Normalized Benchmarks

One could wonder how the transformation from $\mathsf{LTL}_{\mathsf{EBR}}$+P to *normal form* (which is represented by the first two steps in Fig. 7.1) impacts the performance and how much, instead, the good performance of our tool is due to the symbolic (and obviously deterministic) representation of the automaton. In order to answer this question, we transformed to normal form (see Definition 47) the benchmarks of

[3]The reason why we do not have a single survival plot comparing all the four tools is that SSYFT could not have been compiled for the same platform as the others, due to issues with its source code.

[4]The official website of SYNTCOMP is the following: `http://www.syntcomp.org/`

all the four scalable categories that we described above. We run the four tools on this new set of benchmarks.

Fig. 7.15 shows the comparison between EBR-LTL-SYNTH and LTLSYNT on the normalized benchmarks. First of all, we remark that the two unrealizable categories are already in normal form, while this is not true for the other two (realizable) categories. By comparing Fig. 7.7 and Fig. 7.15, we can notice that, while there is no worsening of performance of EBR-LTL-SYNTH over the already normalized unrealizable formulas, a lot of timeouts of LTLSYNT for non-normalized realizable formulas become faults on normalized formulas. By *fault* we mean an error of the tool, like a segmentation fault or a *maximum call stack size exceeded* error of the parser. For LTLSYNT (and for STRIX as well) these faults are indeed due to the parser, which is not able to deal with normalized formulas, which have a lot of nested *next* with respect to the non-normalized ones.

Fig. 7.16 shows the comparison between LTLSYNT executed on the original benchmarks and the same tool run over the normalized benchmarks. This comparison of course involves only the two realizable categories, where the non-normalized version is different from the normalized one. In the top-right corner, on the intersection between the *timeout* line and the *fault* line, there are all the formulas that reach a timeout when not normalized but result in a fault when normalized. In this plot, an interesting trend is the one of the benchmarks of the first category (which corresponds to the orange line in the middle of the plot), on which LTLSYNT reaches the first timeout with $n = 47$ on the original benchmarks but with $n = 28$ on the normalized benchmarks. Moreover, from Fig. 7.16, we clearly see an exponential worsening of the performance, when the formula is normalized.

The comparison between EBR-LTL-SYNTH and STRIX on the normalized benchmarks is shown in Fig. 7.17. Similarly, Fig. 7.18 shows the times of STRIX on the non-normalized and the normalized benchmarks. The plots confirm the fact the normalized does *not* improve the solving time of the tool, but on the contrary it worsen the performance. We can draw similar conclusions for SSYFT (see Fig. 7.21).

We transformed into normal form also the SYNTCOMP benchmarks. Among the 29 SYNTCOMP benchmarks that belong to $\mathsf{LTL}_{\mathsf{EBR}}$+P, in 4 benchmarks the *pastification* step introduces additional *past* operators to the normal form. Since the other three tools do not deal with past operators, the normalization of these 4 formu-

las could not be included in the comparison. Fig. 7.19 and Fig. 7.20 show the comparison between LTLSYNT on non-normalized and normalized SYNTCOMP benchmarks and STRIX on non-normalized and normalized SYNTCOMP benchmarks. In this case, the performance are comparable, since almost all the dots are placed on the diagonal line. Also in this case, we can make similar observations for SSYFT (see Fig. 7.22).

7.5 Conclusions

In this chapter, we focused on the realizability and reactive synthesis problems for $\mathsf{LTL}_{\mathsf{EBR}}$+P. The main contribution is a *fully symbolic* translation from any $\mathsf{LTL}_{\mathsf{EBR}}$+P formula to a *deterministic* symbolic safety automaton on infinite words. The process applies a pastification step and a set of rules to reach a normal form for $\mathsf{LTL}_{\mathsf{EBR}}$+P formulas. The realizability is then decided by solving a safety game on the arena represented by the automaton. We first showed that realizability for $\mathsf{LTL}_{\mathsf{EBR}}$+P belongs to EXPTIME. The problem is indeed EXPTIME-complete, as we proved. Then, we implemented the proposed procedure in a tool, whose experimental evaluation revealed very good performance against tools for realizability and synthesis of full LTL and Safety LTL specifications.

As a future development of this line of work, we believe that the translation from $\mathsf{LTL}_{\mathsf{EBR}}$+P to deterministic SSA may provide many benefits in the context of *symbolic model checking* as well, since the search of the state space could benefit from a deterministic representation of the automaton for the formula [173]. On the automata construction side, an interesting development would be to consider the bounded operators as primitives, without, for instance, expanding $\mathsf{X}^i\alpha$ into i nested *next* operators. Last but not least, we aim at checking whether the synthesis problem for more expressive logics, like, for instance, LTL, can be reduced to the synthesis problem for $\mathsf{LTL}_{\mathsf{EBR}}$+P, for example checking whether it is possible to use $\mathsf{LTL}_{\mathsf{EBR}}$+P for solving the safety problems originated from *bounded synthesis* techniques.

CHAPTER

8

REALIZABILITY OF GR-EBR SPECIFICATIONS

In Section 3.1 and Chapter 7, we have introduced the logic of $\mathsf{LTL}_{\mathsf{EBR}}$+P, we have shown that it allows one to express any safety language definable in LTL+P and we provided an efficient, fully-symbolic algorithm for reactive synthesis.

In this chapter, we focus on realizability of GR-EBR (*Generalized Reactivity(1)* $\mathsf{LTL}_{\mathsf{EBR}}$+P, see Section 3.3), an extension of $\mathsf{LTL}_{\mathsf{EBR}}$+P with fairness conditions, assumptions, and guarantees. The logic of GR-EBR preserves the main strength of $\mathsf{LTL}_{\mathsf{EBR}}$+P, that is, efficient realizability, and makes it possible to specify properties beyond safety. We study the problem of reactive synthesis for GR-EBR and devise a fully-symbolic algorithm that reduces it to a number of safety subproblems. To ensure soundness and completeness, we propose a general framework for safety reductions in the context of realizability of (fragments of) LTL+P. The experimental evaluation shows the feasibility of the approach.

8.1 Overview

In Section 5.3, we have seen that one of the most important problems in formal methods and requirement analysis is establishing whether a specification over a set of controllable and uncontrollable actions is *implementable* (or *realizable*), that is, whether there exists a controller that chooses the value of the controllable actions and satisfies the specification, no matter what the values of uncontrollable actions are. This problem has been formalized in the literature under the name of *realizability* [43]. The very close problem of *reactive synthesis* aims at synthesizing such a controller, whenever the specification is realizable. Usually, these problems are modelled as two-player games between Environment, who tries to violate the specification, and Controller, who tries to fulfill it. Realizability is known to have a very high worst-case complexity. In particular, it has a non-elementary lower bound for S1S specifications [36], and it is 2EXPTIME-complete for LTL+P specifications [160, 164] (see Proposition 14).

In order to apply realizability and reactive synthesis in real-world scenarios, research has focused on the identification of fragments of logics like S1S and LTL+P, with a limited expressive power, for which realizability can be solved efficiently.

A well-known example is *Generalized Reactivity(1)* logic (GR(1), for short) [25] (recall Section 5.3.8). In this fragment, a specification is syntactically partitioned into *assumptions* about the environment and *guarantees* for the controller. Both of them are either Boolean formulas (α) or safety formulas ($\mathsf{G}\alpha$) or conjunctions of recurrence formulas ($\bigwedge_{i=1}^{n} \mathsf{GF}\alpha_i$). The dichotomy between *assumptions* and *guarantees* reflects the way a system engineer usually formalizes system's requirements, which is summarized by the following sentence: *"the controller has to behave in conformance to the guarantees, under the given assumptions on the environment"*.

On a different direction, other approaches focused on safety fragments of LTL+P [201, 46]. In particular, in Section 3.1 we introduced the logic of *Extended Bounded Response* LTL+P ($\mathsf{LTL}_{\mathsf{EBR}}$+P, for short), a safety fragment of LTL+P that allows for a fully symbolic compilation of formulas into deterministic automata (recall Chapter 7). Such a feature contributes to a great improvement in solving time for the synthesis problem. Moreover, $\mathsf{LTL}_{\mathsf{EBR}}$+P has a well-established expressiveness: $\mathsf{LTL}_{\mathsf{EBR}}$+P can define exactly the

set of safety languages definable in LTL+P.

A second line of research on reactive synthesis focuses on finding good algorithms for the average case. Among these, an important class of algorithms comprises the so-called *safety reductions*, which reduce the realizability problem of the starting formula to a sequence of subproblems over *safety* formulas, by *bounding* some behaviors of the former (*e.g.*, the visits to the rejecting states of the corresponding automaton) by some integer k. The rationale behind these techniques is that a safety problem is usually much simpler to solve, since it amounts to a *strong reachability* problem [26]. This is in turn inspired by safety reductions for the model checking problem, where the validity of an LTL+P formula over a Kripke structure is reduced to checking the reachability of a given error state [57, 15]; one example is the K-Liveness algorithm (Section 5.2.6). Usually, safety reductions (both for realizability and for model checking) are: (i) *sound*, meaning that a positive answer to any of the subproblems implies a positive result of the starting formula, and (ii) *complete*, ensuring that there exists an upper bound μ such that, if the k^{th} subproblem has a negative answer for all $k \leq \mu$, then also the result of the starting formula is negative. Important examples of safety reductions for realizability are the works on *bounded synthesis* [92] (recall Section 5.3.6), which are based on the pioneering work on *Safraless algorithms* of [128], and all the different encodings proposed for solving it [88].

8.1.1 Contributions

The main contributions of this chapter are the following ones. First, we devise a novel framework for deriving *complete* safety reductions in the context of realizability of (fragments of) LTL+P. A notable feature of the framework is that it provides a link to safety reductions for the model checking problem and proves that if a reduction is complete for model checking, then it is also complete for realizability. On one hand, this allows one to reason on Kripke structures (Definition 31) instead of on strategies (Definition 41), which is simpler; on the other hand, it enables the use of some reductions already exploited in model checking for realizability, provided that they conform to the framework.

Second, the proposed framework is used to derive a *complete* safety reduction for the realizability problem of GR-EBR. We recall

that a GR-EBR formula is of type:

$$(\psi^1_{ebr} \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\psi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

for some $m, n \in \mathbb{N}$, $\psi^1_{ebr}, \psi^2_{ebr} \in \mathsf{LTL_{EBR}}{+}\mathsf{P}$ and $\alpha_i, \beta_j \in \mathsf{LTL}{+}\mathsf{P_P}$, for each $i, j \in \mathbb{N}$. We refer to Definition 23 for more details.

We use such a reduction as the core of an algorithm for GR-EBR realizability, which, at each iteration, builds a safety sub-problem and checks for its realizability. If it returns a positive result, then the initial formula is realizable as well; otherwise, it continues with the next iteration. If the upper bound given by the reduction has been reached, the algorithm outputs the unrealizability of the initial formula. As a matter of fact, the upper bound is doubly exponential in the size of the formula and thus prohibitively large. For this reason, in practice, it is useful to use the algorithm in *parallel* with another one checking for the unrealizability of the formula. The first that terminates stops the other and, thus, the entire procedure. A crucial property of the algorithm is that the realizability check of each safety sub-problem is performed in a *fully symbolic* way, thus retaining the distinctive feature of $\mathsf{LTL_{EBR}}{+}\mathsf{P}$.

Last but not least, we provide an implementation of the algorithm as a prototype tool called GRACE (*GR-ebr reAlizability ChEcker*). The experimental evaluation shows good performance against tools for full LTL+P synthesis.

8.1.2 Related work

GR(1) (*Generalized Reactivity(1)*) has been introduced in [158, 25] (recall Sections 2.4.5 and 5.3.8). It is known that GR(1) is a good candidate for writing specifications of real-world scenarios, with a relatively low complexity: the realizability problem can be solved with at most a quadratic number of symbolic steps in the size of the specification [25]. On the other hand, GR(1) presents some restrictions that limit its use as a specification language: we refer to Section 3.3.3 to a detailed comparison between the expressive power of GR(1) and GR-EBR.

The (*strict*) realizability problem for GR(1) is solved in [25] by building a *symbolic* arena (called *game structure*) and by solving a fixpoint computation over it, that requires $\mathcal{O}(N^2)$ symbolic op-

erations, where N is the size of the specification (we refer to Section 5.3.8 for more details). In this chapter, we follow a different approach based on a reduction to safety, that generates a sequence of deterministic safety automata over which the corresponding game can be solved with at most a *linear* number of symbolic steps.

In [153], Morgenstern and Schneider identify a syntactical fragment of LTL+P, whose formulas correspond to deterministic Büchi automata. The fragment is defined in such a way that it corresponds to the temporal hierarchy defined in [140] (as a matter of fact, each formula of GR-EBR can be transformed into an equivalent one of that fragment by expanding bounded operators). Realizability is solved by exploiting known algorithms like subset construction and Miyano-Hayashi breakpoint construction for the determinization of the automata. On the contrary, the compilation of GR-EBR to automata is fully symbolic, which has been proved in [46] to be a key point for performance, compared to classical algorithms for determinization.

Recall from Section 5.3.6 that *Bounded synthesis* [92, 88] belongs to the class of *Safraless techniques* [128] (Section 5.3.5), and it consists in bounding the number of times Controller is forced to visit a rejecting state of a Universal co-Büchi automaton (UCA, for short) for the initial formula. This corresponds to a safety automaton, which can be either (i) made deterministic by a suitable generalization of the classical subset construction [90, 81], or (ii) encoded into a constraint system [92, 88] (*e.g.*, SAT- or SMT-based) which bounds also the *size* of a candidate controller (this also allows one to tackle undecidable problems, for instance in the case of distributed or parametric synthesis). Both choices work for the whole class of UCA, and thus for full LTL+P. A significant drawback of such an approach is that the UCA, which can be exponentially larger than the initial specification, is explicitly represented. Moreover, in the first case, the algorithm for the determinization turns out to be quite complex, since each state of the resulting automaton is actually a *function*. This can also result into a very large state space, that can be tackled by exploiting either *antichains* [90] or BDDs [81]. In contrast, as we will see, we define a reduction tailored to GR-EBR formulas that allows us to exploit the $\mathsf{LTL}_{\mathsf{EBR}}$+P transformations introduced in [46] for a *fully symbolic* mapping of the initial formula directly into a sequence of symbolic safety automata. In particular, we never build any explicit-state automaton and we avoid the

subsequent use of determinization algorithms.

Organization

The rest of the chapter is organized as follows. The framework for deriving complete reductions is presented in Section 8.2. In Section 8.3 we describe the algorithm for the realizability of GR-EBR specifications. The outcomes of the experimental evaluation are reported in Section 8.4. Finally, in Section 8.5, we point out some interesting future research directions.

8.2 A Framework of Safety Reductions for Realizability

The central question of this section is: *how can we obtain a complete safety reduction for the realizability problem of specifications written in (fragments of)* LTL+P*?* In the following, we propose a framework to answer it.

8.2.1 A sound but not complete safety reduction

Devising a sound and complete safety reduction for realizability is not a trivial task. Consider for example LTL+P. One could be tempted to define a safety reduction that, given any formula $\phi \in \mathsf{LTL{+}P}$, turns ϕ in *negated normal form* (NNF, for short) and then transforms each F and U into the corresponding k-bounded operator, that is $\mathsf{F}^{[0,k]}$ and $\mathsf{U}^{[0,k]}$, respectively, obtaining a safety formula (see Section 2.4.4). The resulting reduction would be sound, since the resulting formula *implies* the starting one, but it would not be complete. Consider for example the formula $\phi \coloneqq \mathsf{G}u \leftrightarrow \mathsf{G}c$. The formula obtained by means of this safety reduction is:

$$\phi_k \coloneqq (\mathsf{F}^{[0,k]}\neg u \vee \mathsf{G}c) \wedge (\mathsf{F}^{[0,k]}\neg c \vee \mathsf{G}u)$$

Although ϕ is realizable, ϕ_k is unrealizable for each $k \in \mathbb{N}$. In fact, Environment can choose u in the first k steps and $\neg u$ in the step $k+1$: in this way, it falsifies both $\mathsf{F}^{[0,k]}\neg u$ and $\mathsf{G}u$. Since Controller can force exactly one between the formulas $\mathsf{G}c$ and $\mathsf{F}^{[0,k]}\neg c$, we have that ϕ_k is unrealizable, and therefore this particular safety reduction is not complete for realizability.

8.2.2 Definition of safety reduction

The core and the main novelty of our framework is a link with safety reductions for model checking: in order to design a complete reduction for the realizability problem, one can prove that it is complete for the model checking problem and then use our framework to derive completeness for realizability. On one hand, this allows to prove completeness at the level of model checking, which is simpler than proving completeness for realizability. For example, reasoning over *infinite paths* is in general simpler than reasoning over *strategies* and, thus, *sets* of infinite paths. On the other hand, this opens the possibility of using existing safety reductions already devised for model checking for realizability as well. We start by defining what is a safety reduction in the context of our framework.

Definition 53 (Safety reduction)**.** *Let $\mathcal{S} \subseteq \mathsf{LTL{+}P}$ be a fragment of $\mathsf{LTL{+}P}$. A* safety reduction for $\mathcal{S}$ *is a function $[\![\cdot]\!]$ such that, for each formula $\phi \in \mathcal{S}$ over the alphabet Σ, it holds that $[\![\phi]\!] = \{\phi_k\}_{k \in \mathbb{N}}$, where ϕ_k is a* safety *formula over the alphabet Σ such that $\phi_k \rightarrow \phi$, for any $k \in \mathbb{N}$. With $[\![\phi]\!]^k$, we will denote the formula ϕ_k of the set above.*

8.2.3 Link between realizability and model checking

The rationale behind the link between realizability and model checking is the following one: since we can easily view Mealy machines (Definition 44) as a particular type of Kripke structures (Definition 31) and viceversa, and since by the Finite Model Property of $\mathsf{LTL{+}P}$ realizability (Proposition 12) we can restrict realizability to the search of finite strategies representable by Mealy machines, the realizability problem of the $\mathsf{LTL{+}P}$ formula ϕ can be reduced to checking if there exists a Mealy machine M_g such that $M'_g \models \mathsf{A}\,\phi$, where M'_g is the Kripke structure *corresponding to* M_g.

The Kripke structure M'_g *corresponding to* the Mealy machine $M_g = (2^{\mathcal{U}}, 2^{\mathcal{C}}, Q, q_0, \delta)$ is defined as $M'_g = (2^{\mathcal{U} \cup \mathcal{C}}, Q', I', T', L')$ where:

1. $Q' = Q \times \{q_U \mid U \in 2^{\mathcal{U}}\} \times \{q_C \mid C \in 2^{\mathcal{C}}\}$;
2. $I' = \{(q_0, q_U, q_C) \in Q' \mid \delta(q_0, U) = (C, q')$ for any $U \in 2^{\mathcal{U}}, C \in 2^{\mathcal{C}}$ and $q' \in Q\}$,

3. $T' = \{((q, q_U, q_C), (q', q_{U'}, q_{C'})) \mid \delta(q, U) = (C, q')$ for any $U, U' \in 2^{\mathcal{U}}, C, C' \in 2^{\mathcal{C}},$ and $q, q' \in Q'\}$ and
4. $L'((q, q_U, q_C)) = U \cup C$.

The Kripke structure M'_g is such that each trace of M'_g corresponds to a word of M_g, and viceversa.

In proving the completeness theorem, we will abstract from the concrete safety reduction and give the conditions for a general safety reduction $[\![\cdot]\!]$ (as defined in Definition 53) to be complete. These conditions are formalized in Definition 54.

Definition 54 (Sound and Complete safety reduction)**.** *Let $\mathcal{S} \subseteq$ LTL+P be a fragment of* LTL+P*, ϕ a formula in S, and $[\![\cdot]\!]$ a* safety reduction for $\mathcal{S}$*. We say that $[\![\cdot]\!]$ is* μ-complete*, for a given function $\mu : \mathbb{N} \to \mathbb{N}$ if and only if, for all $\phi \in \mathcal{S}$ and for all Kripke structures M:*

$$M \models \mathsf{A}\,\phi \quad \Leftrightarrow \quad \exists k \leq \mu(|M|)\ .\ M \models \mathsf{A}[\![\phi]\!]^k$$

Example Let $[\![\cdot]\!]_{bo}$ and $[\![\phi]\!]^k_{bo}$ be the reduction and the formula ϕ_k described in Section 8.2.1, respectively (*bo* stands for *bounded operators*). Since $[\![\phi]\!]^k_{bo}$ contains only Boolean operators, the *next* temporal operator (coming from the expansion of the bounded operators) and universal temporal operators (*i.e.*, G and R), it is a safety formula [177] and thus $[\![\cdot]\!]_{bo}$ is a safety reduction according to Definition 53. We have already seen that the reduction is sound but not complete for realizability. According to our framework, this reduction is not even complete for the model checking problem (*i.e.*, with respect to Definition 54), and indeed a counterexample can be found on Figure 1 in [57].

We can finally state the main theorem of our framework, which uses Definition 54 and Proposition 12 in order to establish that if a safety reduction is complete for the model checking problem, then it is complete for the realizability problem as well.

Theorem 40 (Soundness and Completeness for LTL+P Realizability)**.** *Let $\mathcal{S} \subseteq$ LTL+P be a fragment of* LTL+P*, $\phi \in \mathcal{S}$ a formula over the input alphabet $\mathcal{U}$ and output alphabet $\mathcal{C}$ (with $n = |\phi|$) and $[\![\cdot]\!]$ a μ-complete safety reduction for $\mathcal{S}$, for a given function μ. It holds that:*

$$\phi \text{ is realizable} \quad \Leftrightarrow \quad \exists k \leq \mu(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}})\ .\ [\![\phi]\!]^k \text{ is realizable}$$

Proof. We first prove the *soundness*, which corresponds to the right-to-left direction. Suppose there exist a $k \leq \mu(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}})$ such that $[\![\phi]\!]^k$ is realizable. Then, there exists a strategy $g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ such that $\mathcal{L}(g) \subseteq \mathcal{L}([\![\phi]\!]^k)$. By Proposition 12, there exists a Mealy machine $M_g = (2^{\mathcal{U}}, 2^{\mathcal{C}}, Q, q_0, \delta)$ with input alphabet $2^{\mathcal{U}}$ and output alphabet $2^{\mathcal{C}}$ such that $\mathcal{L}(M_g) \subseteq \mathcal{L}([\![\phi]\!]^k)$. Starting from M_g, let $M'_g = (2^{\mathcal{U} \cup \mathcal{C}}, Q', I', T', L')$ be the *corresponding* Kripke structure. The Kripke structure M'_g is such that each trace of M'_g corresponds to a word of M_g, and viceversa. Therefore all the traces π of M'_g are such that $L'(\pi) \models [\![\phi]\!]^k$, that is $M'_g \models \mathsf{A}[\![\phi]\!]^k$. Since by hypothesis $[\![\cdot]\!]$ is a μ-complete safety reduction, by Definition 54, it holds that $M'_g \models \mathsf{A}\,\phi$. This means that also $\mathcal{L}(M_g) \subseteq \mathcal{L}(\phi)$. Since M_g is a Mealy machine, this implies that ϕ is realizable.

We now prove *completeness*, which corresponds to the left-to-right direction. Suppose that ϕ is realizable. Since $\phi \in S$ and since $S \subseteq \mathsf{LTL}{+}\mathsf{P}$, ϕ is an $\mathsf{LTL}{+}\mathsf{P}$ formula as well. Therefore, by Proposition 12, there exists a Mealy machine M_g with input alphabet $2^{\mathcal{U}}$ and output alphabet $2^{\mathcal{C}}$ such that $\mathcal{L}(M_g) \subseteq \mathcal{L}(\phi)$ with at most $2^{2^{c \cdot n}}$ states, for some constant $c \in \mathbb{N}$. From M_g, we build an equivalent Kripke structure M'_g with input alphabet $\Sigma' = 2^{\mathcal{U} \cup \mathcal{C}}$, as described above for the soundness proof. It holds that $M'_g \models \mathsf{A}\,\phi$. Since by hypothesis $[\![\cdot]\!]$ is a μ-complete safety reduction for S, and since $|Q'| = 2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot |Q|$ (where Q and Q' are the set of states of M_g and M'_g, respectively), by Definition 54, there exists a $k \leq \mu(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}})$ such that $M'_g \models \mathsf{A}[\![\phi]\!]^k$. This means that also $\mathcal{L}(M_g) \subseteq \mathcal{L}([\![\phi]\!]^k)$. Since M_g is a Mealy machine, this means that there exists a $k \leq \mu(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}})$ such that $[\![\phi]\!]^k$ is realizable. □

Novelty and Usage

As already mentioned before, a distinguished and important feature of our framework is that it provides a link with safety reductions for the *model checking problem*. This opens the possibility to use model checking safety reductions for the realizability problem as well, provided that the reduction fulfills the requirements in Definition 54. In the next sections, we will define a *concrete* safety reduction for GR-EBR specifications that is *complete* with respect to Definition 54, and we will use it for introducing a novel algorithm for GR-EBR realizability. Using Theorem 40, we will derive a corollary for the completeness of our algorithm.

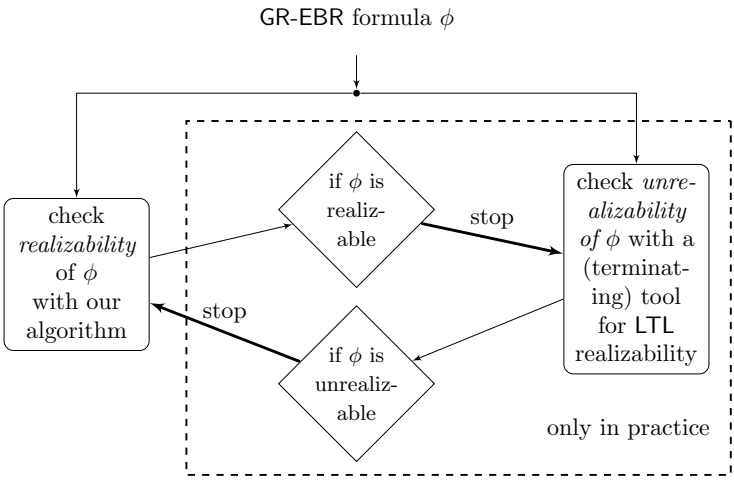


Figure 8.1: High-level view of our procedure for the realizability of GR-EBR formulas.

In practice

The upper bound for the value of $\mu(\cdot)$ (after which we can answer *unrealizable*) is doubly exponential in the size of the initial formula and therefore, in practice, it is prohibitively large. It follows that usually the *completeness* of a safety reduction can be exploited in practice only for making sure that, starting from a *realizable* specification, we will eventually find a $k \in \mathbb{N}$ such that the k^{th} subproblem is realizable. Therefore, like K-Liveness for model checking [57] (recall Section 5.2.6), we can use our algorithm in parallel with another one that checks for the unrealizability of the specification. The first that terminates stops the other and, thus, the entire procedure. The high-level view of our procedure, which we will detail in the next section, is summarized in Fig. 8.1. We remark that we cannot check the unrealizability of ϕ by solving the dualized game (*i.e.*, looking for a Moore-type strategy of Environment) for $\neg\phi$, because GR-EBR and $\mathsf{LTL}_{\mathsf{EBR}}$+P are *not* closed under complementation.

8.2.4 Formalization of Bounded Synthesis

It is worth noting that also *bounded synthesis* techniques [92] can be formalized in our framework. This family of techniques works for the entire class of Universal co-Büchi Word automata (UCA, for short), which subsumes LTL+P. However, in this part, we focus only on bounded synthesis applied on LTL+P formulas. Given an LTL+P formula ϕ, bounded synthesis algorithms build the Büchi automaton $\mathcal{A}_{\neg\phi}$ for the negation of ϕ (eq., the Universal co-Büchi automaton for ϕ) and *bound* the number of times Controller player is forced to visit the set of final states of the Büchi automaton $\mathcal{A}_{\neg\phi}$ (eq., the set of rejecting states of the Universal co-Büchi automaton for ϕ). Consider the safety automaton $\mathcal{A}_k$ defined as $\mathcal{A}_{\neg\phi} \times C_k$, where C_k is the safety automaton corresponding to a *counter* that increments if it visits a final state of $\mathcal{A}_{\neg\phi}$, retains its value otherwise, and whose *safe* states are all those states where the counter has value less than k. We define the set of safe states of $\mathcal{A}_k$ as the set of final states of C_k. We can formalize bounded synthesis in our framework by means of a safety reduction, that we call $[\![\cdot]\!]_{bs}$, defined as follows: for any LTL+P formula ϕ and for any $k \in \mathbb{N}$, we define $[\![\phi]\!]^k_{bs}$ to be any safety formula (not necessarily of LTL+P) such that $\mathcal{L}([\![\phi]\!]^k_{bs}) = \mathcal{L}(\mathcal{A}_k)$. Note that, since $\mathcal{A}_k$ is a safety automaton, $[\![\phi]\!]^k_{bs}$ is a safety formula as well (Section 2.4.4) and thus $[\![\cdot]\!]_{bs}$ is a safety reduction (Definition 53). This reduction is also *complete* with respect to Definition 54, as proved by the theorem below. An interesting feature is that this proof, which reasons on model checking (thanks to Definition 54), amounts to prove the completeness of the K-Liveness algorithm (Section 5.2.6, [57]), which is a simple but very efficient algorithm for model checking based on safety reductions. From now, with $\mathsf{id} : \mathbb{N} \to \mathbb{N}$ we denote the *identity* function.

Theorem 41. *The safety reduction* $[\![\cdot]\!]_{bs}$ *is* id*-complete.*

By using the main theorem of our framework (Theorem 40), we can prove the completeness of bounded synthesis for LTL+P [92, 88]. We remark that bounded synthesis works for the full class of UCA and that this is the reason why here we obtain a smaller upper bound with respect to [92, 90].

Corollary 10 (Completeness of Bounded Synthesis)**.** *Let* ϕ *be an* LTL+P *formula over the set of variables* $\Sigma = \mathcal{U} \cup \mathcal{C}$*. It holds that* ϕ

is realizable if and only if there exists $k \leq 2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}}$ *such that* $[\![\phi]\!]^k_{bs}$ *is realizable.*

In this paper, we do not consider the $[\![\cdot]\!]_{bs}$ safety reduction, which corresponds to bounded synthesis and works with full LTL+P, because we do not know yet if there exists a fully symbolic translation from any $[\![\phi]\!]^k_{bs}$ to a safety automaton. Since we aim at using *symbolic* techniques and since for $\mathsf{LTL}_{\mathsf{EBR}}$+P there exists a fully symbolic procedure for obtaining an equivalent deterministic automaton (Chapter 7, [46]), we focus on fragments of LTL+P for which we can use $\mathsf{LTL}_{\mathsf{EBR}}$+P for this task. As we will see later in Section 8.3, GR-EBR is one of these.

8.3 A Safety Reduction for GR-EBR

In this section, we describe the algorithm for solving realizability of GR-EBR specifications. It consists in three steps. Firstly, we build the product between the two symbolic and safety automata for the safety ($\mathsf{LTL}_{\mathsf{EBR}}$+P) parts of both assumptions and guarantees. This product automaton has a GR(1) accepting condition (Definition 39), that is of the form $\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$, where each α_i and β_j belongs to $\mathsf{LTL}{+}\mathsf{P}_{\mathsf{P}}$. The second step consists in a so-called *degeneralization*, that, by using deterministic monitors, turns the GR(1) accepting condition into a R(1) (Reactivity(1), for short) condition (Definition 39), that is of the form $\mathsf{GF}\alpha \rightarrow \mathsf{GF}\beta$, where $\alpha, \beta \in \mathsf{LTL}{+}\mathsf{P}_{\mathsf{P}}$. The third and last step, that is the core of the procedure, reduces the realizability problem over the above automaton to a *sequence* of safety synthesis problems, that is, realizability problems over safety (and symbolic) automata $\mathcal{A}^{\mathrm{k}}_{safe}$, one for each index $k \in \mathbb{N}$. By introducing a *concrete* safety reduction $[\![\cdot]\!]_{ebr}$ for GR-EBR, and by proving that it is *complete* with respect to Definition 54, we prove the completeness of the entire procedure. The structure of the full procedure is depicted in Fig. 8.2.

At each step, our algorithm checks if $\mathcal{A}^{\mathrm{k}}_{safe}$ is realizable, by solving a safety game. If this is the case, then, by Theorem 40, ϕ is realizable as well. Otherwise, we increment k and continue with the next iteration. Since $[\![\cdot]\!]_{ebr}$ is a *complete* safety reduction, by Theorem 40, in the case ϕ is realizable, we will eventually find a $k \in \mathbb{N}$ such that $\mathcal{A}^{\mathrm{k}}_{safe}$ is realizable. If instead ϕ is unrealizable, then by the same theorem: (i) in theory, we will eventually reach the upper

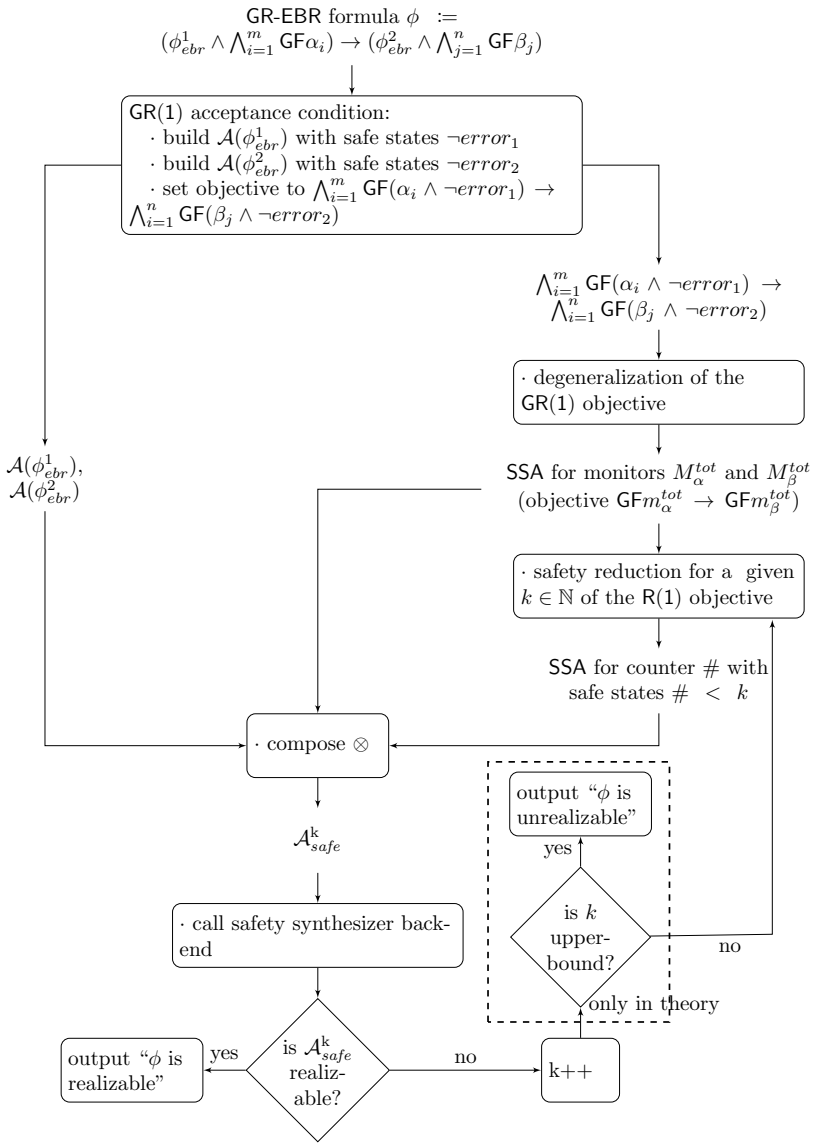


Figure 8.2: Low-level view of the procedure for the realizability of GR-EBR formulas.

bound of $[\![\cdot]\!]_{ebr}$, (ii) or, in practice, the algorithm in parallel will eventually terminate, stopping the entire procedure. Crucially, at each step $k \in \mathbb{N}$, the realizability check for $\mathcal{A}^{\mathrm{k}}_{safe}$ is performed in a *fully symbolic* way.

Finally, note that, as for now, there is no incrementality between an iteration and the next one, because of the lack of incremental safety synthesizers. The only point that we save between one iteration and the next one is the construction of the two symbolic safety automata, which is performed only once during the procedure.

8.3.1 The automaton with the GR(1) condition

In this part, we describe the first step of the algorithm. Starting from a GR-EBR formula $\phi := (\phi^1_{ebr} \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\phi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$, the objective is to obtain an automaton $\mathcal{A}$ such that: (i) it has a GR(1) accepting condition, that is of the form $\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$, and (ii) it recognizes the same language of ϕ, *i.e.*, $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A})$. In order to do that, we first build the two symbolic safety automata for the safety parts of both the assumptions and the guarantees, that is for ϕ^1_{ebr} and ϕ^2_{ebr}. Since by definition both are $\mathsf{LTL_{EBR}}$+P formulas, we use the transformation described in Section 7.2, to which the reader is referred for more details.

From now on, let $\mathcal{A}(\phi^1_{ebr})$ and $\mathcal{A}(\phi^2_{ebr})$ be the automata for ϕ^1_{ebr} and ϕ^2_{ebr}, respecively. Let $\mathcal{A}_{\phi_{\mathrm{ebr}}}$ be the product automaton $\mathcal{A}(\phi^1_{ebr}) \times \mathcal{A}(\phi^2_{ebr})$. The question is how to set the acceptance condition of $\mathcal{A}_{\phi_{\mathrm{ebr}}}$ such that the conditions (i) and (ii) of above are fulfilled. We answer this question by examining how the automata $\mathcal{A}(\phi^1_{ebr})$ and $\mathcal{A}(\phi^2_{ebr})$ are made internally. Take for example the formula $\mathsf{G}p$ (for some atomic proposition $p \in \Sigma$). The safety automaton corresponding to this formula comprises an *error bit* as one of its state variables, let us call it `error`, which is initially set to be `false`. The transition function for `error` is deterministic and updates `error` to `true` if $\neg p$ holds in the current state, or keeps its value otherwise. The set of safe states comprises all and only those states in which `error` is `false`. In a symbolic setting, this is expressed by the formula $\mathsf{G}\neg$`error`. In this way, p is forced to hold constantly in all (and only) the words accepted by the automaton.

A crucial property of each error bit is *monotonicity*: once `error` is set to `true`, it can never be set to `false` again. Formally, given a trace τ of the automaton, it holds that, if there exists $i \geq 0$ such

that $\tau(i) \models \texttt{error}$, then $\tau(j) \models \texttt{error}$, for all $j \geq i$. Monotonicity of the error bits allows us to express an accepting condition of type $\mathsf{G}\neg\texttt{error}$ in terms of $\mathsf{GF}\neg\texttt{error}$, by maintaining the equivalence.

Lemma 20 (Monotonicity of Error Bits)**.** *Each error bit is monotone.*

Proof. Consider a trace τ of an automaton with an accepting condition of the type $\mathsf{G}\neg\texttt{error}$. If $\tau \models \mathsf{G}\neg\texttt{error}$ then of course $\tau \models \mathsf{GF}\neg\texttt{error}$. Suppose now that $\tau \models \mathsf{GF}\neg\texttt{error}$. If by contradiction we suppose that $\tau \not\models \mathsf{G}\neg\texttt{error}$, we have that there exists an $i \geq 0$ such that $\tau(i) \models \texttt{error}$. By the monotonicity property, this would mean that also $\tau(j) \models \texttt{error}$, for all $j \geq i$, that is $\tau \models \mathsf{FG}\texttt{error}$, but this a contradiction with out hypothesis. Therefore, we proved that changing the acceptance condition of an automaton from a $\mathsf{G}\neg\texttt{error}$ to $\mathsf{GF}\neg\texttt{error}$ maintains the equivalence. □

Let $\texttt{error}_1$ and $\texttt{error}_2$ be the error bits of $\mathcal{A}(\phi^1_{ebr})$ and $\mathcal{A}(\phi^2_{ebr})$, respectively. Let $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$ be the automaton obtained from $\mathcal{A}_{\phi_{\text{ebr}}}$ by replacing its acceptance condition with the following $\mathsf{GR}(1)$ condition:

$$(\mathsf{GF}\neg\texttt{error}_1 \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\mathsf{GF}\neg\texttt{error}_2 \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j) \tag{8.1}$$

The intuition is that $\texttt{error}_1$ and $\texttt{error}_2$ keep track of the *safety* parts of ϕ, that is ϕ^1_{ebr} and ϕ^2_{ebr}. The following lemma proves the equivalence between ϕ and $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$.

Lemma 21. *Let ϕ be an* GR-EBR *formula. It holds that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}})$.*

Proof. Let $\phi \in$ GR-EBR. ϕ is of the following form:

$$(\phi^1_{ebr} \rightarrow \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\phi^2_{ebr} \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

By the theorems proved in [46], it holds that $\mathcal{L}(\phi^1_{ebr}) = \mathcal{L}(\mathcal{A}(\phi^1_{ebr}))$ and $\mathcal{L}(\phi^2_{ebr}) = \mathcal{L}(\mathcal{A}(\phi^2_{ebr}))$.

Consider first the left-to-right direction. Let $\sigma \in \mathcal{L}(\phi)$. We prove that $\sigma \in \mathcal{L}(\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}})$. Each $\sigma \in \mathcal{L}(\phi)$ is such that: a. either $\sigma \models \neg\phi^1_{ebr} \vee \neg(\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i)$, b. or $\sigma \models \phi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$ Recall that

$\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$ is defines as the product automaton $\mathcal{A}(\phi_{ebr}^1) \times \mathcal{A}(\phi_{ebr}^2)$ with the acceptance condition α defined as $(\mathsf{GF}\neg\mathtt{error}_1 \wedge \bigwedge_{i=1}^m \mathsf{GF}\alpha_i) \rightarrow (\mathsf{GF}\neg\mathtt{error}_1 \wedge \bigwedge_{j=1}^n \mathsf{GF}\beta_j)$.

Consider case *a*. If $\sigma \models \neg\phi_{ebr}^1 \vee \neg(\bigwedge_{i=1}^m \mathsf{GF}\alpha_i)$, then *the* trace induced by σ in $\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$ is such that at least one of the following two cases hold:

a.1. either $\exists i \geq 0$ such that $\tau(i) \models \mathtt{error}_1$, that is $\tau \models \mathsf{F}(\mathtt{error}_1)$. In this case, we exploit *monotonicity* of $\mathtt{error}_1$. Since $\tau \models \mathsf{F}(\mathtt{error}_1)$, it also holds that $\tau \models \mathsf{FG}(\mathtt{error}_1)$, that is $\tau \not\models \mathsf{GF}(\neg\mathtt{error}_1)$. As a consequence, $\tau \models \alpha$, where α is the acceptance condition of $\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$, and thus $\sigma \in \mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge})$.

a.2. or $\tau \models \neg\bigwedge_{i=1}^m \mathsf{GF}\alpha_i$. In this case, of course, $\tau \models \alpha$ (that is, τ satisfies the acceptance condition of $\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$), and thus $\sigma \in \mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge})$.

Consider now the case *b*. If $\sigma \models \phi_{ebr}^2 \wedge \bigwedge_{j=1}^n \mathsf{GF}\beta_j$, then $\sigma \models \phi_{ebr}^2$ and $\sigma \models \bigwedge_{j=1}^n \mathsf{GF}\beta_j$. Therefore, *the* trace induced by σ in $\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$ is such that $\tau \models \mathsf{G}(\neg\mathtt{error}_2) \wedge \bigwedge_{j=1}^n \mathsf{GF}\beta_j$, that implies that $\tau \models \mathsf{GF}(\neg\mathtt{error}_2) \wedge \bigwedge_{j=1}^n \mathsf{GF}\beta_j$. Therefore, $\tau \models \alpha$, and thus $\sigma \in \mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge})$.

We now prove the opposite direction. Suppose that $\sigma \in \mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge})$. We prove that $\sigma \in \mathcal{L}(\phi)$. By definition of $\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$, it holds that *the* trace τ induced by σ in $\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$ is such that $\tau \models (\mathsf{GF}\neg\mathtt{error}_1 \wedge \bigwedge_{i=1}^m \mathsf{GF}\alpha_i) \rightarrow (\mathsf{GF}\neg\mathtt{error}_2 \wedge \bigwedge_{j=1}^n \mathsf{GF}\beta_j)$. We divide in cases:

c. either $\tau \models \mathsf{FG}\mathtt{error}_1 \vee \neg\bigwedge_{i=1}^m \mathsf{GF}\alpha_i$,

d. or $\tau \models \mathsf{GF}\neg\mathtt{error}_2 \wedge \bigwedge_{j=1}^n \mathsf{GF}\beta_j$.

Consider case *c*. We divide again in cases:

c.1 if $\tau \models \mathsf{FG}\mathtt{error}_1$, even more so it holds that $\tau \models \mathsf{F}\mathtt{error}_1$, that is $\tau \models \neg\mathsf{G}\neg\mathtt{error}_1$, and therefore $\sigma \models \neg\phi_{ebr}^1$ and $\sigma \in \mathcal{L}(\phi)$.

c.2 if instead $\tau \models \neg\bigwedge_{i=1}^m \mathsf{GF}\alpha_i$, then of course $\sigma \models \neg\bigwedge_{i=1}^m \mathsf{GF}\alpha_i$, and $\sigma \in \mathcal{L}(\phi)$.

Finally, consider the case *d*. If $\phi \models \mathsf{GF}\neg\mathtt{error}_2 \wedge \bigwedge_{j=1}^n \mathsf{GF}\beta_j$, then $\tau \models \mathsf{GF}\neg\mathtt{error}_2$ and $\tau \models \bigwedge_{j=1}^n \mathsf{GF}\beta_j$. Since $\tau \models \mathsf{GF}\neg\mathtt{error}_2$ and thanks to the *monotonicity* of $\mathtt{error}_2$, it can *not* exists an $i \geq 0$ such that $\tau(i) \models \mathtt{error}_2$ (otherwise $\tau \models \mathsf{FG}\mathtt{error}_2$, that is

$\tau \models \neg\mathsf{GF}\neg\texttt{error}_2$, but this is a contradiction with our hypothesis). Therefore, for all $i \geq 0$ it holds that $\tau(i) \models \neg\texttt{error}_2$, that is $\tau \models \mathsf{G}(\neg\texttt{error}_2)$. Since τ is a trace induced by σ in $\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$, it follows that $\sigma \models \phi_{ebr}^2$. Since it also holds that $\sigma \models \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$, we have that $\sigma \in \mathcal{L}(\phi)$. □

Optimization Instead of considering the acceptance condition described in Eq. (8.1), we propose to repeat the error bits inside each fairness condition.

$$(\bigwedge_{i=1}^{m} \mathsf{GF}(\alpha_i \wedge \neg\texttt{error}_1)) \rightarrow (\bigwedge_{j=1}^{n} \mathsf{GF}(\beta_j \wedge \neg\texttt{error}_2)) \tag{8.2}$$

This may be helpful during the safety game solving, since it creates a redundancy that the solver can exploit during the search. Obviously, this maintains the equivalence.

8.3.2 Degeneralization

The objective of this part is to transform the $\mathsf{GR}(1)$ accepting condition of the automaton $\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge}$, that is of the form $\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$, into a condition of the form $\mathsf{GF}\alpha \rightarrow \mathsf{GF}\beta$ (also called *Reactivity(1)* objective, $\mathsf{R}(1)$, for short). In this context, we will use the term *monitor* as a synonym of *deterministic symbolic automaton* (Definition 40).

Intuition In order to accomplish the task, for each α_i (resp. for each β_i), we define a monitor M_{α_i} (resp. M_{β_i}) that is set to *true* when α_i (resp. β_i) has been read and is reset to *false* when all the α_i (resp. β_i) have been read. For this last condition, we define the monitors M_{α}^{tot} and M_{β}^{tot}.

Monitors for the degeneralization Let M_{α_i} and M_{α}^{tot} be the symbolic safety automata such that their input alphabet is 2^{Σ} (where Σ is the alphabet of the starting GR-EBR formula), their set of *state variables* are $\{m_{\alpha_i}\}$ and $\{m_{\alpha}^{tot}\}$, respectively, all their reachable states are safe states, and their transition relations are the following:

```
init(m_{α_i}) := 0
next(m_{α_i}) := case
  α_i         :    1
  m_α^tot     :    0
  default     :    m_{α_i}
esac
```

```
init(m_α^tot) := 0
next(m_α^tot) := case
  m_{α_1} ∧ ··· ∧ m_{α_m} : 1
  default                 : 0
esac
```

We define M_{β_i} and M_β^{tot} as M_{α_i} and M_α^{tot}, respectively, but with α_i substituted with β_i and α substituted with β. Let $\mathcal{A}_{degen}$ be the product between all the M_{α_i}, M_{β_i}, M_α^{tot} and M_β^{tot}. Let $\mathcal{A}_{\text{degen}}^{\mathsf{GF}\to\mathsf{GF}}$ be the automaton obtained from $\mathcal{A}_{degen}$ by replacing its accepting condition with the Reactivity(1) condition $\mathsf{GF}m_\alpha^{tot} \to \mathsf{GF}m_\beta^{tot}$. We can prove the following lemma, which states that this step of the algorithm maintains the equivalence.

Lemma 22. $\mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\to\wedge}) = \mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{\text{degen}}^{\mathsf{GF}\to\mathsf{GF}})$.

Proof. We prove separately the two directions. Consider first the right-to-left direction. Let σ be an infinite word of $\mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{\text{degen}}^{\mathsf{GF}\to\mathsf{GF}})$. Then σ is a word in $\mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}})$. Moreover, σ is a word in $\mathcal{L}(\mathcal{A}_{\text{degen}}^{\mathsf{GF}\to\mathsf{GF}})$ and thus there exists a run τ induced by σ such that $\tau \models \mathsf{GF}m_\alpha^{tot} \to \mathsf{GF}m_\beta^{tot}$, that is, $\tau \models \mathsf{FG}\neg m_\alpha^{tot} \vee \mathsf{GF}m_\beta^{tot}$. We divide in cases:

- if $\tau \models \mathsf{FG}\neg m_\alpha^{tot}$, then by the semantics of the temporal operators F and G, there exists an $i \geq 0$ such that for all $j \geq i$, $\tau_j \models \neg m_\alpha^{tot}$. By construction of the monitors m_α^{tot}, this means that there exists an $i \geq 0$ such that for all $j \geq i$, $\tau_j \models \bigvee_{k=1}^m \neg m_{\alpha_k}$. This implies that, there exists a $k \in [1, m]$ and an $i \geq 0$ such that for all $j \geq i$, such that $\tau_j \models \neg m_{\alpha_k}$. Indeed, suppose by contradiction that it is not so: then for all $k \in [1, m]$, there exists infinitely many positions $i \geq 0$ such that $\tau_i \models m_{\alpha_k}$. This would mean that the monitor M_α^{tot} is set to *true* infinitely many times, that is $\mathsf{GF}m_\alpha^{tot}$, but this is a contradiction with our hypothesis. Therefore, it holds that $\tau \models \bigvee_{k=1}^m \mathsf{FG}\neg m_{\alpha_k}$, and thus also that $\tau \models \bigwedge_{i=1}^m \mathsf{GF}\alpha_i \to \bigwedge_{j=1}^n \mathsf{GF}\beta_j$. Overall, since τ is induced by σ, we have that σ is a word of $\mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}})$ that induces a run τ such that $\tau \models \bigwedge_{i=1}^m \mathsf{GF}\alpha_i \to \bigwedge_{j=1}^n \mathsf{GF}\beta_j$, that is $\sigma \in \mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\to\wedge})$.

- If otherwise $\tau \models \mathsf{GF}m_\beta^{tot}$, then there exists infinitely many positions $i \geq 0$ such that $\tau_i \models m_\beta^{tot}$. Moreover, it holds that for all $i_1 \geq 0$ and for all $i_2 \geq i_1$, if $\tau_{i_1} \models m_\beta^{tot}$ and $\tau_{i_2} \models m_\beta^{tot}$,

then, for all $1 \leq k \leq n$, there exists a $i_1 \leq j \leq i_2$ such that $\tau_j \models m_{b_k}$. Putting together these two points, we have that for all $1 \leq k \leq n$, there exists infinitely many $i \geq 0$ such that $\tau_i \models m_{b_k}$. That is, $\tau \models \bigwedge_{k=1}^{n} \mathsf{GF} m_{b_k}$. By definition of the monitors M_{β_i} and since τ is induced by σ, we have that σ is a word in $\mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}})$ that induces a run τ such that $\tau \models \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$. That is, $\sigma \in \mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge \rightarrow \wedge})$.

The proof the left-to-right direction is specular, and therefore is omitted from the presentation. □

Related work Our degeneralization step is similar to the one proposed in [22] for transforming a $\mathsf{GR}(1)$ condition to a *one-pair Streett condition*, in the context of parity game solving. The main difference is that, in this paper, we do *not* fix any order on the visits to the α_i (resp. β_i). For example, M_{α}^{tot} is set to *true* whenever all the α_i have been read, no matter the order. As noted in [83], this has the potential to be more effective than imposing an order to the visits (like in [22]), for example in the case where the order is $\langle \beta_1, \beta_2, \ldots, \beta_n \rangle$ but Controller can never satisfy fairness β_i after having satisfied first the fairness β_{i+1}.

8.3.3 Reduction to Safety for $\mathsf{R}(1)$ objectives

In this part, we describe a *complete* safety reduction (see Definition 54) tailored for Reactivity(1) objectives. We will apply this reduction on the automaton $\mathcal{A}_{\text{degen}}^{\mathsf{GF} \rightarrow \mathsf{GF}}$ obtained from the previous step. The intuition is to use a *counter* to count and limit the number of positions, after a position in which m_{β}^{tot} holds, in which $m_{\alpha}^{tot} \wedge \neg m_{\beta}^{tot}$ holds. We define the counter as follows.

Definition 55 (Counter for the Reactivity(1) objective)**.** *Let $\mathcal{A}_{\#_{\alpha,\beta}^{\rightarrow}}^{k}$ be the symbolic and deterministic safety automaton whose set of safe states is represented by the formula $\mathsf{G}(\#_{\alpha,\beta}^{\rightarrow} < k)$ and whose transition relation is the following:*

$$
\begin{array}{lll}
init(\#_{\alpha,\beta}^{\rightarrow}) & := & 0 \\
next(\#_{\alpha,\beta}^{\rightarrow}) & := & case \\
\quad m_{\beta}^{tot} & : & 0 \\
\quad m_{\alpha}^{tot} & : & \#_{\alpha,\beta}^{\rightarrow} + 1 \\
\quad default & : & \#_{\alpha,\beta}^{\rightarrow} \\
esac &&
\end{array}
$$

We define $\mathcal{A}^{\mathrm{k}}_{safe} \coloneqq \mathcal{A}_{\phi_{\mathrm{ebr}}} \times \mathcal{A}_{degen} \times \mathcal{A}^{k}_{\#^{\rightarrow}_{\alpha,\beta}}$, and we set the accepting condition of $\mathcal{A}^{\mathrm{k}}_{safe}$ to be the one of $\mathcal{A}^{k}_{\#^{\rightarrow}_{\alpha,\beta}}$, *i.e.*, $\mathsf{G}(\#^{\rightarrow}_{\alpha,\beta} \leq \#^{\rightarrow}_{\alpha,\beta} < k)$. The automaton $\mathcal{A}^{\mathrm{k}}_{safe}$ is a symbolic and deterministic safety automaton, and therefore it can be used as an arena for a safety game. In practice, we check the realizability of $\mathcal{A}^{\mathrm{k}}_{safe}$ by means of a tool for safety synthesis. We start with $k = 0$, and we check the realizabilty of $\mathcal{A}^{\mathrm{k}}_{safe}$: if Controller has a strategy, than we stop, otherwise we increment k and we repeat the cycle.

In order to prove that this step is sound and complete, we use the framework described in Section 8.2. We call $[\![\cdot]\!]_{ebr}$ the safety reduction described in this part. Since the framework works with formulas rather than with automata, for all $\phi \in \mathsf{GR\text{-}EBR}$, we define $[\![\phi]\!]^{k}_{ebr}$ to be any *safety formula* such that $\mathcal{L}([\![\phi]\!]^{k}_{ebr}) = \mathcal{L}(\mathcal{A}^{\mathrm{k}}_{safe})$. From now, with $\mathsf{id} : \mathbb{N} \to \mathbb{N}$ we denote the *identity* function.

Theorem 42. $[\![\cdot]\!]_{ebr}$ *is a* id*-complete safety reduction for* $\mathsf{GR\text{-}EBR}$.

Proof. We have to prove that, for all $\phi \in \mathsf{GR\text{-}EBR}$, for all Kripke structures M and for all $k \in \mathbb{N}$, it holds that:

$$M \models \mathsf{A}\,\phi \qquad \Leftrightarrow \qquad \exists k \leq \mathsf{id}(|M|)\,.\, M \models \mathsf{A}[\![\phi]\!]^{k}_{ebr}$$

We prove separately the two directions. Consider first the *soundness* which corresponds to the right-to-left direction. Suppose that $M \models \mathsf{A}[\![\phi]\!]^{k}_{ebr}$. It holds that, for each initialized trace π of M, $L(\pi) \models [\![\phi]\!]^{k}_{ebr}$, where $L(\cdot)$ is the labeling function of M. Let π be an initialized trace of M. By definition of $[\![\cdot]\!]_{ebr}$, it holds that, there exists a run τ induced by $L(\pi)$ such that: (i) τ is accepting in $\mathcal{A}_{\phi_{\mathrm{ebr}}} \times \mathcal{A}_{degen}$, and (ii) τ is accepting in $\mathcal{A}^{k}_{\#^{\rightarrow}_{\alpha,\beta}}$. From the second point, we have that:

- either, $\#^{\rightarrow}_{\alpha,\beta}$ make infinitely many *resets*. This means that there exists infinitely many positions in τ in which m^{tot}_{α} holds and, after at most k occurrences of m^{tot}_{α}, there is a m^{tot}_{β}. Therefore, in particular, there exists infinitely many positions in which m^{tot}_{β} holds, that is $\tau \models \mathsf{GF}m^{tot}_{\beta}$.
- or the counter $\#^{\rightarrow}_{\alpha,\beta}$ stops to increment because, because it does not read any m^{tot}_{α}. This means that there exists *finitely* many positions in which m^{tot}_{α} holds, that is $\tau \models \mathsf{FG}\neg m^{tot}_{\alpha}$.

Therefore, it holds that $\tau \models \mathsf{FG}\neg m_\alpha^{tot} \vee \mathsf{GF} m_\beta^{tot}$, that is $\tau \models \mathsf{GF} m_\alpha^{tot} \rightarrow \mathsf{GF} m_\beta^{tot}$. Finally, we have that τ is an accepting run of $\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{degen}$ such that $\tau \models \mathsf{GF} m_\alpha^{tot} \rightarrow \mathsf{GF} m_\beta^{tot}$. Since by hypothesis $L(\pi)$ is induced by τ, by definition of $\mathcal{A}_{\text{degen}}^{\mathsf{GF}\rightarrow\mathsf{GF}}$, we have that $L(\pi) \in \mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{\text{degen}}^{\mathsf{GF}\rightarrow\mathsf{GF}})$. By concatenating Lemma 21 and Lemma 22, we have that $L(\pi) \in \mathcal{L}(\phi)$, and therefore $\pi \models \phi$. It follows that $M \models \mathsf{A}\,\phi$.

We now prove *completeness*, which corresponds to the left-to-right direction. Suppose that $M \models \mathsf{A}\,\phi$, where $\phi \in \mathsf{GR\text{-}EBR}$. We prove this case by contradiction. Suppose therefore that for all $k \leq \mathsf{id}(|M|)$, $M \not\models \mathsf{A}[\![\phi]\!]_{ebr}^k$. This means that there exists an initialized trace π in M such that $L(\pi) \notin \mathcal{L}([\![\phi]\!]_{ebr}^k)$, for all $k \leq \mathsf{id}(|M|)$. By definition of $[\![\cdot]\!]_{ebr}$, for $k = \mathsf{id}(|M|)$, we have that *for all* runs τ induced by $L(\pi)$ in $\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{degen} \times \mathcal{A}^k_{\#^{\rightarrow}_{\alpha,\beta}}$, it holds that $\tau \not\models \mathsf{G}(\#^{\rightarrow}_{\alpha,\beta} \leq k)$. Let τ be one of these runs. There exists a position i in τ such that $\tau_i \models (\#^{\rightarrow}_{\alpha,\beta} = v)$, for some $v > k$. By definition of the counter $\#^{\rightarrow}_{\alpha,\beta}$, the run τ is such that:

$$\begin{aligned}
&\exists 0 < h_1 < h_2 < \cdots < h_v \,.\, (\quad \tau_{h_1} \models m_\alpha^{tot} \wedge \\
&\quad \tau_{h_2} \models m_\alpha^{tot} \wedge \ldots \tau_{h_v} \models m_\alpha^{tot} \wedge \\
&\quad \forall h_1 \leq h \leq h_v \,.\, (\tau_j \models \neg m_\beta^{tot}))
\end{aligned}$$

Recall that τ is a run induced by $L(\pi)$. Since $v > k$, $k = \mathsf{id}(|M|)$ and M is a *finite-state* Kripke structure, the positions $h_1 \ldots h_v$ in π (attention: *not* in τ) cannot be all different. That is, there exists at least two indexes $s, e \in \mathbb{N}$ such that: (i) $1 \leq s < e \leq v$, (ii) $\pi_{h_s} = \pi_{h_e}$, and (iii) $\pi_{h_s} \models m_\alpha^{tot}$. Starting from π, we can build a *looping trace* π' that agrees with π in the prefix $\pi_{[0,h_e]}$ and then loops on the interval $\pi_{[h_s,h_e]}$. It holds that π' is an initialized trace of M and it induces a run τ' such that $\tau' \models \mathsf{GF} m_\alpha^{tot} \wedge \mathsf{FG}\neg m_\beta^{tot}$, that is $\tau' \not\models \mathsf{GF} m_\alpha^{tot} \rightarrow \mathsf{GF} m_\beta^{tot}$. Nevertheless, since $M \models \mathsf{A}\,\phi$, by Lemma 21 and Lemma 22, we have that $L(\pi') \in \mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{\text{degen}}^{\mathsf{GF}\rightarrow\mathsf{GF}})$, and therefore this is a contradiction. This means that it has to hold that $L(\pi) \in \mathcal{L}([\![\phi]\!]_{ebr}^k)$, that is $\pi \models [\![\phi]\!]_{ebr}^k$ for all the initialized traces π of M, and thus there exists a $k \leq \mathsf{id}(|M|)$ such that $M \models \mathsf{A}[\![\phi]\!]_{ebr}^k$. □

With Theorem 40, we derive the following corollary that proves the completeness of our procedure.

Corollary 11. *For any formula $\phi \in$ GR-EBR, it holds that: ϕ is realizable iff $\exists k \leq \mathsf{id}(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}})$ such that $[\![\phi]\!]^k_{ebr}$ is realizable.*

8.4 Experimental Evaluation

We implemented the algorithm described in Section 8.3 and summarized in Fig. 8.2 in a prototype tool called GRACE (which stands for *GR-ebr reAlizability ChEcker*) [1] . We chose SAFETYSYNTH [116] as a BDD-based backend for solving each safety game. SAFETYSYNTH implements the classical BDD-based backward fixpoint for finding a strategy for Controller in a safety game represented in AIGER format [19]. We set a timeout of 180 seconds. The experiments have been run on a 16-cores machine with a 2696.6 MHz AMD core with 62 GB of RAM.

8.4.1 Description of the competitor tools

As competitor tools, we chose BoSy [91, 89, 88] and STRIX [137, 150].

BoSy implements the Bounded Synthesis approach (Section 5.3.6), while STRIX is based on parity games and is the winner of SYNTCOMP 2018, 2019 and 2020. The main algorithm of BoSy takes a temporal formula ϕ in input and consists of the following steps:

1. it builds the Universal co-Büchi automaton (UCA) for ϕ; this automaton is built by executing one of the two tools LTL3BA [10] and SPOT [78], and it is *explicitly* represented;

2. the automaton is optimized, *e.g.*, by analyzing its strongly connected components [92];

3. the optimized automaton, along with the bound k on the visits to its rejecting states, is encoded into a constraint system (*e.g.*, SAT, QBF, SMT) and solved by a corresponding backend.

Among the different encodings, the one based on *Quantified Boolean Formulas* (QBF) appears to be the most efficient one in practice [88], and thus it is the default one and the one with which we compare our tool GRACE. Finally, BoSy starts two threads, one checking the realizability of the formula and the other checking the unrealizability. Since we will evaluate GRACE and BoSy only on realizable formulas

[1] `https://es-static.fbk.eu/tools/grace/`

(the only ones of interest in our context), in order to make fair the comparison with BoSy, we commented the part of the source code of BoSy that starts the thread for the unrealizability check.

strix is based on the classical approach with *parity games* (Section 5.3.7) and in addition it implements several optimizations like specification splitting, that enables to split the initial formula in safety, co-safety, Büchi, and co-Büchi subformulas and speeds up the process of solving of the game.

We remark that a comparison with $\mathsf{GR}(1)$ synthesis tools is non-trivial. The majority of the tools for $\mathsf{GR}(1)$ only support the realizability of the *strict* implication (see for example [84]), not the standard one (which is our case). Therefore, although the latter can be reduced to the former [25], a non-trivial practical effort is required to write an algorithm for this translation.

8.4.2 Description of the benchmarks set

We considered benchmarks of two types: (i) artificial, and (ii) derived from the SYNTCOMP [116] benchmarks' set. Regarding the artifical benchmarks, we partitioned them in four categories, each containing 30 benchmarks scalable in their dimension N, for a total of 120 formulas. The categories are the following ones:

1. $\mathsf{G}(u_0 \rightarrow \mathsf{X}(u_1 \rightarrow \mathsf{X}(u_2 \rightarrow \cdots \rightarrow \mathsf{X}(u_N)\dots))) \rightarrow \mathsf{G}(\bigwedge_{i=1}^{N}(u_i \leftrightarrow \mathsf{X}c_i))$
2. $(\mathsf{G}(u_0 \rightarrow \mathsf{X}(u_1 \rightarrow \mathsf{X}(u_2 \rightarrow \cdots \rightarrow \mathsf{X}(u_N)\dots))) \wedge \mathsf{X}^N\mathsf{G}u_N \wedge \mathsf{GF}u_N) \rightarrow (\bigwedge_{i=1}^{N}(u_i \leftrightarrow \mathsf{X}^N c_i) \wedge \mathsf{GF}c_N)$
3. $(\mathsf{G}(u_0) \wedge \mathsf{XG}(u_1) \wedge \cdots \wedge \mathsf{X}^N\mathsf{G}(u_N) \wedge \bigwedge_{i=1}^{N}\mathsf{GF}u_i) \rightarrow (\bigwedge_{i=1}^{N}\mathsf{G}(u_i \leftrightarrow c_i) \wedge \bigwedge_{i=1}^{N}\mathsf{GF}c_i)$
4. $(\neg u_0 \wedge \mathsf{G}^{[0,N]}\neg u_0 \wedge \mathsf{X}^{N+1}\mathsf{G}u_0) \rightarrow (\bigwedge_{i=1}^{N}\mathsf{G}(u_0 \leftrightarrow \mathsf{X}c_i) \wedge \bigwedge_{i=1}^{N}\mathsf{GF}(c_i \wedge u_0))$

The variables starting with u are *u*ncontrollable, while those starting with c are *c*ontrollables. All the benchmarks are realizable, and were specifically crafted to elicit potential criticalities of grace. The formulas in the first category consist of an implication between two $\mathsf{LTL}_{\mathsf{EBR}}$+P formulas. The second category extends the first one by adding to its assumptions (resp. the guarantees) the stabilization constraint $\mathsf{X}^N\mathsf{G}u_N$ (resp. $\mathsf{X}^N\mathsf{G}c_N$) and a *single* fairness $\mathsf{GF}u_N$ (resp.

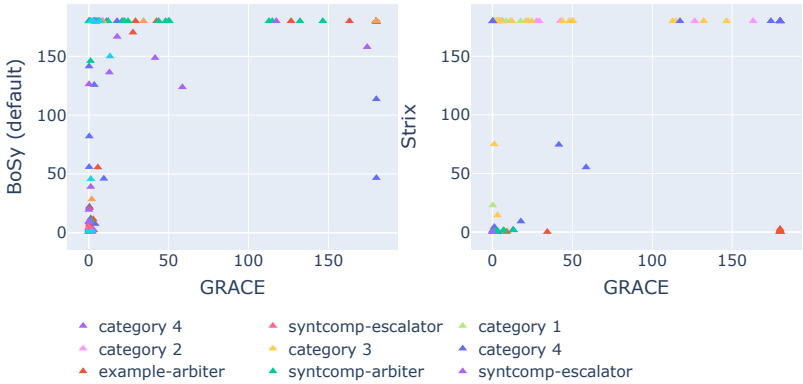


Figure 8.3: GRACE compared to BoSy (on the left) and to STRIX (on the right)

GFc_N). In the third category, the formula corresponding to both the assumptions and the guarantees is such that half of it is safety and the other half is fairness. In particular, there are *multiple* fairness, as many as the dimension N. The benchmarks in the fourth category have been specifically designed in order to force the minimum k of the termination of GRACE to increase with their dimension.

Regarding the benchmarks derived from the SYNTCOMP benchmarks' set, we included (i) *simple_arbiter_N* (for each $N \in \{2, 4, 6, 8, 10, 12\}$), *escalator_ bidirectional*, which belong to the SYNTCOMP benchmarks' set, and (ii) our example for an arbiter (Section 3.3.2), with $N \in \{1, \ldots, 15\}$.

8.4.3 Discussion of the results

Fig. 8.3 show the comparison between the tools on all the benchmarks of both types. All times are in seconds. From Fig. 8.3 (left), it is clear the exponential blowup in solving time in which BoSy occurs. The blowup involves formulas of both types, and of all four categories (of artificial benchmarks). For example, (i) on category 1, the blowup is immediate starting from $N = 4$ (on which BoSy takes 0.463 sec.) and $N = 5$ (on which BoSy reaches the time-

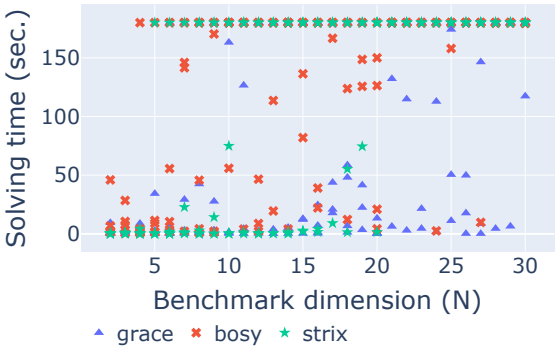


Figure 8.4: The size of the benchmarks compared to solving time.

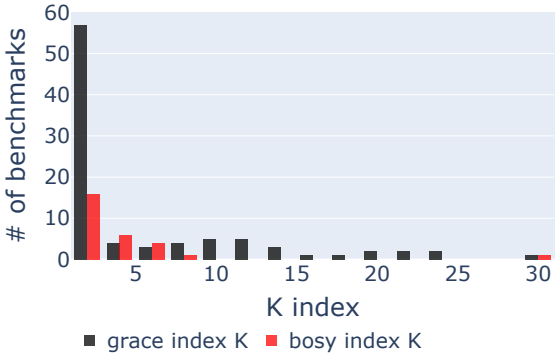


Figure 8.5: On the right, GRACE vs BoSy on number of safety sub-problems.

out); for $N = 4$, the corresponding automaton (after optimization) has 40 states. (ii) on category 2 (and similarly for category 3), the blowup starts with $N = 5$; the solving time of BoSy is of 49.272 sec., and the corresponding automaton (after optimization) has 48 states; with greater values of N, BoSy reaches the timeout; (iii) on category 4, the solving times of BoSy on $N = 13, 14$ are 19.4 and 136.3 sec., respectively, and the corresponding automata have 27 and 31 states, respectively. (iv) on `simple_arbiter_N`, BoSy takes 45.714 sec. for $N = 8$, and reaches the timeout for $N = 10$. Fig. 8.4, which compares the dimension of the benchmarks (X axis) with the solving time of GRACE, BoSy and STRIX (Y axis), clearly shows this trend. A more precise study of the complexity of BoSy shows that the majority of the time spent by it is due to the construction of the UCA corresponding to the input formula, which is the task of the tools LTL3BA and SPOT. On the contrary, it is clear from Fig. 8.3 (left) that GRACE avoids this bad behavior, most likely due to the fact that the explicit state UCA is never built. Similar considerations can be made for the tool STRIX (see Fig. 8.3, right), except for the category `example-arbiter`, in which the solving times of STRIX are consistently better than the ones of GRACE. A careful study revealed that all these benchmarks are transformed to the equi-realizable formula `true` by the preprocessor of OWL [126] (a tool for ω-automata manipulation), which STRIX is based on.

The plot in Fig. 8.5 shows, for each index k ranging from 1 to 31 (these correspond to the columns), on how many benchmarks (of both types) GRACE or BoSy terminate with index k (this corresponds to the height of a column). The benchmarks in category 4 and the ones of `simple_arbiter_N` force GRACE to terminate with increasing values of k. The plot in Fig. 8.5 points out that BoSy does not incur in this growth, except for one benchmark. Nevertheless, the solving times of GRACE are still better than the ones of BoSy. This witnesses the fact that each safety sub-problem generated by GRACE is very simple to solve.

8.5 Conclusions

In this paper, we focused on the realizability problem of GR-EBR, an extension of $\mathsf{LTL}_{\mathsf{EBR}}$+P adding fairness conditions and assumes-guarantees formulas, thus able to express properties beyond the

safety fragment.

We proposed a general framework to derive complete safety reductions in the context of realizability of (fragments of) LTL+P, and then we used it as the core of an algorithm for the realizability of GR-EBR formulas. The algorithm reduces the original problem to a sequence of safety realizability problems. The properties of the framework ensure the completeness of the algorithm. We implemented our approach in a prototype tool and showed the achieved improvement in running time with respect to a tool for full LTL+P realizability based on Bounded Synthesis (Section 5.3.6) and one based on parity games. The experimental evaluations showed good performances against tools for bounded synthesis.

We aim at extending the work done in three directions: (i) as far as we know, there are no safety synthesizer (like SAFETYSYNTH [116]) that are able to exploit *incrementality*; since in our context, the only part of the automaton that changes between one iteration and the next one is the counter, some work may be saved; (ii) *stabilizing constraints* are successfully used in model checking, in particular by the K-Liveness algorithm (Section 5.2.6, [57]), in order to shrink the search of the search space; we expect that realizability may also benefit from them; (iii) since GR(1) is a very efficient fragment, it is important to investigate whether there is a compilation from GR-EBR to GR(1); (iv) last but not least, we aim at exploiting the proposed framework for more expressive logics, such as full LTL+P.

CHAPTER

9

COMPATIBILITY CHECK OF TIMING REQUIREMENTS

The initialization of complex cyber-physical systems often requires the interaction of various components that must start up with strict timing requirements on the provision of signals (power, refrigeration, light, etc.). In order to safely allow an independent development of components, it is necessary to ensure that the specification of local timing requirements prevents later integration errors due to the dependencies (*safe decomposition*).

We propose a high-level formalism to model local timing requirements and dependencies. We consider the problem of checking the consistency (existence of an execution satisfying the requirements) and compatibility (absence of an execution that reaches an integration error) of the local requirements, and the problem of synthesizing a region of timing constraints that represents all possible correct re-

finements of the original specification. We show how the problems can be naturally translated into a model checking and parameter synthesis problem (Section 5.3.9) for timed automata with shared variables. We propose an encoding of the problem into SMT (Section 5.2.4) by exploiting the linear structure of the requirements. We evaluate the SMT-based approach using MathSAT and show how it scales better than the automata-based approach using Uppaal and nuXmv.

9.1 Introduction

Complex industrial cyber-physical systems often have an initialization procedure that requires to reach a startup mode within a specified design target time interval. In order for the system as a whole to complete the startup within the required interval, each subcomponent of the system may have to go through a number of intermediate phases, within their own target intervals, each of which may itself be dependent upon other subcomponents reaching startup or intermediate phases. E.g. for a power generation system to startup at full power, it may need to transition first through a low power output phase and a number of subsidiary systems (perhaps cooling or fuel supply) may first have to undergo their own phase transitions. In turn, these subsidiary systems may require transitions to occur in systems subsidiary to them and so on.

Traditionally, the integration of these distributed transition targets are validated via simulation and testing, which while sufficient to reach a desired design performance are labor and time intensive. Having a more efficient process for arriving at and validating a set of design targets that satisfy the overall system requirements is clearly beneficial in these contexts. Firstly, we would like to verify that these requirements prevent failed transitions in which the time performance of the subsidiary systems lead to outcomes where our main system (e.g., the power generation system) cannot perform a transition within its time window. For example, suppose the power system has a time window within which it must transition from low-power mode to high-power mode; in order for it to achieve this transition, however, it requires that two subsidiary systems, a cooling system and a fuel supply system, must themselves transition from a low-output mode to a high-output mode, each within their

own target transition time windows. If these time windows are not compatible, the power generator may fail to provide the high power in time. Secondly, if our starting set of requirements is inadequate to provide this guarantee, we would like to be able to synthesize a set of requirements that is adequate to this task.

In this chapter, we formalize the problem starting from a simple industrially relevant setting, where the components have a linear sequence of phases, must progress to the next phase within a certain interval of time, and must respect some dependencies upon the phases of other components. Dependencies are expressed as Boolean combinations of variables representing the component phases and are divided into two types: (i) *signal dependencies*, where the entering of a component into a phase is conditioned by the presence of other components in some specific phases; (ii) *state dependencies*, where a component can stay in a phase only if, during all its stay, other components are in some specific phases. We are interested in the following problems:

1. checking if the requirements are compatible, *i.e.*, if all reachable states can be extended with an execution satisfying the requirements; thus, if the components satisfy the local requirements, they cannot lead the system to an illegal state (where a component does not receive the input in time);

2. checking if the requirements are consistent, *i.e.*, there exists an execution of the components satisfying all requirements (inconsistency is actually a pathological case of incompatibility);

3. synthesizing the set of refinements (same requirements with stricter intervals) that are consistent and compatible.

We show how the first two verification problems can be naturally translated into a model checking problem for timed automata (TA) with shared variables. Exploiting the linear structure of the requirements, we propose an encoding of the problem into SMT (Section 5.2.4). If all intervals are bounded, the encoding is quantifier-free. Finally, both approaches have been extended to solve also the parameter synthesis problem (Section 5.3.9), using synthesis for parametrized model checking of TAs and quantifier elimination in SMT, respectively.

We implemented the SMT-based approach in a tool called TRICKER and carried out experimental evaluation, comparing it with

other tools for the verification of timed automata. We used UPPAAL [11] and nuXMV [49] to model check timed automata and MATHSAT [55] to solve the SMT problems. We performed an experimental evaluation based on a test-set of randomly generated local requirements. When comparing the SMT-based approach with the automata-based one, the results highlight a better performance of the former technique on all three problems.

9.1.1 Related Work

The problem of the integration and compatibility of input/output timed automata has been extensively studied in the literature. Typically, works in the literature focus on deadlock checking (see, e.g., [8, 9]). The work of [6] also addresses the parameter synthesis to avoid deadlocks in timed automata. In order to check for livelocks, liveness properties can be addressed with approaches proposed in [52, 49]. A general definition of illegal states for timed interface automata is given in [69]. As shown in Section A.2, the compatibility problem addressed in this chapter can be seen as a subcase of the homonym problem for input/output timed interface automata. As we are considering a closed system, the problem reduces to the existence of a deadlock or livelock in a phase of some component (depending if the related time interval is bounded or not). Moreover, compared to the above model checking approaches we are considering a specific fragment of timed automata with a linear structure that can be exploited for specialized solutions.

Related problems have been addressed in the context of task scheduling. In the formalism introduced in [183, 184], called DRT (short for *digraph real-time task* model), in which tasks and deadlines are expressed as directed graphs, the problem of determining whether a schedule exists (*feasibility problem*) bears some similarities with the consistency checking problem we study here. The DRT model allows the use of very general graph topologies, with multiple outgoing branches and loop-backs, but it does not consider dependencies across different tasks. The main difference with our work is that the problem is addressed from a *global* point of view (*i.e.*, the existence of a global scheduler that can coordinate the execution of the tasks), whereas we are interested in local solutions, in which each requirement can be considered in isolation. Another difference is the approach used to tackle the problem: while in [183]

dynamic programming is used to deal with the possible explosion of the search space, we use SMT [75] as the main framework for all the three above-mentioned problems.

9.1.2 Outline

In Section 9.2, we introduce a suitable formalism to model local requirements and we formalize the three problems. In Section 9.3, we propose the reductions of *compatibility checking* and *consistency checking* into timed automata and SMT. The corresponding solutions for the *parameter synthesis* problem are then described in Section 9.4. The experimental results are described in Section 9.5. In Section 9.6, we draw some conclusions and highlight possible future directions of this work.

9.2 Problem Statement

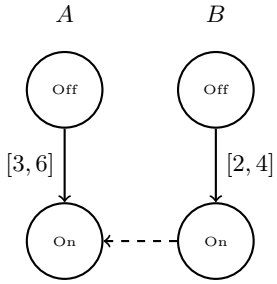
9.2.1 Domain formalization

We propose a high level formalism to model the local requirements.

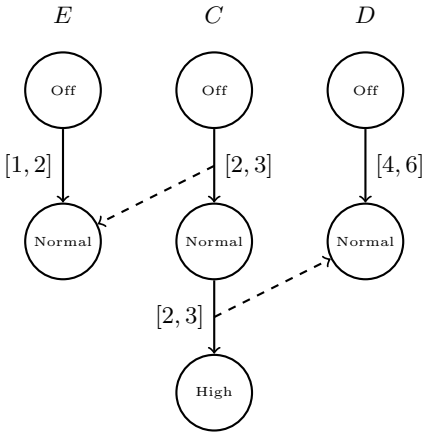
Definition 56 (Local Requirements)**.** *A specification S is given by a set of local (or component) requirements, where each local requirement $C \in S$ is given by an (ordered) sequence $\langle P_1^C, \ldots, P_n^C \rangle$ of phases. In turn, when $i > 1$ each phase P_i of C is associated with a closed real interval β_{P_i} with non-negative lower limit l_{P_i} and (finite or infinite) upper limit u_{P_i}, with a formula ϕ_{P_i} (called* signal dependency*) and, when $i > 0$ with a formula ψ_{P_i} (called* state dependency*). Both ϕ_{P_i} and ψ_{P_i} are Boolean formulae over the atoms in $\{\langle D, Q \rangle\}_{D \in S \setminus \{C\}, Q \in D}$ (i.e., the phases of other components).*

If a dependency ψ_P is just a conjunction of atoms, then we say that ψ_P is *convex*. With the notation $|C|$, we will refer to the number of phases of C.

Figs. 9.1a and 9.1b show two examples of sets of local requirements. In Fig. 9.1a, we have two local requirements A and B (i.e., $S = \{A, B\}$); each local requirement has two phases *Off* and *On* (i.e., $P_1^A = \mathit{Off}$ and $P_2^A = \mathit{On}$ and similarly for B); the bounds are depicted in square brackets (thus, for example $\beta_{\mathit{On}}^A = [3, 6]$); all dependencies are trivially true apart from the state dependency $\psi_{\mathit{On}}^B = \langle A, \mathit{On} \rangle$ of the local requirement B, which is plotted as an



(a) Example of system with two local requirements and one state dependency.



(b) Example of system with two local requirements and two signal dependencies.

arrow from the phase *On* of B to phase *On* of A. In Fig. 9.1b, we have another example with three components and some signal dependencies; for example, signal dependency $\phi^C_{Normal} = \langle E, Normal \rangle$ is plotted as an arrow from the *transition* to phase *Normal* of C to phase *Normal* of E.

Definition 57 (Stronger local requirements)**.** *We say that a local requirement* $C' = \langle P_1^{C'}, \ldots, P_n^{C'} \rangle$ *is* stronger *than* $C = \langle P_1^C, \ldots, P_n^C \rangle$ *(written* $C' \preceq C$*), iff phase* $P_i^{C'}$ *is identical to* P_i^C *except that* $l_{P_i^C} \leq l_{P_i^{C'}}$ *and* $u_{P_i^{C'}} \leq u_{P_i^C}$*, for all* $1 \leq i \leq n$*. Given two specifications* $S = \{C_1, \ldots, C_n\}$ *and* $S' = \{C'_1, \ldots, C'_n\}$*, we say that* S' *is stronger than* S *(written* $S' \preceq S$*) iff for all* i*,* $1 \leq i \leq n$*,* $|C_i| = |C'_i|$ *and* $C'_i \preceq C_i$*.*

In defining the semantics of a composition of local requirements $C_1 \ldots C_n$, every local requirement C_i is associated with a local clock, which is reset each time it enters a new phase. Given a local requirements specification $\{C_1, \ldots, C_n\}$, we define its semantics formally by defining the predicate $Reach((C_1, j_1, t_1), \ldots, (C_n, j_n, t_n))$, which is true iff the phases $P_{j_1}^{C_1} \ldots P_{j_n}^{C_n}$ are reachable at local times $t_1 \ldots t_n$.

Definition 58 (Reachability for local requirements)**.** *Given the specification* $\{C_1 \ldots C_n\}$ *and the time points* $t_1 \in \mathbb{R} \ldots t_n \in \mathbb{R}$*, we in-*

ductively define the predicate $Reach((C_1, j_1, t_1), \ldots, (C_n, j_n, t_n))$ *as follows:*

- (base case) $Reach((C_1, 1, 0), \ldots, (C_n, 1, 0))$ *holds and for all* $i \in \{1 \ldots n\}$ *it holds that (state dependencies):* $((C_1, 1), \ldots, (C_n, 1)) \models \psi_1^{C_i}$
- (timed transition) *if* $Reach((C_1, j_1, t_1), \ldots, (C_n, j_n, t_n))$ *and there exists a* $\delta \in \mathbb{R}$ *such that* $t_i + \delta \leq u_{j_i+1}^{C_i}$ *for all* $i \in \{1 \ldots n\}$*, then* $Reach((C_1, j_1, t_1 + \delta), \ldots, (C_n, j_n, t_n + \delta))$.
- (discrete transition) *if* $Reach((C_1, j_1, t_1), \ldots, (C_n, j_n, t_n))$ *and there exists a* $\delta \in \mathbb{R}$ *and a* $M \subseteq \{1, \ldots, n\}$ *such that:*
  1. *for all* $i \in \{1 \ldots n\}$ *such that* $j_i < |C_i|$, $t_i + \delta \in [l_{j_i+1}^{C_i}, u_{j_i+1}^{C_i}]$ *if* $i \in M$*, and* $t_i + \delta \leq u_{j_i+1}^{C_i}$ *otherwise;*
  2. *for all* $i \in M$*, it holds that (signal dependencies):* $((C_1, j_1), \ldots, (C_n, j_n)) \models \phi_{j_i+1}^{C_i}$
  3. *for all* $i \in M$*, it holds that (state dependencies - entry):* $((C_1, j_1), \ldots, (C_n, j_n)) \models \psi_{j_i+1}^{C_i}$
  4. *for all* $i \in \{1 \ldots n\}$*, it holds that (state dependencies - invariant):* $((C_1, j_1'), \ldots, (C_n, j_n')) \models \psi_{j_i'}^{C_i}$

  then it holds that $Reach((C_1, j_1', t_1'), \ldots, (C_n, j_n', t_n'))$*, where* $j_i' = j_i + 1$ *and* $t_i' = 0$ *if* $i \in M$ *and* $j_i < |C_i|$*, and* $j_i' = j_i$ *and* $t_i' = t_i + \delta$ *otherwise.*

We define the predicate $Comp_S$ to be true iff there are no reachable states in S such that no component can proceed to its next phase.

Definition 59 (Compatibility for local requirements)**.** *Given the set of local requirements* $S = \{C_1 \ldots C_n\}$*, the predicate* $Comp_S$ *is true iff:*

$$\forall j_1 \in \{1 \ldots |C_1| - 1\} \ldots \forall j_n \in \{1 \ldots |C_n| - 1\}\ \forall t_1 \ldots t_n \in \mathbb{R}\Big($$
$$Reach((C_1, j_1, t_1), \ldots, (C_n, j_n, t_n)) \Rightarrow$$
$$\exists M \subseteq \{1 \ldots n\}\big(M \neq \varnothing \wedge Reach((C_1, j_1', t_1'), \ldots, (C_n, j_n', t_n'))\big)\Big)$$

where $j_i' = j_i + 1$ *and* $t_i' = 0$ *for all* $i \in M$*, or* $j_i' = j_i$ *and* $t_i' = t_i$ *otherwise. If* $Comp_S$ *holds, we say that* $C_1 \dots C_n$ *are compatible, or equivalently that* S *is compatible.*

For example, in Fig. 9.1a, predicate $Reach((A,1,4),(B,1,4))$ holds, but predicate $Reach((A,1,4),(B,2,0))$ does not, because for all $\delta \in \mathbb{R}$ and for all $S \subseteq \{1 \dots n\}$, predicate $Reach((A,1,4),(B,2,0))$ is false.

Strict Semantics The above definition adopts a weakly-monotonic model of time, where discrete transitions are instantaneous and, therefore, the system may be in two different states at the same instant. The definition and the reductions to model checking and SMT can be easily adapted to have a strict semantics.

Verification and Parameter Synthesis Problems The core problem we address is to check if a given specification $S = \{C_1, \dots, C_n\}$ is *compatible*, *i.e.*, if $Comp_S$ holds. The *consistency checking* problem amounts to checking if there *exists* a time point in which the final phase of all the local requirements is reached, that is it amounts to checking if the following formula holds:

$$\exists t_1 \dots \exists t_n \; Reach((C_1, |C_1|, t_1), \dots, (C_n, |C_n|, t_n))$$

If this is the case, then we say that S is *consistent*. Finally, we can formalize the *parameter synthesis problem* (Section 5.3.9) as the problem of computing (a symbolic representation of) the set: $\{S' \mid Comp_{S'} \wedge S' \preceq S\}$

9.2.2 NP-hardness

In this section, we show that the simplest of the problems defined above is already NP-hard. In fact, we show a reduction from SAT to the *consistency checking* problem.

Let $\varphi(\overline{x})$ be a Boolean formula over the variables $\overline{x} = \langle x_1 \dots x_n \rangle$; without loss of generality, we assume $\varphi(\overline{x})$ to be in negated normal form, *i.e.*, with all the negations only in front of literals. For all $1 \leq i \leq n$, we define the local requirement corresponding to variable x_i as $C_i = \langle P_1^i, P_2^i \rangle$, such that $B_{P_2^i} = [0, +\infty)$ and $\phi_{P_1^i} = \psi_{P_1^i} = \phi_{P_2^i} = \psi_{P_2^i} = \top$; the idea is to encode the values $\bot$ and $\top$ of each x_i with the two phases P_1^i and P_2^i, respectively. Moreover, we define

the local requirement G, which will be useful as a gadget for the reduction, as follows: $G = \langle P_1^G, P_2^G \rangle$, where $P_2^G = \langle [0, +\infty), \varphi[x_i \mapsto \langle C_i, P_2^i \rangle, \neg x_i \mapsto \langle C_i, P_1^i \rangle], \top \rangle$. The specification S^φ corresponding to the Boolean formula $\varphi(\overline{x})$ is defined as $S^\varphi = \{G, C_1, \ldots, C_n\}$. It holds that $\varphi(\overline{x})$ is satisfiable if and only if S^φ is consistent. In fact, if S^φ is consistent, then there exists a time point in which the signal dependency of the second phase of G has been satisfied, and thus $\varphi(\overline{x})$ is satisfiable. Viceversa, let's suppose that $\varphi(\overline{x})$ is satisfiable and let M be an arbitrary model of it, expressed as the set of true atoms, in which we also substitute every x_i in it with the pair $\langle C_i, P_2^i \rangle$. Since the local requirements $C_1 \ldots C_n$ have no dependencies and, together with G, have only infinite bounds, there exists a time t such that predicate $\mathit{Reach}((G, P_1^G, t), (C_1, P_{b_1}^G, t_1), \ldots, (C_n, P_{b_n}^n, t_n))$ is true, where for all $1 \leq i \leq n$, $b_i = 2$ and $t_i = 0$ iff $x_i \in M$ and $t_i = t$ otherwise. By definition of *Reach* (see Definition 58), this implies that $\mathit{Reach}((G, P_2^G, t), (C_1, P_2^1, t), \ldots, (C_n, P_2^n, t))$ holds, *i.e.*, S is consistent.

In Section 9.3.2, we will give an encoding of the consistency checking problem based on SMT(DL) (*i.e.*, *Satisfiability Modulo Theory of Difference Logic*). In particular, we will show that the problem can be reduced to the satisfiability of a formula in SMT(DL). Since the latter belongs to NP [154], the consistency checking problem belongs to NP as well, having that *consistency checking* is NP-complete.

9.3 Verification

In this section, we first describe how to reduce both the problem of consistency checking and safe decomposition to model checking of a network of timed automata. We then propose a direct encoding into SMT: as we will see in the next section, if from the theoretical side the latter reduction requires a greater effort to be written down correctly, from the practical side it reveals itself more efficient then the former.

9.3.1 Reduction to Model Checking

In order to formalize the two verification problems into ones of model checking networks of timed automata, we use timed automata with shared variables. To this end, besides the clock constraints $\Xi(C)$, we

define $L = \{l_{\mathcal{A}}, l_{\mathcal{B}}, \dots\}$ as a set of *location variables* (one for each automaton $\mathcal{A}$ in the network), and $\Theta(L)$ as the set of all Boolean combinations of atoms of type $l_{\mathcal{A}} = v_{\mathcal{A}}$, where $\mathcal{A}$ is a timed automata, $l_{\mathcal{A}} \in L$ and $v_{\mathcal{A}}$ is a state of $\mathcal{A}$.

Definition 60 (Timed Automata with Shared Variables)**.** *A* timed automaton with shared variables *(TASV, for short)* $\mathcal{A} = \langle V_{\mathcal{A}}, v^0_{\mathcal{A}}, l_{\mathcal{A}}, C_{\mathcal{A}}, inv^{cl}_{\mathcal{A}}, inv^{loc}_{\mathcal{A}}, T_{\mathcal{A}}\rangle$ *consists of:*

- *a finite set of locations* $V_{\mathcal{A}}$*;*
- *an initial location* $v^0_{\mathcal{A}} \in V_{\mathcal{A}}$*;*
- *a location variable* $l_{\mathcal{A}}$ *with range* $V_{\mathcal{A}}$*;*
- *a finite set of clocks* $C_{\mathcal{A}}$*, where a clock is a real-valued variable;*
- *a* clock invariant $inv^{cl}_{\mathcal{A}} : V_{\mathcal{A}} \to \Xi(C_{\mathcal{A}})$ *for each location;*
- *a* location invariant $inv^{loc}_{\mathcal{A}} : V_{\mathcal{A}} \to \Theta(C_{\mathcal{A}})$ *for each location;*
- *a transition relation* $T_{\mathcal{A}} \subseteq V_{\mathcal{A}} \times 2^{C_{\mathcal{A}}} \times \Xi(C_{\mathcal{A}}) \times \Theta(L) \times V_{\mathcal{A}}$*.*

Given a set of clocks C, we denote with $\nu : C \to \mathbb{R}$ a *clock valuation*, that is a function assigning a rational value to each clock; with $\mathcal{V}_C$, we denote the set of all possible clock valuations over C. For $t \in \mathbb{R}$, $\nu+t$ is the clock valuation which maps every clock $c \in C$ to the value $\nu(c)+t$. For $R \subseteq C$, we define $\nu[R \mapsto 0]$ to be the valuation that maps x to 0 if $x \in R$, and to $\nu(x)$ otherwise. When defining the product of two TASVs, we will deal with tuples $(l_{\mathcal{A}_1}, \dots, l_{\mathcal{A}_n})$ of location variables; in this context, we usually denote with λ any function from the set of n-tuples of location variables to the set $V_{\mathcal{A}_1} \times \cdots \times V_{\mathcal{A}_n}$. Moreover, we write that $\lambda \models \Phi$ (where $\Phi \in \Theta(L)$) iff $\Phi[l_{\mathcal{A}_i} \mapsto v_{\mathcal{A}_i}$, for all $1 \le i \le n]$ is true and $\lambda((\dots, l_{\mathcal{A}_i}, \dots)) = (\dots, v_{\mathcal{A}_i}, \dots)$. We give the semantics of a TASV in terms of traces and we define their product as described below.

Definition 61 (Trace of a TASV)**.** *A* trace τ *of a TASV* $\mathcal{A} = \langle V_{\mathcal{A}}, v^0_{\mathcal{A}}, l_{\mathcal{A}}, C_{\mathcal{A}}, inv^{cl}_{\mathcal{A}}, inv^{loc}_{\mathcal{A}}, T_{\mathcal{A}}\rangle$ *is a (either finite or infinite) sequence of* states *of the form:*

$$\langle v_0, \nu_0, \lambda_0\rangle \xrightarrow{\alpha_1} \langle v_1, \nu_1, \lambda_1\rangle \xrightarrow{\alpha_2} \langle v_2, \nu_2, \lambda_2\rangle \xrightarrow{\alpha_3} \dots$$

such that $v_i \in V_{\mathcal{A}}$*,* $\alpha_i \in \mathbb{R} \cup \{\tau\}$*,* $\nu_i \in \mathcal{V}_{C_{\mathcal{A}}}$ *and* $\lambda_i \in \mathcal{V}_L$ *for all* $i \ge 0$*, and:*

- (initiation) $v_0 = v^0_{\mathcal{A}}$, $\nu_0(x) = 0$ *for all* $x \in C_{\mathcal{A}}$, $\nu_0 \models inv^{cl}_{\mathcal{A}}(v^0_{\mathcal{A}})$, $\lambda_0(l_{\mathcal{A}}) = v_0$ *and* $\lambda_0 \models inv^{loc}_{\mathcal{A}}(v^0_{\mathcal{A}})$;
- (consecution)*: for all* $i \geq 0$
  - *(timed transition) if* $\alpha \in \mathbb{R}$, *then* $v_{i+1} = v_i$ *and* $\nu_{i+1} = \nu_i + \alpha$, $\nu_i + \delta \models inv^{cl}_{\mathcal{A}}(v_i)$, *for all* $0 \leq \delta \leq \alpha$, *and* $\lambda_{i+1}(l_{\mathcal{A}}) = v_i$;
  - *(discrete transition) if* $\alpha = \tau$ *then there is a tuple* $(v_i, R_i, \Xi_i, \Phi_i, v_{i+1}) \in T_{\mathcal{A}}$ *such that:* $\nu_i \models inv^{cl}_{\mathcal{A}}(v_i) \wedge \Xi_i$; $\lambda_i \models \Phi_i$; $\nu_{i+1} = \nu_i[R_i \mapsto 0]$; $\nu_{i+1} \models inv^{cl}_{\mathcal{A}}(v_{i+1})$; $\lambda_{i+1}(l_{\mathcal{A}}) = v_{i+1}$, *and* $\lambda_{i+1} \models inv^{loc}_{\mathcal{A}}(v_{i+1})$.

Definition 62 (Product of TASVs)**.** *Given two TASVs* $\mathcal{A}$ *and* $\mathcal{B}$, *their product is the TASV* $\mathcal{A} \otimes \mathcal{B}$ *defined as follows:*

- $V_{\mathcal{A}\otimes\mathcal{B}} = V_{\mathcal{A}} \times V_{\mathcal{B}}$ *and* $v^0_{\mathcal{A}\otimes\mathcal{B}} = (v^0_{\mathcal{A}}, v^0_{\mathcal{B}})$;
- $l_{\mathcal{A}\otimes\mathcal{B}} = (l_{\mathcal{A}}, l_{\mathcal{B}})$;
- $C_{\mathcal{A}\otimes\mathcal{B}} = C_{\mathcal{A}} \cup C_{\mathcal{B}}$;
- $inv^{cl}_{\mathcal{A}\otimes\mathcal{B}}(v, u) = inv^{cl}_{\mathcal{A}}(v) \wedge inv^{cl}_{\mathcal{B}}(u)$, *for all* $(v, u) \in V_{\mathcal{A}\otimes\mathcal{B}}$;
- $inv^{loc}_{\mathcal{A}\otimes\mathcal{B}}(v, u) = inv^{loc}_{\mathcal{A}}(v) \wedge inv^{loc}_{\mathcal{B}}(u)$, *for all* $(v, u) \in V_{\mathcal{A}\otimes\mathcal{B}}$;
- *the transition relation is defined as follows:*

$$\begin{aligned} T_{\mathcal{A}\otimes\mathcal{B}} = &\{((v, u), R, \Xi, \Phi, (v', u)) \mid (v, R, \Xi, \Phi, v') \in T_{\mathcal{A}}\} \cup \\ &\{((v, u), R, \Xi, \Phi, (v, u')) \mid (u, R, \Xi, \Phi, u') \in T_{\mathcal{B}}\} \end{aligned}$$

It is worth noting that each TASV corresponds to a timed automaton defined in the standard way [5], and viceversa. We define now the TASV corresponding to a local requirement.

Definition 63 (TASV for a Local Requirement)**.** *Let* $C = \langle P^C_1, \ldots, P^C_n \rangle$ *be a local requirement. We define the corresponding TASV* $\mathcal{A} = \{V_{\mathcal{A}}, v^0_{\mathcal{A}}, l_{\mathcal{A}}, C_{\mathcal{A}}, inv^{cl}_{\mathcal{A}}, inv^{loc}_{\mathcal{A}}, T_{\mathcal{A}}\}$ *as follows:*

- *for each phase* P^C_i *of local requirement* C, $v^i_{\mathcal{A}}$ *is the corresponding location in* $V_{\mathcal{A}}$; P^C_0 *corresponds to* $v^0_{\mathcal{A}}$ *and* $C_{\mathcal{A}} = \{c_{\mathcal{A}}\}$;
- *for each phase* P^C_i *(but the last) of* C, $inv^{cl}_{\mathcal{A}}(v^i_{\mathcal{A}}) \coloneqq c_{\mathcal{A}} \leq u_{P^C_{i+1}}$;

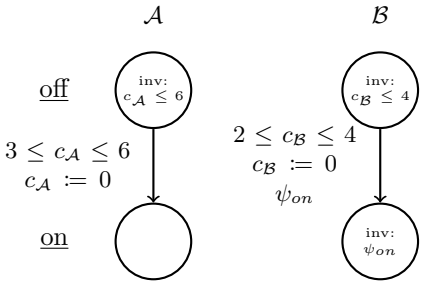


Figure 9.2: Example of TASV corresponding to a local requirement.

- (discrete transition) *for each phase* P_i^C *(but the last) of* C*, it holds that* $(v_\mathcal{A}^i, \{c_\mathcal{A}\}, \Xi_i^C, \Phi_{P_{i+1}^C} \wedge \Psi_{P_{i+1}^C}, v_\mathcal{A}^{i+1}) \in T_\mathcal{A}$*, where* $\Xi_i^C \coloneqq l_{P_{i+1}^C} \leq c_\mathcal{A} \leq u_{P_{i+1}^C}$.
- (state deps) *for each phase* P_i^C *of* C*, it holds that* $inv_\mathcal{A}^{loc}(v_\mathcal{A}^i) \coloneqq \Psi_{P_i^C}$*;*

where $\Phi_P \coloneqq \phi_P[(d, j) \mapsto (l_d = v_j)]$*, for each phase* P *(the same holds for* Ψ*);*

Example. Consider Fig. 9.1a: the corresponding TASV is depicted in Fig. 9.2. Each phase of each local requirement corresponds to a location of the corresponding TASV; in the example, phase *off* is mapped into location off. The first locations of automata $\mathcal{A}$ and $\mathcal{B}$ have attached the invariants $c_\mathcal{A} \leq 6$ and $c_\mathcal{B} \leq 4$, respectively. Automaton $\mathcal{A}$ proceeds to location *on* (corresponding to phase $A.on$) by a transition labelled with clock constraint $3 \leq c_\mathcal{A} \leq 6$ and clock reset $c_\mathcal{A} \coloneqq 0$. Since the second phase of local requirement A has no dependencies, the transition to on has no constraints on the location variables. The situation is different for automaton $\mathcal{B}$, for which the transition to on is labelled with $2 \leq c_\mathcal{B} \leq 4$ and $c_\mathcal{B} \coloneqq 0$, and also with $\psi_{on} \coloneqq (l_\mathcal{A} = on)$, that is the state dependency of phase $B.on$; moreover, ψ_{on} is also an invariant for the second location of automaton $\mathcal{B}$, since it is a state dependency.

Given a network $\mathcal{S} \coloneqq \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ of TASVs, the problem of *consistency checking* can be expressed as the reachability of location $(\mathcal{A}_1.last, \ldots, \mathcal{A}_n.last) \in V_\mathcal{S}$. A *deadlock* of a TASV $\mathcal{A}$ is defined as

a state $(v,t) \in V_{\mathcal{A}} \times \mathbb{R}$ such that $\mathcal{A}$ can take neither a timed nor a discrete transition from (v,t). We call *livelock* a state (v,t) such that $\mathcal{A}$ can take only timed transitions. The *compatibility checking* problem can be expressed as the problem of checking if there exists a trace of $\mathcal{S}$ such that (i) either the trace is finite and its final state is a deadlock of $\mathcal{S}$; we can check this property by adding a *sink* location to the TASV $\mathcal{S}$ to which all locations can transition to and by checking the reachability of it; (ii) or the trace is infinite and there exists a location $v \in V_{\mathcal{S}}$ and a point $k \geq 0$ such that $l_{\mathcal{S}} = v \neq (\mathcal{A}_1.last, \ldots, \mathcal{A}_n.last)$, for all the states after k in the trace, where the i^{th} component of v together with the time of the current state is a *livelock* for automata $\mathcal{A}_i$, for some $1 \leq i \leq n$. The second point is fundamental for local requirements featuring *infinite bounds*: in these automata, it is not sufficient to check for deadlocks, since a timed transition could be always enabled; instead, an illegal state can be described by a trace of the system that reaches a *livelock* whose location has no invariants attached and then stays constantly in this location. Having reached a livelock, the automaton can proceed only with timed moves: in particular, it can't proceed to the next location because its dependencies are violated. We can check the second point in this way: we first add a sink location $sink_v^{\mathcal{A}_i}$ for each location $v \in \mathcal{A}_i$ (and of course a transition from the latter to the former), for each $1 \leq i \leq n$, and we attach to it the invariant $\neg inv_{\mathcal{A}_i}^{loc}(v)$. Now, in the product S of these modified automata, we look for a trace such that, from a certain time point onwards, it stays constantly in a location $(l_1, \ldots, l_n)$ such that at least one l_i is a sink state. This property can be formalized in *Linear Temporal Logic* as $\mathsf{FG}(\bigvee_{1 \leq i \leq n, v \in \mathcal{A}_i} sink_v^{\mathcal{A}_i})$.

9.3.2 Encoding into SMT

We describe the encoding into SMT(DL) (Satisfiability Modulo Theory of Difference Logic) for the problems of *consistency checking* and *compatibility checking*. For all $1 \leq c \leq n$ and $1 \leq i \leq |c|$, we introduce the following variables: (i) $r_i^c \in \mathbb{B}$ represents the fact that phase i of local requirement c is *reachable*; (ii) $s_i^c = (t_i^c, p_i^c)$ represents the superdense time instant in which local requirement c enters phase i, where $t_i^c \in \mathbb{R}$ and $p_i^c \in \mathbb{N}$. We can compare two superdense-valued variables (t,p) and (t',p') with the lexicographical order, which we define as follows: $(t,p) \preceq (t',p')$ iff $t \leq t' \wedge (t = t' \rightarrow p \leq p')$. We

now give the set of (conjunctively related) constraints which form our SMT(DL) encoding.

Initialization. Each local requirement starts in its first phase at the same time, *i.e.*, the real time point 0. Hence, for all $1 \leq c \leq n$, we add the constraint $t_0^c = 0$.

Reachability. For all local requirements c and all phases i, it holds that if $i-1$ is not reachable then so is phase i, *i.e.*, $\neg r_{i-1}^c \rightarrow \neg r_i^c$. Moreover, we require the monotonicity over time, *i.e.*, $r_i^c \rightarrow (s_{i-1}^c \prec s_i^c)$.

Bounds. For all local requirements c and all phases i, c can move to i only if it respects the bounds $[l_i^c, u_i^c]$ of phase i, namely $r_i^c \rightarrow (l_i^c \leq t_i^c - t_{i-1}^c \leq u_i^c)$. If $u_i^c = \infty$, then we add only the left-most inequality.

Signal and State dependencies. Consider a local requirement c and one of its phases i. Since we have only a *finite* number of phases, we can preprocess both signal and state dependencies to remove from them all negations, as explained in Section A.3.1; this means that every atom in ϕ_i^c and ψ_i^c occurs positive.

We want c to reach i only if all its signal and state dependencies are satisfied. For signal dependencies, we require the time point in which c enters i to be strictly greater[1]than the time point of the entry of the target phase and smaller than or equal to the time point of the exit of the target phase.

$$r_i^c \rightarrow \phi_i^c[(d,j) \mapsto (r_j^d \wedge s_j^d \prec s_i^c \preceq s_{j+1}^d)]$$

Moreover, we have to guarantee that the state dependencies hold as well. In particular, if phase i is reachable, then surely the time point in which c enters i has to be strictly greater than the time point in which the other local requirement reaches the target phase.

$$r_i^c \rightarrow \psi_i^c[(d,j) \mapsto (r_j^d \wedge s_j^d \prec s_i^c)]$$

Since state dependencies are invariant properties, *i.e.*, they have to hold for each time instant a local requirement is in a particular phase,

[1]This allows us to model the observability of the events: c first observes d entering its phase j and *then* moves.

if one state dependency is violated at some time point of phase $i-1$, then phase i is not reachable. The contrapositive means that if phase i is reachable, then the state dependencies of phase $i-1$ have to be invariant for phase $i-1$, namely:

$$r_i^c \rightarrow \forall \widetilde{s}(s_{i-1}^c \preceq \widetilde{s} \preceq s_i^c \rightarrow \psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge s_j^d \prec \widetilde{s} \preceq s_{j+1}^d)]) \tag{9.1}$$

Illegal States. If phase i of local requirement c is not reachable, *i.e.*, i is an *illegal state*, then there exists a time point s_{ill}^c such that, for all the next (remaining) time points $\overline{s}$ between s_{ill}^c and the upperbound of the transition, at least one dependency is not satisfied.

$$\begin{aligned}&(r_{i-1}^c \wedge \neg r_i^c) \rightarrow \\ &\quad \exists s_{ill}^c \forall \overline{s}(s_{ill}^c \preceq \overline{s} \preceq s_{i-1}^c + u_{i-1}^c \rightarrow \text{VIOLATION}(\overline{s}))\end{aligned} \tag{9.2}$$

where

$$\text{VIOLATION}(\overline{s}) \coloneqq \tag{9.3}$$

$$\neg \phi_i^c[(d,j) \mapsto (r_j^d \wedge s_j^d \prec \overline{s} \preceq s_{j+1}^d)] \vee \tag{9.4}$$

$$\neg \psi_i^c[(d,j) \mapsto (r_j^d \wedge s_j^d \prec \overline{s})] \vee \tag{9.5}$$

$$\exists \widetilde{s}(s_{i-1}^c \preceq \widetilde{s} \preceq \overline{s} \wedge \neg \psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge s_j^d \prec \widetilde{s} \preceq s_{j+1}^d)]) \tag{9.6}$$

We interpret $\overline{s} \preceq s_{i-1}^c + u_i^c$ as $\forall \overline{p}(\overline{s} \preceq s_{i-1}^c + (u_i^c, \overline{p}))$ and the $+$ symbol as the pairwise sum. In the case the upperbound of the transition is infinite, we simply do not add the $\overline{s} \preceq s_{i-1}^c + u_i^c$ inequality. We refer to the conjunction of all these constraints as W.

For *consistency checking*, we define

$$\text{END} \coloneqq \bigwedge_{1 \leq c \leq n} r_{|c|}$$

and we call W^{cons} the conjunction of W with END. We check consistency by checking the satisfiability of W^{cons}.

For *compatibility checking*, we define

$$\text{ILL} \coloneqq \bigvee_{\substack{1 \leq c \leq n \\ 1 \leq i \leq |c|}} \neg r_i^c$$

and we call $\mathrm{W}^{\mathrm{ill}}$ the conjunction of W with ILL. We check the existence of an illegal state in the system by checking the satisfiability of $\mathrm{W}^{\mathrm{ill}}$, *i.e.*, $\mathrm{W}^{\mathrm{ill}}$ is satisfiable iff the local requirements are *not* compatible.

Strict Semantics In the strict semantics setting, we forbid two events to occur at the same real-time point. For strict semantics, the encoding is equal to W except that we interpret $\prec$ and $\preceq$ as $<$ and $\leq$, respectively, and all the s_i^c variables as single real-valued variables $t_i^c \in \mathbb{R}$. We call S this encoding and we define $\mathrm{S}^{\mathrm{cons}}$ and $\mathrm{S}^{\mathrm{ill}}$ as above.

Finite bounds and convex dependencies. Despite being very close to the problem formalization, the W encoding features a high number of quantifications, also in alternation; therefore, in the general case, it is very burdensome for an SMT solver to first perform quantifier elimination on W and then to solve the resulting formula. Nevertheless, if we make some restrictions on the type of local requirements we consider, we are able to remove *upfront* all the quantifiers from W, without the need to use quantifier elimination techniques. In fact, suppose we consider only local requirements with *finite bounds* and *convex* state dependencies (see Section 9.2). We call $\widehat{\mathrm{W}^{\mathrm{ill}}_{\mathrm{fin}}}$ the encoding equal to W except that Eq. (9.1) is replaced by:

$$r_i^c \rightarrow \psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge s_i^c \preceq s_{j+1}^d)] \tag{9.7}$$

and we add the following constraint:

$$(r_{i-1}^c \wedge \neg r_i^c) \rightarrow (t_i^c = t_{i-1}^c + u_{i-1}^c) \tag{9.8}$$

and we replace Eq. (9.2) with:

$$(r_{i-1}^c \wedge \neg r_i^c) \rightarrow \mathrm{WEAKVIOL}(t_i^c) \tag{9.9}$$

where:

$$\begin{aligned} \mathrm{WEAKVIOL}(t_i^c) \;:=\; & \neg\phi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq t_i^c < t_{j+1}^d)] \vee \\ & \neg\psi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \preceq t_i^c)] \vee \\ & \neg\psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge t_i^c \leq t_{j+1}^d)]) \end{aligned} \tag{9.10}$$

We can prove that

Proposition 19 (Equisatisfiability)**.** $\mathrm{W}^{\mathrm{ill}}$ *and* $\widehat{\mathrm{W}^{\mathrm{ill}}_{\mathrm{fin}}}$ *are equisatisfiable for every set of local requirements with only finite bounds and convex dependencies.*

Notably, there are no quantifiers in $\mathrm{W}^{\mathrm{ill}}_{\mathrm{fin}}$: as said before, this makes the encoding dramatically more efficient with respect to W: in Section 9.5, we will consider only local requirements of this type. The details of the proofs are reported in Sections A.3 and A.4 in which, given that the proofs are a bit involved, we proceed incrementally, showing first how we can remove upfront the quantifiers in case of finite bounds with strict semantics, then in the case with weak semantics and finally in case of convex dependencies.

9.3.3 Optimization of the Encoding

In order to let illegal state checking in the SMT-based approach to perform better even in the case of system with a very high number of dependencies, we propose the following improvement to the encoding described in Section 9.3.2. Let $DEPS^c_i$ be the set of all the dependencies of phase i of component c; formally:

$$DEPS^c_i \coloneqq \{(d,j) \mid (d,j) \in \phi^c_i \cup \psi^c_i \cup \psi^c_{i-1}\}$$

The idea is the following one: instead of looking for (describing) all the possibile configurations in which phase (c,i) can go to an illegal state, we only look for the situation in which all the other phases which (c,i) depends on *are reachable* and (c,i) goes into an illegal state *only* because of its own dependencies. In other words, the improvement proposed here follows this rationale: if a phase (d,j) which (c,i) depends on is not reachable, for the sake of illegal state checking it doesn't matter if we do not look for the situation in which (c,i) is not reachable because of that, since we have already found an illegal state, *i.e.*, the one of (d,j). Hence, Eq. (9.2) can be modified as follows:

$$(r^c_{i-1} \wedge \neg r^c_i) \rightarrow \bigwedge_{(d,j)\in DEPS^c_i} r^d_j \wedge \exists s^c_{ill} \forall \overline{s} (s^c_{ill} \preceq \overline{s} \preceq s^c_{i-1} + u^c_{i-1} \rightarrow \mathrm{VIOLATION}'(\overline{s}))$$

where $\mathrm{VIOLATION}'(\overline{s})$ is obtained from $\mathrm{VIOLATION}(\overline{s})$ by replacing all the boolean variables r^d_j with True.

9.4 Parameter Synthesis

In this section, we tackle the *parameter synthesis* problem, *i.e.*, computing the set of *all stronger* local requirements (as defined in Definition 57) of the initial local requirements such that their composition is *compatible*. We solve this problem by reducing it to a *parameter synthesis* problem (see [51] for a more detailed description); given a local requirement C, its corresponding *parametric local requirement* $\langle C, \pi \rangle$ is defined as C (see Section 9.2), except that the bounds l_P and u_P of each phase P are now the parameters $\overline{l}_P$ and $\overline{u}_P$, respectively, and $\pi \coloneqq \{\overline{l}_P \mid P \text{ is a phase of } C\} \cup \{\overline{u}_P \mid P \text{ is a phase of } C\}$. Given a set of local requirements $S = \{C_1, \ldots, C_n\}$, we write $\langle S, \Pi \rangle$ for its parametric version $\{\langle C_1, \pi_1 \rangle, \ldots, \langle C_n, \pi_n \rangle\}$, where the set of parameters is defined as $\Pi \coloneqq \bigcup_{i=1}^{n} \pi_i$. A parameter valuation $\gamma : \Pi \to \mathbb{Q}$ assigns a rational value to each parameter; moreover, for each $1 \leq i \leq n$, it also induces a (concrete) local requirement $\langle C_i, \gamma(\pi_i) \rangle$, obtained from $\langle C_i, \pi_i \rangle$ by replacing every parameter $p \in \pi_i$ with the concrete value $\gamma(p)$. In the same way, we can define the *concrete* version $\langle S, \gamma(\pi) \rangle$ of $\langle S, \pi \rangle$. γ is said to be *feasible for* S if $\langle C_i, \gamma(\pi_i) \rangle$ is a *stronger* local requirement of C_i, for all $1 \leq i \leq n$, and $\langle S, \gamma(\pi) \rangle$ is *compatible*. A *feasible region* is a set $\mathcal{R} \coloneqq \{\gamma \mid \gamma \text{ is feasible for } S\}$. Also in this case, we can either use parameter synthesis algorithms over timed automata [7] or reduce the problem to SMT(LRA); we focus on the latter and in particular, we will synthesize a symbolic representation of the region $\mathcal{R}$, namely an SMT formula $\varphi_{\mathcal{R}}$ with the following property: $\gamma \models \varphi_{\mathcal{R}}$ iff $\gamma \in \mathcal{R}$, for each valuation γ.

Let $\overline{\mathrm{W}^{\mathrm{ill}}}$ be the encoding equal to $\mathrm{W}^{\mathrm{ill}}$ except that each number l_i^c (resp. u_i^c) is replaced with the variable $\overline{l}_i^c$ (resp. $\overline{u}_i^c$) and each phase is required to have finite bounds. We define the sets of variables $\mathbb{R} \coloneqq \{r_i^c \mid c \in S,\ i \text{ is a phase of } c\}$ and $\mathbb{S} \coloneqq \{s_i^c \mid c \in S,\ i \text{ is a phase of } c\}$: these are the variables we are going to remove by means of quantifier elimination. Finally, we define:

$$\text{DOMAIN} \coloneqq \bigwedge_{\substack{1 \leq c \leq n \\ 1 \leq i \leq |c|}} (\overline{l}_i^c \geq 0 \wedge \overline{l}_i^c \leq \overline{u}_i^c)$$

$$\text{REFINE} \coloneqq \bigwedge_{\substack{1 \leq c \leq n \\ 1 \leq i \leq |c| \\ u_i^c \neq \infty}} (a_i^c \leq \overline{l}_i^c \wedge \overline{u}_i^c \leq b_i^c)$$

The symbolic representation of the *feasible region* $\mathcal{R}$ is given by:

$$\text{SYNTH} \;:=\; \text{DOMAIN} \wedge \text{REFINE} \wedge \neg\exists \mathbb{S}, \mathbb{R}\left(\overline{\mathrm{W}^{\mathrm{ill}}}\right) \qquad (9.11)$$

By removing the existential quantification on $\mathbb{S}$ and $\mathbb{R}$ (this can be done by means of quantifier elimination techniques), we obtain a quantifier-free formula over the variables in Π. By construction, we have that each model γ of SYNTH is a feasible valuation, and viceversa. Therefore SYNTH is the symbolic representation of the feasible region $\mathcal{R}$.

9.5 Experimental Evaluation

We implemented the encoding described in Section 9.3.2 in a tool called TRICKER (Timing Requirements Integration ChecKer) [2], which uses MATHSAT [55] as the backend SMT engine. We compared TRICKER with UPPAAL [11] and TIMED-NUXMV [50], both using the automata-based encoding described in Section 9.3.1.

The test set is partitioned into three categories: (i) `bounded convex` contains only systems with finite bounds and convex state dependencies; (ii) `bounded` contains systems with only finite bounds, but with arbitrary dependencies (not necessarily convex); (iii) `general` contains systems with infinite bounds and arbitrary dependencies (this is the most general fragment). Each category in turn consists of ca. 500 randomly-generated systems, divided in 10 subcategories, namely 2C3P, 2C15P, 5C3P, 5C20P, 10C4P, 10C30P, 50C5P, 50C30P, 100C3P and 100C10P, where NCMP is the category containing only systems with N components and (approximately) M phases for each component. Inside each sub-category, each benchmark is randomly generated, meaning that the exact number of phases for each component and the density of its signal and state dependencies was chosen uniformly at random. For each benchmark, we compare the time spent by the three tools on the *consistency checking* and *compatibility checking* problems. We ran the experiments on a cluster of Linux machines with a 2.27GHz Xeon CPU, with a timeout of 360 seconds for each instance.

We consider first the `bounded convex` category. Fig. 9.3 shows the comparison of TRICKER with TIMED-NUXMV and UPPAAL on

[2] `http://users.dimi.uniud.it/~luca.geatti/tricker.html`

the two verification problems. In both cases, TIMED-NUXMV runs the infinite-state variant of IC3 described in [53] after discretizing the timed automata. As for UPPAAL, we verify a property in the form $EF\varphi$, where φ is a Boolean formula. For both problems, the SMT-based approach implemented in TRICKER outperforms the model checkers. While there are a number of instances for which the model checkers perform better than TRICKER (especially for UPPAAL), the latter overall solves a significantly larger amount of problems within the timeout, showing a clear improvement in scalability. This can be seen also in the survival plots comparing the three tools with the Virtual Best Solver (*vbs* for short). We can make similar considerations for the `bounded` and `general` categories, shown respectively in Fig. 9.4 and Fig. 9.5. (Note that for the `general` case, we could not evaluate UPPAAL as it does not support the verification of fairness properties.) We remark that we did not note any kind of correlation between the number of signal or state dependencies in the benchmarks and the time spent by the solver. Finally, Fig. 9.6 shows the correlation between the memory (measured in MB) and the time (in seconds) spent by TRICKER on consistency and compatibility checking, respectively.

We also evaluated the parameter synthesis algorithm described in Section 9.4. Since UPPAAL currently does not support parameter synthesis for timed automata, we could not include it in the comparison. We therefore compared TRICKER with TIMED-NUXMV, for which we used the PARAM-IC3 parameter synthesis algorithm described in [51]. The algorithm is based on the inverse method, *i.e.*, it finds a bad configuration for the parameters and it tries to generalize it, maximizing the set of bad parameters removed from the current approximation of the region. We took all the consistent benchmarks of the previous test sets, which amounts to approximately 100 instances (note that for each instance of the class NCMP, the number of parameters is $\approx 2 \cdot N \cdot M^3$). The results of the comparison are shown in Fig. 9.7; as in the previous cases, TRICKER shows better performance and scalability than PARAM-IC3, though there are several instances for which synthesis via quantifier elimination is still very expensive.

[3] recall that both the lower and the upper bounds are parameters.

9.6 Conclusions

In this chapter, we defined verification and synthesis problems of industrial relevance focused on the decomposition of startup requirements into local timing constraints and dependencies on components. Namely, we addressed the problem of checking if the local requirements are free of integration errors (i.e., consistent and compatible), and the problem of synthesizing the region of refinements of the original specification that are error free. The problem can be naturally translated into model checking and synthesis problems for timed automata with shared variables. Exploiting the structure of the requirements, we provide an encoding into SMT where consistency and incompatibility correspond to satisfiability queries, while synthesis is resolved by means of quantifier elimination.

In the future, we will consider various directions, such as extending the applicability of the approach to more general structures with loops, enriching the synthesis problem with cost functions to repair the specification driven by specific industrial goals, and considering more complex representations of signals exchanged between components.

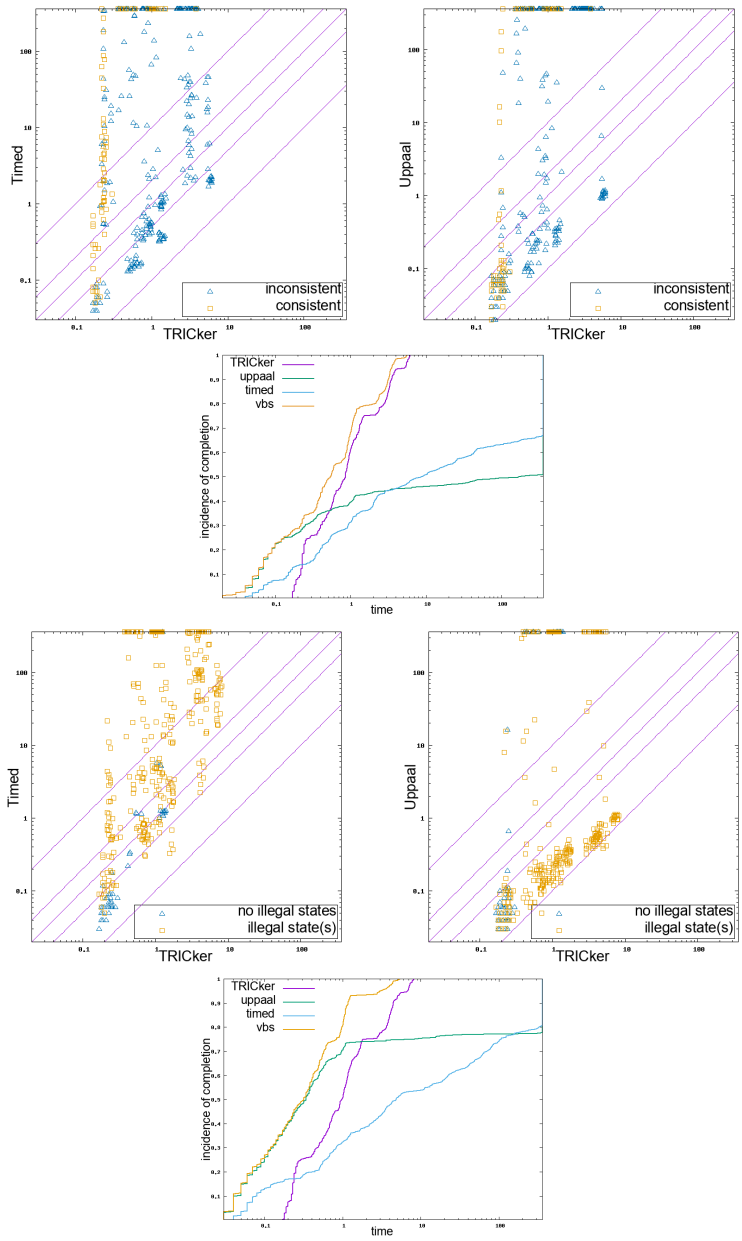


Figure 9.3: Comparison on the `bounded convex` category (*consistency checking* on the first three plots and *compatibility checking* on the last three plots).

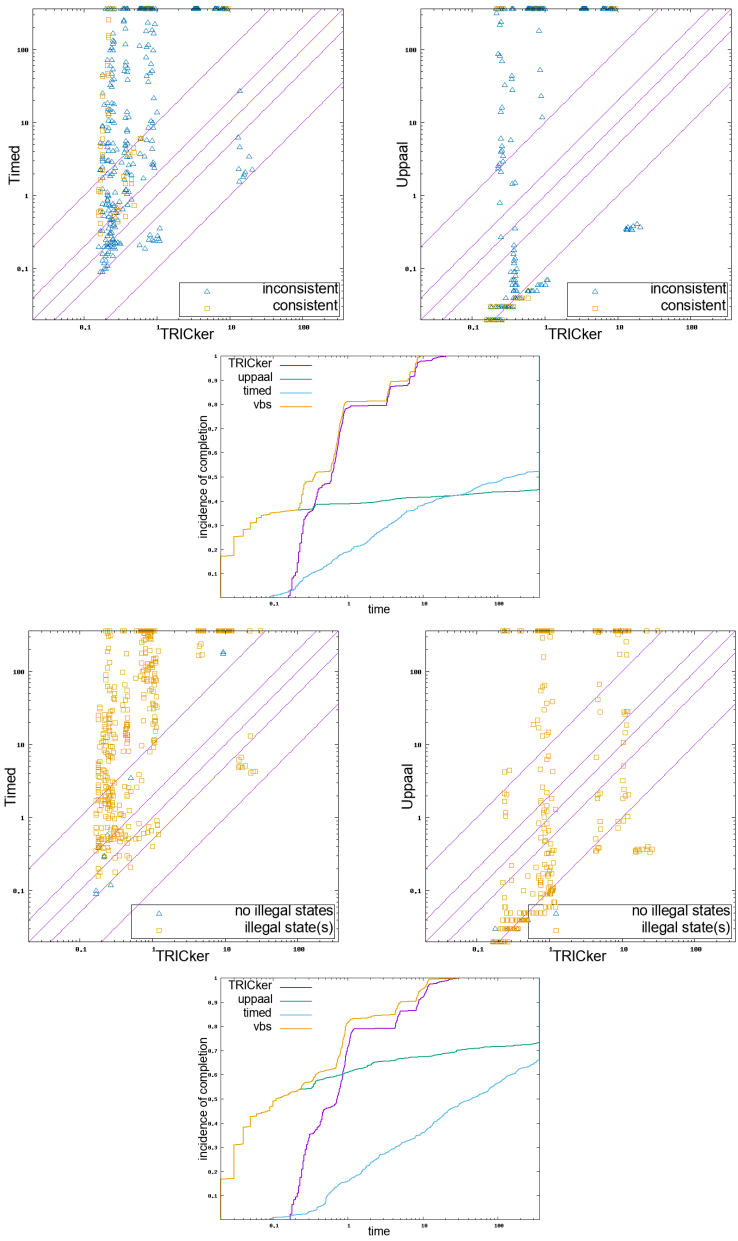


Figure 9.4: Comparison on the `bounded` category (*consistency checking* on the first three plots and *compatibility checking* on the last three plots).

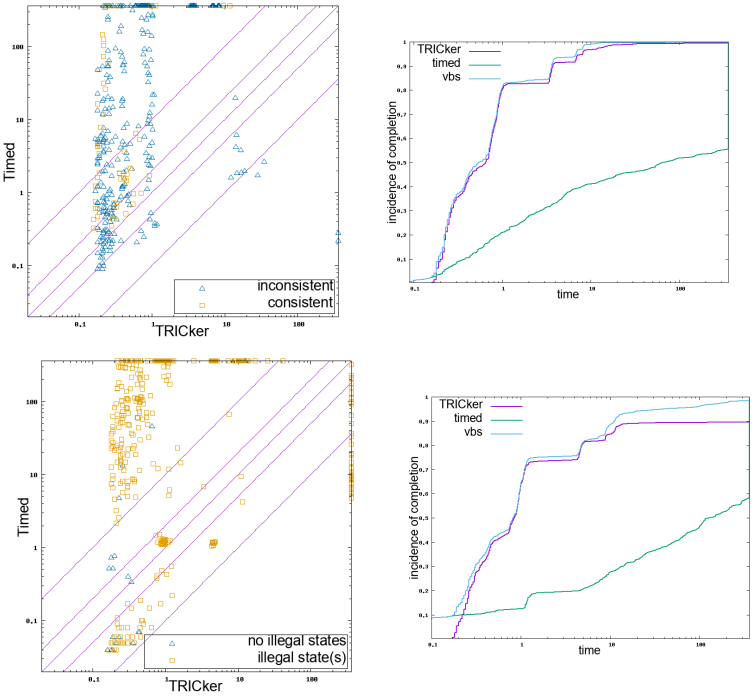


Figure 9.5: Comparison on the `general` category (*consistency checking* on the first row and *compatibility checking* on the second).

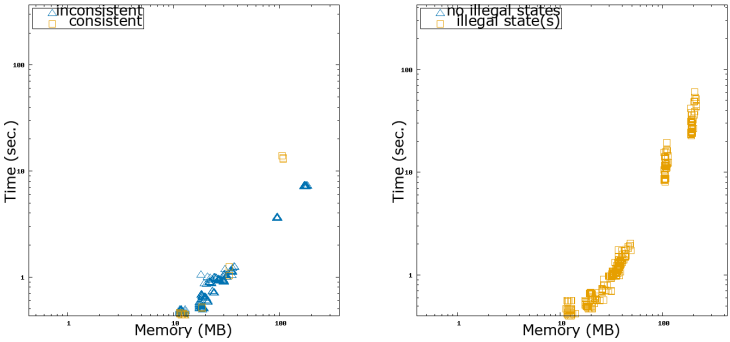


Figure 9.6: Comparison between time and memory consuption of TRICKER (*consistency checking* on the left and *compatibility checking* on the right).

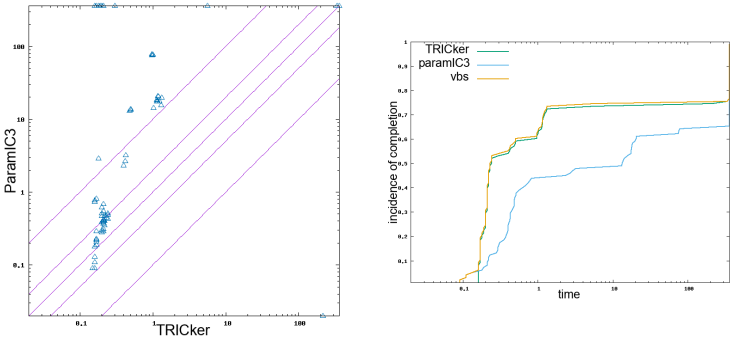


Figure 9.7: Comparison on parameter synthesis.

CHAPTER

10

CONCLUSIONS

The main topic of this thesis is about temporal logics specifications. We introduced some temporal logics, we studied their expressive power, and we give symbolic algorithms for (i) the satisfiability problem of LTL+P, (ii) the realizability problem of those fragments, and (iii) and the compatibility problem of real-time specifications. Particular attention has been devoted to follow the theoretical results (whenever this was possible) with an algorithmic study and empirical evaluations.

Theory

We introduced three fragments of the Linear Temporal Logic with Past (LTL+P), that is: (i) $\mathsf{LTL}_{\mathsf{EBR}}$+P; (ii) $\mathsf{LTL}_{\mathsf{EBR}}$; and (iii) GR-EBR.

In Section 3.1, we proved that $\mathsf{LTL}_{\mathsf{EBR}}$+P is expressively complete with respect to the safety fragment of LTL+P, that is any safety property that is definable in LTL+P is definable in $\mathsf{LTL}_{\mathsf{EBR}}$+P as well.

A peculiarity of $\mathsf{LTL}_{\mathsf{EBR}}$+P is that it features *past operators*, that

is operators for looking into the past starting from a time point of a sequence. Past operators play a crucial role in the completeness proof of $\mathsf{LTL_{EBR}}$+P. We considered the problem of establishing the expressive power of $\mathsf{LTL_{EBR}}$, that is $\mathsf{LTL_{EBR}}$+P devoid of past operators. We proved that $\mathsf{LTL_{EBR}}$ is *strictly less expressive* than full $\mathsf{LTL_{EBR}}$+P (Section 3.2). This is somehow surprising, since for LTL+P the presence or the absence of past operators does not affect the expressive power. Therefore our result show that past operators, despite being not important for the expressive power of LTL+P, can play a crucial role for the expressiveness of *fragments* of LTL+P, like for instance $\mathsf{LTL_{EBR}}$+P.

In Section 3.3, we compared the logic of GR-EBR, defined as an extension of $\mathsf{LTL_{EBR}}$+P able to define properties beyond the safety fragment, with the Temporal Hierarchy of Manna and Pnueli [140], showing that its expressive power stands between the one of R(1) (Reactivity(1)) and GR(1) (Generalized Reactivity(1)).

In Chapter 4, we identified a *syntactical* fragment of the first-order logic of one successor (S1S[FO]) and proved that it is expressively complete with respect the semantically safety fragment of S1S[FO], and thus also complete with respect to the semantical safety fragment of LTL+P. This result provides a first-order characterization of LTL-definable safety languages, and joins Kamp's Theorem for the equivalence between LTL and the first-order logic of one successor. In addition, this result allows us to prove that Safety-LTL (*i.e.*, LTL with only universal temporal operators) is expressively complete as well. This proof seems not to be very much known in the literature, as some authors presented the problem as open as lately as 2021 [201, 71].

Problems and Algorithms

We studied the satisfiability problem of LTL+P specifications (Chapter 6). We gave a symbolic encoding of the one-pass and tree-shaped tableau system by Reynolds [163]. The algorithm encodes all the branches of the tableau up to depth k by means of a Boolean formula and uses efficient SAT-solvers for deciding their satisfiability. We implemented this algorithm in a tool called BLACK with the goals of stability, reliability, and efficiency in mind. We compared the symbolic encoding of the tableau with the (explicit-state) tableau of Reynolds. The experimental evaluation shows a clear improvement

in solving time.

In Chapter 7, we gave a fully symbolic algorithm for the realizability problem of $\mathsf{LTL}_{\mathsf{EBR}}$+P specifications. The algorithm comprises two steps: (i) the construction of a symbolic safety automaton starting from the initial formula; (ii) the solving of a safety game over the arena represented by the safety automaton. We implemented this approach in a prototype tool called EBR-LTL-SYNTH and compared the performance with other tools for realizability of specifications of (fragments of) LTL+P, which rely all on an explicit-state representation of the automaton. The outcomes show that there is a clear improvement in time and memory consumption using the symbolic method: in some cases, our symbolic algorithm could solve some instances that were prohibitively large for the other tools based on an explicit-state representation.

Exploiting the algorithm for $\mathsf{LTL}_{\mathsf{EBR}}$+P realizability, we gave a symbolic algorithm for the realizability problem of GR-EBR specifications (Chapter 8). The algorithm performs a safety reduction in order to reduce the original problem to a sequence of safety synthesis (sub-)problems. The completeness of the procedure was proved by introducing a general framework for guaranteeing completeness of arbitrary fragments of LTL+P and then by instantiating it for the case of GR-EBR. Interestingly, our framework proves that if a reduction is complete for the model checking problem, then it is also complete for the realizability problem. Moreover, the framework can easily be used for proving the completeness of bounded synthesis [92], a well-known approach for realizability. Also in this case, we implemented the approach in a prototype tool called GRACE, showing some improvements with respect to competitor tools.

Last but not least, in Chapter 9, we considered timed temporal specifications, that is requirements expressing not only the ordering between events but also the amount of time elapsed between two events. In particular, we defined and formalized the *compatibility problem of timed requirements*: given a set of timed specifications, the problem is to check whether all possible implementations of each requirement can be *composed* to the others in such a way to guarantee the completion of each of them (in some sense, checking the absence of deadlocks in the all implementation's compositions). We gave two encodings of the problem, one into SMT(LRA) (Satisfiability Modulo the Theory of Linear Real Arithmetic) and one into model checking of timed automata. The outcomes of the experi-

mental evaluations of our implementation (in the TRICKER prototype tool) shows that the encoding into SMT(LRA) is much more efficient than the other.

Future directions

We list some open points regarding the research that we have done in this thesis.

The exact expressive power of GR-EBR is still unknown. In Section 3.3, we proved that GR-EBR is at least as expressive as R(1) and at most as expressive as GR(1). However, we do not know yet if the two inclusions (or which of the two) are strict. In the first place, as a starting point for addressing the problem, one has to understand whether R(1) and GR(1) are or are not expressively equivalent.

In Chapter 4 we proved that the Safety-FO fragment captures exactly all safety languages that are definable in LTL+P (or equivalently in the first-order theory of one successor). Different equivalent characterisations of LTL+P are known, in terms of (i) first-order logics, (ii) regular expressions, (iii) automata, and (iv) monoids (they have been summarised by Thomas in [192]). The work done in Chapter 4 focuses on the first item, but for LTL-definable safety languages. A natural follow-up would be to investigate the other items, looking for what kind of automata (resp., regular expressions, monoids) captures exactly safety and co-safety LTL-definable languages. While on finite traces simple characterizations in terms of automata and syntactic monoids exist, the infinite-traces scenario is more complex. In fact:

- there exists a characterization of LTL in terms of counter-free automata [149];
- moreover, an automata-based characterization for safety ω-regular languages seems not to be difficult (see *e.g.*, terminal automata [186, 39])

However, in order to combine these two characterizations and obtain, let's say, a counter-free terminal automata starting from any safety LTL-definable ω-language, one must apply the two above-mentioned characterizations/transformation to the very same automaton. For this reason, one needs a theorem ensuring the existence of a canonical/minimal ω-automaton for each ω-regular language.

The BLACK satisfiability checker described in Chapter 6 is based on a SAT-encoding (described in the same chapter) of Reynolds'tableau. One of the advantages of this tableau system is that it is very simple to extend to different temporal logics. For example, it has been extended to Timed Propositional Temporal Logic (TPTL) in [100]. An interesting question is whether a SAT- or SMT-encoding of the one-pass and tree-shaped tableau for TPTL proposed in [100] or for other more expressive temporal logics is possible.

Regarding the realizability problem of $\mathsf{LTL}_{\mathsf{EBR}}$+P (Chapter 7), there are several interesting open points:

- The fully-symbolic automata construction can be used not only for realizability but also for model-checking. In particular, the deterministic ad symbolic safety automata (that we obtain with our algorithm) may provide many benefits for symbolic model checking, since the search of the state space could benefit from a deterministic representation of the automaton for the formula [173].

- In $\mathsf{LTL}_{\mathsf{EBR}}$+P, the bounded operators are short-cuts for the equivalent expansions using the *next* operators. It would be interesting to consider the bounds as *primitives*, represented with a logarithmic encoding. This would allow: (i) on the one hand, to obtain exponentially more succinct specifications; and (ii) on the other hand, to open the possibility to the underlying algorithm of expoiting the symbolic bounds for a more clever automata construction.

- Last but not least, we aim at checking whether the synthesis problem for more expressive logics, like, for instance, LTL, can be reduced to the synthesis problem for $\mathsf{LTL}_{\mathsf{EBR}}$+P, for example checking whether it is possible to use $\mathsf{LTL}_{\mathsf{EBR}}$+P for solving the safety problems originated from *bounded synthesis* techniques.

BIBLIOGRAPHY

[1] Abate, P., Goré, R., and Widmann, F. An On-the-fly Tableau-based Decision Procedure for PDL-satisfiability. *Electronic Notes in Theoretical Computer Science 231* (2009), 191–209.

[2] Abel, A., and Reineke, J. Memin: Sat-based exact minimization of incompletely specified mealy machines. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2015), IEEE, pp. 94–101.

[3] Akers, S. B. Binary decision diagrams. *IEEE Transactions on computers 27*, 06 (1978), 509–516.

[4] Alpern, B., and Schneider, F. B. Defining liveness. *Information processing letters 21*, 4 (1985), 181–185.

[5] Alur, R., and Dill, D. L. A theory of timed automata. *Theoretical computer science 126*, 2 (1994), 183–235.

[6] André, É. Parametric Deadlock-Freeness Checking Timed Automata. In *Theoretical Aspects of Computing - ICTAC 2016*

- 13th International Colloquium, Taipei, Taiwan, ROC, October 24-31, 2016, Proceedings (2016), pp. 469–478.

[7] André, É., Chatain, T., Fribourg, L., and Encrenaz, E. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science 20*, 05 (2009), 819–836.

[8] Astefanoaei, L., Rayana, S. B., Bensalem, S., Bozga, M., and Combaz, J. Compositional Invariant Generation for Timed Systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings* (2014), pp. 263–278.

[9] Astefanoaei, L., Rayana, S. B., Bensalem, S., Bozga, M., and Combaz, J. Compositional Verification of Parameterised Timed Systems. In *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings* (2015), pp. 66–81.

[10] Babiak, T., Křetínskỳ, M., Řehák, V., and Strejček, J. Ltl to büchi automata translation: Fast and more deterministic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (2012), Springer, pp. 95–109.

[11] Behrmann, G., David, A., Larsen, K. G., Håkansson, J., Pettersson, P., Yi, W., and Hendriks, M. Uppaal 4.0.

[12] Bertello, M., Gigante, N., Montanari, A., and Reynolds, M. Leviathan: A new ltl satisfiability checking tool based on a one-pass tree-shaped tableau. In *IJCAI* (2016), pp. 950–956.

[13] Bertello, M., Gigante, N., Montanari, A., and Reynolds, M. Leviathan: A new LTL satisfiability checking tool based on a one-pass tree-shaped tableau. In *Proc. of the 25th International Joint Conference on Artificial Intelligence* (2016), IJCAI/AAAI Press, pp. 950–956.

[14] BETH, E. W. Semantic entailment and formal derivability.

[15] BIERE, A., ARTHO, C., AND SCHUPPAN, V. Liveness checking as safety checking. *Electronic Notes in Theoretical Computer Science 66*, 2 (2002), 160–177.

[16] BIERE, A., CIMATTI, A., CLARKE, E., AND ZHU, Y. Symbolic model checking without bdds. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (1999), Springer, pp. 193–207.

[17] BIERE, A., CIMATTI, A., CLARKE, E. M., STRICHMAN, O., AND ZHU, Y. Bounded model checking.

[18] BIERE, A., HELJANKO, K., JUNTTILA, T. A., LATVALA, T., AND SCHUPPAN, V. Linear encodings of bounded LTL model checking. *Log. Methods Comput. Sci. 2*, 5 (2006).

[19] BIERE, A., HELJANKO, K., AND WIERINGA, S. Aiger 1.9 and beyond. *Available at fmv. jku. at/hwmcc11/beyond1. pdf* (2011).

[20] BIERE, A., HEULE, M., AND VAN MAAREN, H. *Handbook of satisfiability*, vol. 185. IOS press, 2009.

[21] BLOEM, R., CAVADA, R., PILL, I., ROVERI, M., AND TCHALTSEV, A. Rat: A tool for the formal analysis of requirements. In *International Conference on Computer Aided Verification* (2007), Springer, pp. 263–267.

[22] BLOEM, R., CHATTERJEE, K., GREIMEL, K., HENZINGER, T. A., AND JOBSTMANN, B. Robustness in the presence of liveness. In *International Conference on Computer Aided Verification (CAV)* (2010), Springer, pp. 410–424.

[23] BLOEM, R., CIMATTI, A., GREIMEL, K., HOFFEREK, G., KÖNIGHOFER, R., ROVERI, M., SCHUPPAN, V., AND SEEBER, R. Ratsy–a new requirements analysis tool with synthesis. In *International Conference on Computer Aided Verification* (2010), Springer, pp. 425–429.

[24] BLOEM, R., GALLER, S., JOBSTMANN, B., PITERMAN, N., PNUELI, A., AND WEIGLHOFER, M. Automatic hardware synthesis from specifications: A case study. In *2007 Design,*

Automation & Test in Europe Conference & Exhibition (2007), IEEE, pp. 1–6.

[25] Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., and Saar, Y. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences 78*, 3 (2012), 911–938.

[26] Bloem, R., Könighofer, R., and Seidl, M. Sat-based synthesis methods for safety specs. In *International Conference on Verification, Model Checking, and Abstract Interpretation* (2014), Springer, pp. 1–20.

[27] Boole, G. *The Laws of Thought*. Dover, New York (original edition 1854), 1957.

[28] Bradley, A. R. Sat-based model checking without unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation* (2011), Springer, pp. 70–87.

[29] Bradley, A. R. Understanding ic3. In *International Conference on Theory and Applications of Satisfiability Testing* (2012), Springer, pp. 1–14.

[30] Brafman, R. I., and De Giacomo, G. Planning for LTLf/LDLf goals in non-markovian fully observable nondeterministic domains. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence* (2019), S. Kraus, Ed., ijcai.org, pp. 1602–1608.

[31] Brafman, R. I., De Giacomo, G., and Patrizi, F. LTLf/LDLf non-markovian rewards. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence* (2018), S. A. McIlraith and K. Q. Weinberger, Eds., AAAI Press, pp. 1771–1778.

[32] Brayton, R., and Mishchenko, A. Abc: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification (CAV)* (2010), Springer, pp. 24–40.

[33] Bryant, R. E. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR) 24*, 3 (1992), 293–318.

[34] BUCHI, J. On a decision method in restricted second-order arithmetics. *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science, 1960* (1960).

[35] BÜCHI, J. R. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly 6*, 1-6 (1960), 66–92.

[36] BUCHI, J. R., AND LANDWEBER, L. H. Solving sequential conditions by finite-state strategies. In *The Collected Works of J. Richard Büchi*. Springer, 1990, pp. 525–541.

[37] BURCH, J. R., CLARKE, E. M., MCMILLAN, K. L., DILL, D. L., AND HWANG, L.-J. Symbolic model checking: 1020 states and beyond. *Information and computation 98*, 2 (1992), 142–170.

[38] CAVADA, R., CIMATTI, A., DORIGATTI, M., GRIGGIO, A., MARIOTTI, A., MICHELI, A., MOVER, S., ROVERI, M., AND TONETTA, S. The nuxmv symbolic model checker. In *International Conference on Computer Aided Verification (CAV)* (2014), Springer, pp. 334–342.

[39] CERNÁ, I., AND PELÁNEK, R. Relating hierarchy of temporal properties to model checking. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science 2003* (2003), B. Rovan and P. Vojtás, Eds., vol. 2747 of *Lecture Notes in Computer Science*, Springer, pp. 318–327.

[40] CHANG, E. Y., MANNA, Z., AND PNUELI, A. Characterization of temporal property classes. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming* (1992), W. Kuich, Ed., vol. 623 of *Lecture Notes in Computer Science*, Springer, pp. 474–486.

[41] CHLEBUS, B. S. Domino-tiling games. *Journal of Computer and System Sciences 32*, 3 (1986), 374–392.

[42] CHURCH, A. A set of postulates for the foundation of logic. *Annals of mathematics* (1932), 346–366.

[43] CHURCH, A. Logic, arithmetic, and automata. In *Proceedings of the international congress of mathematicians* (1962), vol. 1962, pp. 23–35.

[44] Church, A. Application of recursive arithmetic to the problem of circuit synthesis. *Journal of Symbolic Logic 28*, 4 (1963).

[45] Cimatti, A., Geatti, L., Gigante, N., Montanari, A., and Tonetta, S. Fairness, assumptions, and guarantees for extended bounded response LTL+P synthesis.

[46] Cimatti, A., Geatti, L., Gigante, N., Montanari, A., and Tonetta, S. Reactive synthesis from extended bounded response LTL specifications. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020* (2020), IEEE, pp. 83–92.

[47] Cimatti, A., Geatti, L., Gigante, N., Montanari, A., and Tonetta, S. Expressiveness of Extended Bounded Response LTL. *arXiv preprint arXiv:2109.08319* (2021).

[48] Cimatti, A., Geatti, L., Griggio, A., Kimberly, G., and Tonetta, S. Safe decomposition of startup requirements: Verification and synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I* (2020), A. Biere and D. Parker, Eds., vol. 12078 of *Lecture Notes in Computer Science*, Springer, pp. 155–172.

[49] Cimatti, A., Griggio, A., Magnago, E., Roveri, M., and Tonetta, S. Extending nuXmv with Timed Transition Systems and Timed Temporal Properties. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I* (2019), pp. 376–386.

[50] Cimatti, A., Griggio, A., Magnago, E., Roveri, M., and Tonetta, S. Smt-based satisfiability of first-order ltl with event freezing functions and metric operators.

[51] Cimatti, A., Griggio, A., Mover, S., and Tonetta, S. Parameter synthesis with ic3. In *2013 Formal Methods in Computer-Aided Design* (2013), IEEE, pp. 165–168.

[52] CIMATTI, A., GRIGGIO, A., MOVER, S., AND TONETTA, S. Verifying LTL Properties of Hybrid Systems with K-Liveness. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings* (2014), pp. 424–440.

[53] CIMATTI, A., GRIGGIO, A., MOVER, S., AND TONETTA, S. Infinite-state invariant checking with IC3 and predicate abstraction. *Formal Methods in System Design 49*, 3 (2016), 190–218.

[54] CIMATTI, A., GRIGGIO, A., SCHAAFSMA, B. J., AND SEBASTIANI, R. The MathSAT5 SMT solver. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2013), N. Piterman and S. A. Smolka, Eds., vol. 7795 of *Lecture Notes in Computer Science*, Springer, pp. 93–107.

[55] CIMATTI, A., GRIGGIO, A., SCHAAFSMA, B. J., AND SEBASTIANI, R. The mathsat5 smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2013), Springer, pp. 93–107.

[56] CIMATTI, A., ROVERI, M., AND SHERIDAN, D. Bounded Verification of Past LTL. In *Formal Methods in Computer-Aided Design* (2004), LNCS, Springer, pp. 245–259.

[57] CLAESSEN, K., AND SÖRENSSON, N. A liveness checking algorithm that counts. In *2012 Formal Methods in Computer-Aided Design (FMCAD)* (2012), IEEE, pp. 52–59.

[58] CLARKE, E. M., AND EMERSON, E. A. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs* (1981), Springer, pp. 52–71.

[59] CLARKE, E. M., EMERSON, E. A., AND SIFAKIS, J. Model checking: algorithmic verification and debugging. *Communications of the ACM 52*, 11 (2009), 74–84.

[60] CLARKE, E. M., HENZINGER, T. A., VEITH, H., AND BLOEM, R. *Handbook of model checking*, vol. 10. Springer, 2018.

[61] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (1971), pp. 151–158.

[62] D'Agostino, M., Gabbay, D. M., Hähnle, R., and Posegga, J. *Handbook of tableau methods*. Springer Science & Business Media, 2013.

[63] Daniele, M., Giunchiglia, F., and Vardi, M. Y. Improved automata generation for linear temporal logic. In *International Conference on Computer Aided Verification* (1999), Springer, pp. 249–260.

[64] Davis, M. *The universal computer: The road from Leibniz to Turing*. AK Peters/CRC Press, 2018.

[65] Davis, M., Logemann, G., and Loveland, D. A machine program for theorem-proving. *Communications of the ACM 5*, 7 (1962), 394–397.

[66] Davis, M., and Putnam, H. A computing procedure for quantification theory. *Journal of the ACM (JACM) 7*, 3 (1960), 201–215.

[67] De Alfaro, L., and Henzinger, T. A. Interface automata. In *ACM SIGSOFT Software Engineering Notes* (2001), vol. 26, ACM, pp. 109–120.

[68] De Alfaro, L., Henzinger, T. A., and Kupferman, O. Concurrent reachability games. *Theoretical Computer Science 386*, 3 (2007), 188–217.

[69] De Alfaro, L., Henzinger, T. A., and Stoelinga, M. Timed interfaces. In *International Workshop on Embedded Software* (2002), Springer, pp. 108–122.

[70] De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F. M., and Montali, M. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *Proceedings of the 12th International Conference on Business Process Management* (2014), S. W. Sadiq, P. Soffer, and H. Völzer, Eds., vol. 8659 of *Lecture Notes in Computer Science*, Springer, pp. 1–17.

[71] DE GIACOMO, G., DI STASIO, A., TABAJARA, L. M., VARDI, M., AND ZHU, S. Finite-trace and generalized-reactivity specifications in temporal synthesis.

[72] DE GIACOMO, G., IOCCHI, L., FAVORITO, M., AND PATRIZI, F. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling* (2019), J. Benton, N. Lipovetzky, E. Onaindia, D. E. Smith, and S. Srivastava, Eds., AAAI Press, pp. 128–136.

[73] DE GIACOMO, G., AND VARDI, M. Y. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence* (2013), F. Rossi, Ed., IJCAI/AAAI, pp. 854–860.

[74] DE GIACOMO, G., AND VARDI, M. Y. Synthesis for LTL and LDL on finite traces. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015), Q. Yang and M. J. Wooldridge, Eds., AAAI Press, pp. 1558–1564.

[75] DE MOURA, L., AND BJØRNER, N. Satisfiability modulo theories: introduction and applications. *Communications of the ACM 54*, 9 (2011), 69–77.

[76] DE MOURA, L. M., AND BJØRNER, N. Z3: an efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2008), C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963 of *Lecture Notes in Computer Science*, Springer, pp. 337–340.

[77] DEMRI, S., GORANKO, V., AND LANGE, M. *Temporal logics in computer science: finite-state systems*, vol. 58. Cambridge University Press, 2016.

[78] DURET-LUTZ, A., LEWKOWICZ, A., FAUCHILLE, A., MICHAUD, T., RENAULT, E., AND XU, L. Spot 2.0—a framework for LTL and ω-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)* (2016), Springer, pp. 122–129.

[79] Duret-Lutz, A., and Poitrenaud, D. Spot: an extensible model checking library using transition-based generalized bu/spl uml/chi automata. In *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings.* (2004), IEEE, pp. 76–83.

[80] Eén, N., and Sörensson, N. An extensible sat-solver. In *Selected Revised Papers of the 6th International Conference on Theory and Applications of Satisfiability Testing* (2003), pp. 502–518.

[81] Ehlers, R. Symbolic bounded synthesis. In *International conference on Computer Aided Verification (CAV)* (2010), Springer, pp. 365–379.

[82] Ehlers, R. Unbeast: Symbolic bounded synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2011), Springer, pp. 272–275.

[83] Ehlers, R. ACTL ∩ LTL synthesis. In *International Conference on Computer Aided Verification (CAV)* (2012), Springer, pp. 39–54.

[84] Ehlers, R., and Raman, V. Slugs: Extensible GR(1) synthesis. In *International Conference on Computer Aided Verification* (2016), Springer, pp. 333–339.

[85] Elgot, C. C. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society 98*, 1 (1961), 21–51.

[86] Emerson, E. A., Jutla, C. S., and Sistla, A. P. On model-checking for fragments of μ-calculus. In *International Conference on Computer Aided Verification* (1993), Springer, pp. 385–396.

[87] Esparza, J., Křetínskỳ, J., Raskin, J.-F., and Sickert, S. From ltl and limit-deterministic büchi automata to deterministic parity automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2017), Springer, pp. 426–442.

[88] FAYMONVILLE, P., FINKBEINER, B., RABE, M. N., AND TENTRUP, L. Encodings of bounded synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (2017), Springer, pp. 354–370.

[89] FAYMONVILLE, P., FINKBEINER, B., AND TENTRUP, L. Bosy: An experimentation framework for bounded synthesis. In *International Conference on Computer Aided Verification (CAV)* (2017), Springer, pp. 325–332.

[90] FILIOT, E., JIN, N., AND RASKIN, J.-F. An antichain algorithm for LTL realizability. In *International Conference on Computer Aided Verification (CAV)* (2009), Springer, pp. 263–277.

[91] FINKBEINER, B., HAHN, C., LUKERT, P., STENGER, M., AND TENTRUP, L. Synthesizing reactive systems from hyperproperties. In *International Conference on Computer Aided Verification (CAV)* (2018), Springer, pp. 289–306.

[92] FINKBEINER, B., AND SCHEWE, S. Bounded synthesis. *International Journal on Software Tools for Technology Transfer 15*, 5-6 (2013), 519–539.

[93] FIONDA, V., AND GRECO, G. LTL on finite and process traces: Complexity results and a practical reasoner. *J. Artif. Intell. Res. 63* (2018), 557–623.

[94] FREGE, G. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. *From Frege to Gödel: A source book in mathematical logic 1931* (1879), 1–82.

[95] FREGE, G. *Grundgesetze der arithmetik*, vol. 1. Georg Olms Verlag, 1998.

[96] FRIEDMANN, O., AND LANGE, M. Solving parity games in practice. In *International Symposium on Automated Technology for Verification and Analysis* (2009), Springer, pp. 182–196.

[97] GABBAY, D. M., PNUELI, A., SHELAH, S., AND STAVI, J. On the temporal analysis of fairness. In *Conference Record*

of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980 (1980), P. W. Abrahams, R. J. Lipton, and S. R. Bourne, Eds., ACM Press, pp. 163–173.

[98] Gastin, P., and Oddoux, D. Fast ltl to büchi automata translation. In *International Conference on Computer Aided Verification* (2001), Springer, pp. 53–65.

[99] Geatti, L., Gigante, N., and Montanari, A. A sat-based encoding of the one-pass and tree-shaped tableau system for LTL. In *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings* (2019), S. Cerrito and A. Popescu, Eds., vol. 11714 of *Lecture Notes in Computer Science*, Springer, pp. 3–20.

[100] Geatti, L., Gigante, N., Montanari, A., and Reynolds, M. One-pass and tree-shaped tableau systems for tptl and tptlb+ past. *Information and Computation 278* (2021), 104599.

[101] Geatti, L., Gigante, N., Montanari, A., and Reynolds, M. One-pass and tree-shaped tableau systems for TPTL and TPTL_{b}+Past. *Inf. Comput. 278* (2021), 104599.

[102] Geatti, L., Gigante, N., Montanari, A., and Venturato, G. Past matters: Supporting LTL+Past in the BLACK satisfiability checker. In *Proceedings of the 28th International Symposium on Temporal Representation and Reasoning* (2021).

[103] Gerth, R., Kuiper, R., Peled, D., and Penczek, W. A partial order approach to branching time logic model checking. In *Proceedings Third Israel Symposium on the Theory of Computing and Systems* (1995), IEEE, pp. 130–139.

[104] Giacomo, G. D., Masellis, R. D., and Montali, M. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (2014), C. E. Brodley and P. Stone, Eds., AAAI Press, pp. 1027–1033.

[105] GIANTAMIDIS, G., BASAGIANNIS, S., AND TRIPAKIS, S. Efficient translation of safety ltl to dfa using symbolic automata learning and inductive inference. In *International Conference on Computer Safety, Reliability, and Security* (2020), Springer, pp. 115–129.

[106] GIGANTE, N., MONTANARI, A., AND REYNOLDS, M. A one-pass tree-shaped tableau for ltl+ past. In *LPAR* (2017), pp. 456–473.

[107] GIGANTE, N., MONTANARI, A., AND REYNOLDS, M. A one-pass tree-shaped tableau for LTL+Past. In *Proc. of 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning* (2017), vol. 46 of *EPiC Series in Computing*, pp. 456–473.

[108] GÖDEL, K. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik 38*, 1 (1931), 173–198.

[109] GOTTLOB, G. Computer science as the continuation of logic by other means. *Keynote Address, European Computer Science Summit* (2009).

[110] GÖDEL, K. *Über die Vollständigkeit des Logikkalküls*. 1929.

[111] HALPERN, J. Y., HARPER, R., IMMERMAN, N., KOLAITIS, P. G., VARDI, M. Y., AND VIANU, V. On the unusual effectiveness of logic in computer science. *Bulletin of Symbolic Logic 7*, 2 (2001), 213–236.

[112] HELJANKO, K., JUNTTILA, T., AND LATVALA, T. Incremental and complete bounded model checking for full pltl. In *International Conference on Computer Aided Verification* (2005), Springer, pp. 98–111.

[113] HENRIKSEN, J. G., JENSEN, J., JØRGENSEN, M., KLARLUND, N., PAIGE, R., RAUHE, T., AND SANDHOLM, A. Mona: Monadic second-order logic in practice. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems* (1995), Springer, pp. 89–110.

[114] HENZINGER, T. A. *The temporal specification and verification of real-time systems*. PhD thesis, stanford university, 1991.

[115] Hopcroft, J. E., Motwani, R., and Ullman, J. D. Introduction to automata theory, languages, and computation. *Acm Sigact News 32*, 1 (2001), 60–65.

[116] Jacobs, S., and Bloem, R. The 5th reactive synthesis competition (SYNTCOMP 2018).

[117] Jacobs, S., Bloem, R., Brenguier, R., Ehlers, R., Hell, T., Könighofer, R., Pérez, G. A., Raskin, J.-F., Ryzhyk, L., Sankur, O., et al. The first reactive synthesis competition (syntcomp 2014). *International journal on software tools for technology transfer 19*, 3 (2017), 367–390.

[118] Jobstmann, B., and Bloem, R. Optimizations for ltl synthesis. In *2006 Formal Methods in Computer Aided Design* (2006), IEEE, pp. 117–124.

[119] Jurdziński, M. Deciding the winner in parity games is in UP co-UP. *Information Processing Letters 68*, 3 (1998), 119–124.

[120] Kamp, J. A. W. Tense logic and the theory of linear order.

[121] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[122] Kesten, Y., Manna, Z., McGuire, H., and Pnueli, A. A decision algorithm for full propositional temporal logic. In *International Conference on Computer Aided Verification* (1993), Springer, pp. 97–109.

[123] Kleene, S. Representation of events in nerve nets and finite automata. Tech. rep., RAND PROJECT AIR FORCE SANTA MONICA CA, 1951.

[124] Kontchakov, R., Lutz, C., Wolter, F., and Zakharyaschev, M. Temporalising tableaux. *Stud Logica 76*, 1 (2004), 91–134.

[125] Koymans, R. Specifying real-time properties with metric temporal logic. *Real-time systems 2*, 4 (1990), 255–299.

[126] Kretínský, J., Meggendorfer, T., and Sickert, S. Owl: A library for ω-words, automata, and LTL. In *Automated*

Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings (2018), S. K. Lahiri and C. Wang, Eds., vol. 11138 of *Lecture Notes in Computer Science*, Springer, pp. 543–550.

[127] KUPFERMAN, O., AND VARDI, M. Y. Model checking of safety properties. *Formal Methods in System Design 19*, 3 (2001), 291–314.

[128] KUPFERMAN, O., AND VARDI, M. Y. Safraless decision procedures. In *46th Annual Symposium on Foundations of Computer Science (FOCS)* (2005), IEEE, pp. 531–540.

[129] KUPFERMAN, O., VARDI, M. Y., AND WOLPER, P. An automata-theoretic approach to branching-time model checking. *Journal of the ACM (JACM) 47*, 2 (2000), 312–360.

[130] LAWRENCE, B., WIEGERS, K., AND EBERT, C. The top risk of requirements engineering. *IEEE Software 18*, 6 (2001), 62–63.

[131] LEUCKER, M., AND SCHALLHART, C. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming 78*, 5 (2009), 293–303.

[132] LI, J., PU, G., ZHANG, Y., VARDI, M. Y., AND ROZIER, K. Y. Sat-based explicit ltlf satisfiability checking. *Artif. Intell. 289* (2020), 103369.

[133] LI, J., YAO, Y., PU, G., ZHANG, L., AND HE, J. Aalta: an LTL satisfiability checker over infinite/finite traces. In *Proc. of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2014), S. Cheung, A. Orso, and M. D. Storey, Eds., ACM, pp. 731–734.

[134] LI, J., ZHU, S., PU, G., AND VARDI, M. Y. Sat-based explicit ltl reasoning. In *Haifa Verification Conference* (2015), Springer, pp. 209–224.

[135] LICHTENSTEIN, O., AND PNEULI, A. Propositional temporal logics: Decidability and completeness. *Logic Journal of IGPL 8*, 1 (2000), 55–85.

[136] Lichtenstein, O., Pnueli, A., and Zuck, L. The glory of the past. In *Workshop on Logic of Programs* (1985), Springer, pp. 196–218.

[137] Luttenberger, M., Meyer, P. J., and Sickert, S. Practical synthesis of reactive systems from ltl specifications via parity games. *Acta Informatica 57*, 1 (2020), 3–36.

[138] Maler, O., Nickovic, D., and Pnueli, A. Real time temporal logic: Past, present, future. In *International Conference on Formal Modeling and Analysis of Timed Systems* (2005), Springer, pp. 2–16.

[139] Maler, O., Nickovic, D., and Pnueli, A. On synthesizing controllers from bounded-response properties. In *International Conference on Computer Aided Verification* (2007), Springer, pp. 95–107.

[140] Manna, Z., and Pnueli, A. A hierarchy of temporal properties (invited paper, 1989). In *Proceedings of the 9th annual ACM symposium on Principles of distributed computing* (1990), pp. 377–410.

[141] Manna, Z., and Pnueli, A. *Temporal verification of reactive systems - safety*. Springer, 1995.

[142] Manna, Z., and Wolper, P. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS) 6*, 1 (1984), 68–93.

[143] Markey, N. Temporal logic with past is exponentially more succinct.

[144] Marques-Silva, J. P., and Sakallah, K. A. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers 48*, 5 (1999), 506–521.

[145] McCabe-Dansted, J. C., and Reynolds, M. A parallel linear temporal logic tableau. *arXiv preprint arXiv:1709.02101* (2017).

[146] McMillan, K. L. Symbolic model checking. In *Symbolic Model Checking*. Springer, 1993, pp. 25–60.

[147] MCMILLAN, K. L. The smv language. *Cadence Berkeley Labs* (1999), 1–49.

[148] MCNAUGHTON, R. Testing and generating infinite sequences by a finite automaton. *Information and control 9*, 5 (1966), 521–530.

[149] MCNAUGHTON, R., AND PAPERT, S. A. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.

[150] MEYER, P. J., SICKERT, S., AND LUTTENBERGER, M. Strix: Explicit reactive synthesis strikes back! In *International Conference on Computer Aided Verification* (2018), Springer, pp. 578–586.

[151] MIYANO, S., AND HAYASHI, T. Alternating finite automata on ω-words. *Theoretical Computer Science 32*, 3 (1984), 321–330.

[152] MONTANARI, A., PUPPIS, G., SALA, P., AND SCIAVICCO, G. Decidability of the interval temporal logic ABB over the natural numbers. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France* (2010), J. Marion and T. Schwentick, Eds., vol. 5 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 597–608.

[153] MORGENSTERN, A., AND SCHNEIDER, K. A LTL fragment for GR(1)-synthesis. In *Proceedings International Workshop on Interactions, Games and Protocols, iWIGP 2011, Saarbrücken, Germany, 27th March 2011* (2011), J. Reich and B. Finkbeiner, Eds., vol. 50 of *EPTCS*, pp. 33–45.

[154] NIEMELÄ, I. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence 53*, 1-4 (2008), 313–329.

[155] PELED, D. Ten years of partial order reduction. In *International Conference on Computer Aided Verification* (1998), Springer, pp. 17–28.

[156] Pill, I., Semprini, S., Cavada, R., Rovers, M., Bloem, R., and Cimatti, A. Formal analysis of hardware requirements. In *2006 43rd ACM/IEEE Design Automation Conference* (2006), IEEE, pp. 821–826.

[157] Piterman, N. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science 3* (2007).

[158] Piterman, N., Pnueli, A., and Sa'ar, Y. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation* (2006), Springer, pp. 364–380.

[159] Pnueli, A. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)* (1977), IEEE, pp. 46–57.

[160] Pnueli, A., and Rosner, R. On the synthesis of an asynchronous reactive module. In *International Colloquium on Automata, Languages, and Programming (ICALP)* (1989), Springer, pp. 652–671.

[161] Queille, J.-P., and Sifakis, J. Specification and verification of concurrent systems in cesar. In *International Symposium on programming* (1982), Springer, pp. 337–351.

[162] Rabinovich, A. A proof of kamp's theorem. *Log. Methods Comput. Sci. 10*, 1 (2014).

[163] Reynolds, M. A New Rule for LTL Tableaux. 287–301.

[164] Rosner, R. *Modular synthesis of reactive systems*. PhD thesis, PhD thesis, Weizmann Institute of Science, 1992.

[165] Rozier, K. Y., and Vardi, M. Y. Ltl satisfiability checking. In *International SPIN Workshop on Model Checking of Software* (2007), Springer, pp. 149–167.

[166] Rozier, K. Y., and Vardi, M. Y. Ltl satisfiability checking. *International journal on software tools for technology transfer 12*, 2 (2010), 123–137.

[167] Safra, S. On the complexity of omega-automata. In *Proc. 29th IEEE Symp. Found. of Comp. Sci* (1988), pp. 319–327.

[168] SAVITCH, W. J. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences 4*, 2 (1970), 177–192.

[169] SCHEWE, S. Solving parity games in big steps. In *International Conference on Foundations of Software Technology and Theoretical Computer Science* (2007), Springer, pp. 449–460.

[170] SCHIERING, I., AND THOMAS, W. Counter-free automata, first-order logic, and star-free expressions extended by prefix oracles. *Developments in Language Theory, II (Magdeburg, 1995), Worl Sci. Publishing, River Edge, NJ* (1996), 166–175.

[171] SCHUPPAN, V., AND DARMAWAN, L. Evaluating LTL Satisfiability Solvers. In *Proc. of the 9th International Symposium on Automated Technology for Verification and Analysis* (2011), pp. 397–413.

[172] SCHWENDIMANN, S. A new one-pass tableau calculus for PLTL. In *Proc. of the 7th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods* (1998), vol. 1397 of *LNCS*, Springer, pp. 277–292.

[173] SEBASTIANI, R., AND TONETTA, S. "More Deterministic" vs. "Smaller" Büchi Automata for Efficient LTL Model Checking. In *CHARME* (2003), D. Geist and E. Tronci, Eds., vol. 2860 of *Lecture Notes in Computer Science*, Springer, pp. 126–140.

[174] SHERMAN, R., PNUELI, A., AND HAREL, D. Is the interesting part of process logic uninteresting? A translation from PL to PDL. *SIAM J. Comput. 13*, 4 (1984), 825–839.

[175] SICKERT, S., ESPARZA, J., JAAX, S., AND KŘETÍNSKÝ, J. Limit-deterministic büchi automata for linear temporal logic. In *International Conference on Computer Aided Verification* (2016), Springer, pp. 312–332.

[176] SILVA, J. P. M., AND SAKALLAH, K. A. Grasp—a new search algorithm for satisfiability. In *The Best of ICCAD*. Springer, 2003, pp. 73–89.

[177] SISTLA, A. P. On characterization of safety and liveness properties in temporal logic. In *Proceedings of the fourth an-*

nual ACM symposium on Principles of distributed computing (1985), pp. 39–48.

[178] Sistla, A. P. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing 6*, 5 (1994), 495–511.

[179] Sistla, A. P., and Clarke, E. M. The complexity of propositional linear temporal logics. *Journal of the ACM (JACM) 32*, 3 (1985), 733–749.

[180] Smullyan, R. M. *First-order logic*. Courier Corporation, 1995.

[181] Somenzi, F., and Bloem, R. Efficient büchi automata from ltl formulae. In *International Conference on Computer Aided Verification* (2000), Springer, pp. 248–263.

[182] Soos, M., Nohl, K., and Castelluccia, C. Extending SAT solvers to cryptographic problems. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing* (2009), O. Kullmann, Ed., vol. 5584 of *Lecture Notes in Computer Science*, Springer, pp. 244–257.

[183] Stigge, M., Ekberg, P., Guan, N., and Yi, W. The digraph real-time task model. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium* (2011), IEEE, pp. 71–80.

[184] Stigge, M., and Yi, W. Combinatorial abstraction refinement for feasibility analysis of static priorities. *Real-Time Systems 51*, 6 (2015), 639–674.

[185] Stockmeyer, L. J. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

[186] Strejcek, J. *Linear temporal logic: Expressiveness and model checking*. PhD thesis, Faculty of Informatics, Masaryk University in Brno, 2004.

[187] Tahrat, S., Braun, G., Artale, A., and Ozaki, A. Abstracting temporal aboxes in tdl-lite. In *Proceedings of the 34th International Workshop on Description Logics* (2021).

[188] TAURIAINEN, H., AND HELJANKO, K. Testing LTL formula translation into Büchi automata. *International Journal on Software Tools for Technology Transfer 4*, 1 (Oct. 2002), 57–70.

[189] THIRIOUX, X. Simple and efficient translation from ltl formulas to büchi automata. *Electronic Notes in Theoretical Computer Science 66*, 2 (2002), 145–159.

[190] THOMAS, W. Star-free regular sets of ω-sequences. *Information and Control 42*, 2 (1979), 148–156.

[191] THOMAS, W. A combinatorial approach to the theory of ω-automata. *Information and Control 48*, 3 (1981), 261–283.

[192] THOMAS, W. Safety-and liveness-properties in propositional temporal logic: characterizations and decidability. *Banach Center Publications 1*, 21 (1988), 403–417.

[193] THOMAS, W. Automata on infinite objects. In *Formal Models and Semantics*. Elsevier, 1990, pp. 133–191.

[194] TURING, A. M. On computable numbers, with an application to the entscheidungsproblem. a correction. *Proceedings of the London Mathematical Society 2*, 1 (1938), 544–546.

[195] VAN EMDE BOAS, P., ET AL. The convenience of tilings. *Lecture Notes in Pure and Applied Mathematics* (1997), 331–363.

[196] VARDI, M. Y. An automata-theoretic approach to linear temporal logic. In *Logics for concurrency*. Springer, 1996, pp. 238–266.

[197] VARDI, M. Y., AND WOLPER, P. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science* (1986), IEEE Computer Society, pp. 322–331.

[198] VARDI, M. Y., AND WOLPER, P. Reasoning about infinite computations. *Information and computation 115*, 1 (1994), 1–37.

[199] WOLPER, P. Temporal logic can be more expressive. *Information and control 56*, 1-2 (1983), 72–99.

[200] Wolper, P. The tableau method for temporal logic: An overview. *Logique et Analyse* (1985), 119–136.

[201] Zhu, S., Tabajara, L. M., Li, J., Pu, G., and Vardi, M. Y. A Symbolic Approach to Safety LTL Synthesis. In *Proceedings of the 13th International Haifa Verification Conference* (2017), O. Strichman and R. Tzoref-Brill, Eds., vol. 10629 of *Lecture Notes in Computer Science*, Springer, pp. 147–162.

[202] Zuck, L. Past temporal logic. *Weizmann Institute of Science 67* (1986).

Part IV

Appendix

APPENDIX

A

PROOFS

A.1 Proofs of Chapter 7

Proposition 15 (Soundness of pastification)**.** *Let φ be a* $\mathsf{LTL}{+}\mathsf{P}_{\mathsf{BF}}$ *formula. For all state sequences $\sigma \in (2^{\Sigma})^{\omega}$, all $i \in \mathbb{N}$, and all $d \geq D(\phi)$, it holds that:*

$$\sigma, i \models \varphi \;\Leftrightarrow\; \sigma, i \models \mathsf{X}^{d}\Pi(\varphi, d)$$

Proof. The proof goes by structural induction over φ. As the base case, consider a $\mathsf{LTL}{+}\mathsf{P}_{\mathsf{P}}$ formula ψ, and since $D(\psi) = 0$, consider any $d \geq 0$. It holds that $\sigma, i \models \psi$ if and only if $\sigma, i \models \mathsf{X}^{d}\mathsf{Y}^{d}\psi$, hence $\sigma, i + d \models \mathsf{Y}^{d}\psi$, which by definition of $\Pi(\cdot)$ is equivalent to $\sigma, i + d \models \Pi(\psi, d)$. For the inductive case, we consider multiple cases. The case for the *negation* and the *disjunction* operators are straightforward. Consider now the case $\phi \equiv \mathsf{X}\phi_1$. We prove first the

left-to-right direction. It holds that:

$$
\begin{aligned}
& \sigma, i \models \mathsf{X}\phi_1 \\
\leftrightarrow\ & \sigma, i+1 \models \phi_1 \\
& \qquad\qquad \textit{semantics of } \text{next} \\
\leftrightarrow\ & \forall d_1 \geq D(\phi_1)\ .\ \sigma, i+1+d_1 \models \Pi(\phi_1, d_1) \\
& \qquad\qquad \textit{inductive hypothesis on } \phi_1 \\
\leftrightarrow\ & \forall d \geq D(\phi_1)+1\ .\ \sigma, i+d \models \Pi(\phi_1, d-1) \\
& \qquad\qquad \textit{with } d = d_1 + 1 \\
\leftrightarrow\ & \forall d \geq D(\phi)\ .\ \sigma, i+d \models \Pi(\phi_1, d-1) \\
& \qquad\qquad \textit{since } D(\phi) = D(\phi_1) + 1 \\
\leftrightarrow\ & \forall d \geq D(\phi)\ .\ \sigma, i+d \models \Pi(\mathsf{X}\phi_1, d) \\
& \qquad\qquad \textit{by definition of } \Pi(\cdot) \\
\leftrightarrow\ & \forall d \geq D(\phi)\ .\ \sigma, i \models \mathsf{X}^d \Pi(\phi, d)
\end{aligned}
$$

Consider now the case $\phi \equiv \phi_1\ \mathsf{U}^{[a,b]}\ \phi_2$. The following equivalences hold:

$$
\begin{aligned}
& \sigma, i \models \phi_1\ \mathsf{U}^{[a,b]}\ \phi_2 \\
\leftrightarrow\ & \exists j_2\ .\ \big((a \leq j_2 \leq b) \wedge \sigma, i+j_2 \models \phi_2 \wedge \\
& \qquad \forall j_1\ .\ ((0 \leq j_1 < j_2) \rightarrow \sigma, i+j_1 \models \phi_1)\big) \\
& \qquad\qquad \textit{semantics of } \text{bounded until} \\
\leftrightarrow\ & \forall d_1 \geq D(\phi_1)\ .\ \forall d_2 \geq D(\phi_2)\ .\ \Big(\\
& \qquad \exists j_2\ .\ \big((a \leq j_2 \leq b) \wedge \sigma, i+j_2+d_2 \models \Pi(\phi_2, d_2) \wedge \\
& \qquad\qquad \forall j_1\ .\ ((0 \leq j_1 < j_2) \rightarrow \sigma, i+j_1+d_1 \models \Pi(\phi_1, d_1))\big)\Big) \\
& \qquad\qquad \textit{by the inductive hypothesis}
\end{aligned}
$$

Since $D(\phi) = b + \max\{D(\phi_1), D(\phi_2)\}$, it holds that $D(\phi_1) \leq D(\phi) - b$ and $D(\phi_2) \leq D(\phi) - b$. Therefore, for any *first-order* formula $\phi(d_1, d_2)$ where d_1 and d_2 are free variables, it holds that if $\forall d_1 \geq D(\phi_1)\ .\ \forall d_2 \geq D(\phi_2)\ .\ \phi(d_1, d_2)$ then $\forall d \geq D(\phi)\ .\ \phi[d_1 \mapsto (d -$

$b), d_2 \mapsto (d-b)]$. And thus:

$$\Rightarrow \forall d \geq D(\phi) \,.\, \Big(\exists j_2 \,.\, \big((a \leq j_2 \leq b) \wedge \sigma, i + j_2 + d - b \models \Pi(\phi_2, d-b) \wedge \forall j_1 \,.\, ((0 \leq j_1 < j_2) \rightarrow \sigma, i + j_1 + d - b \models \Pi(\phi_1, d-b))\big)\Big)$$

since $D(\phi) = b + \max D(\phi_1), D(\phi_2)$

$$\Leftrightarrow \forall d \geq D(\phi) \,.\, \Big(\exists k_2 \,.\, \big((a \leq b - k_2 \leq b) \wedge \sigma, i + d - k_2 \models \Pi(\phi_2, d-b) \wedge \forall j_1 \,.\, ((0 \leq b - k_1 < b - k_2) \rightarrow \sigma, i + d - k_1 \models \Pi(\phi_1, d-b))\big)\Big)$$

with $k_2 = b - j_2$ *and* $k_1 = b - j_1$

$$\Leftrightarrow \forall d \geq D(\phi) \,.\, \Big(\exists k_2 \,.\, \big((0 \leq k_2 \leq b - a) \wedge \sigma, i + d - k_2 \models \Pi(\phi_2, d-b) \wedge \forall j_1 \,.\, ((0 \leq k_1 \leq b - k_2 - 1) \rightarrow \sigma, i + d - k_1 - 1 \models \Pi(\phi_1, d-b))\big)\Big)$$

simple arithmetic

$$\Leftrightarrow \forall d \geq D(\phi) \,.\, \big(\sigma, i + d \models \bigvee_{k_2=0}^{b-a} \mathsf{Y}^{k_2}(\Pi(\phi_2, d-b) \wedge \mathsf{H}^{[0, b-k_2-1]} \mathsf{Y} \Pi(\phi_1, d-b))\big)$$

definition of yesterday *and* bounded historically *operators*

$$\Leftrightarrow \forall d \geq D(\phi) \,.\, \big(\sigma, i \models \mathsf{X}^d \Pi(\phi_1 \,\mathsf{U}^{[a,b]}\, \phi_2, d)\big)$$

definition of $\Pi(\cdot)$

We now prove the right-to-left direction. It holds that:

$$\forall d \geq D(\phi)\ .\ \big(\sigma, i \models \mathsf{X}^d(\bigvee_{t=0}^{b-a} \mathsf{Y}^t(\Pi(\phi_2, d-t) \wedge \mathsf{H}^{[0,b-t-1]}\mathsf{Y}\Pi(\phi_1, d-b)))\big)$$

$$\Leftrightarrow \forall d \geq D(\phi)\ .\ \big($$
$$\exists k_2\ .\ ((0 \leq k_2 \leq b-a) \wedge \pi, i+d-k_2 \models \Pi(\phi_2, d-b) \wedge$$
$$\forall k_1\ .\ ((0 \leq k_1 \leq b-k_2-1) \wedge$$
$$\pi, i+d-k_2-k_1-1 \models \Pi(\phi_1, d-b)))\big)$$

semantics of yesterday *and* bounded until

$$\Leftrightarrow \forall d \geq D(\phi)\ .\ \big($$
$$\exists k_2\ .\ ((0 \leq k_2 \leq b-a) \wedge \pi, i-k_2 \models \mathsf{X}^d\Pi(\phi_2, d-b) \wedge$$
$$\forall k_1\ .\ ((0 \leq k_1 \leq b-k_2-1) \wedge$$
$$\pi, i-k_2-k_1-1 \models \mathsf{X}^d\Pi(\phi_1, d-b)))\big)$$

semantics of next

$$\Leftrightarrow \forall d_1 \geq D(\phi)-b\ .\ \forall d_2 \geq D(\phi)-b\ .\ \big($$
$$\exists k_2\ .\ ((0 \leq k_2 \leq b-a) \wedge \pi, i-k_2 \models \mathsf{X}^{d_2+b}\Pi(\phi_2, d_2) \wedge$$
$$\forall k_1\ .\ ((0 \leq k_1 \leq b-k_2-1) \wedge$$
$$\pi, i-k_2-k_1-1 \models \mathsf{X}^{d_1+b}\Pi(\phi_1, d_1)))\big)$$

with $d_1 = d-b$ *and* $d_2 = d-b$

$$\Leftrightarrow \forall d_1 \geq D(\phi)-b\ .\ \forall d_2 \geq D(\phi)-b\ .\ \big($$
$$\exists k_2\ .\ ((0 \leq k_2 \leq b-a) \wedge \pi, i-k_2+b \models \mathsf{X}^{d_2}\Pi(\phi_2, d_2) \wedge$$
$$\forall k_1\ .\ ((0 \leq k_1 \leq b-k_2-1) \wedge$$
$$\pi, i-k_2-k_1-1+b \models \mathsf{X}^{d_1}\Pi(\phi_1, d_1)))\big)$$

semantics of next

Since $D(\phi_1) \leq D(\phi) - b$ and $D(\phi_2) \leq D(\phi) - b$, the inductive hypothesis applies in particular for all $d_1 \geq D(\phi_1) - b$ and forall

$d_2 \geq D(\phi_2) - b$, and thus we have:

$$\Rightarrow \exists k_2 \,.\, ((0 \leq k_2 \leq b - a) \wedge \pi, i - k_2 + b \models \phi_2 \wedge \\ \forall k_1 \,.\, ((0 \leq k_1 \leq b - k_2 - 1) \wedge \pi, i - k_2 - k_1 - 1 + b \models \phi_1))$$

by inductive hypothesis

$$\Leftrightarrow \exists j_2 \,.\, ((0 \leq b - j_2 \leq b - a) \wedge \pi, i + j_2 \models \phi_2 \wedge \\ \forall j_1 \,.\, ((0 \leq b - j_1 \leq b + j_2 - b - 1) \wedge \pi, i + j_2 - b + j_1 - 1 \models \phi_1))$$

with $j_2 = b - k_2$ *and* $k_1 = b - j_1$

$$\Leftrightarrow \exists j_2 \,.\, ((a \leq j_2 \leq b) \wedge \pi, i + j_2 \models \phi_2 \wedge \\ \forall j_1 \,.\, ((b + 1 - j_2 \leq j_1 \leq b) \wedge \pi, i + j_2 - b + j_1 - 1 \models \phi_1))$$

by simple arithmetics

$$\Leftrightarrow \exists j_2 \,.\, ((a \leq j_2 \leq b) \wedge \pi, i + j_2 \models \phi_2 \wedge \\ \forall l \,.\, ((b + 1 - j_2 \leq l - j_2 + 1 + b \leq b) \wedge \pi, i + l \models \phi_1))$$

with $l = j_1 + j_2 - b - 1$

$$\Leftrightarrow \exists j_2 \,.\, ((a \leq j_2 \leq b) \wedge \pi, i + j_2 \models \phi_2 \wedge \\ \forall l \,.\, ((0 \leq l < j_2) \wedge \pi, i + l \models \phi_1))$$

by simple arithmetics

$$\Leftrightarrow \sigma, i \models \phi_1 \,\mathsf{U}^{[a,b]}\, \phi_2$$

by the semantics of bounded until

This concludes the proof. □

Proposition 16 (Size of pastification)**.** *Let* ϕ *be a* $\mathsf{LTL}{+}\mathsf{P}_{\mathsf{BF}}$ *formula. Then,* $\mathsf{pastify}(\phi)$ *is a formula of size* $\mathcal{O}(n)$*, where* $n = |\phi|$*.*

Proof. We first give a bound for the $\Pi(\cdot)$ operator. It holds that:

- $|\Pi(\psi, d)| = d + |\psi|$ for each $\psi \in \mathsf{LTL}{+}\mathsf{P}_{\mathsf{P}}$;
- $|\Pi(\neg\phi, d)| = |\Pi(\phi, d)| + 1$;
- $|\Pi(\phi_1 \vee \phi_2, d)| = |\Pi(\phi_1, d)| + |\Pi(\phi_2, d)| + 1$;
- $|\Pi(\mathsf{X}\phi_1, d)| = |\Pi(\phi_1, d - 1)| + 1 \leq |\Pi(\phi_1, d)| + 1$;

Recall that, since bounded operators are shortcuts, it suffices to consider the cases of atomic propositions, Boolean operators and the *next* temporal operator.

The case for *disjunctions* clearly dominates all the others. Suppose without loss of generality that $|\phi_1| = |\phi_2| = \frac{|\phi|-1}{2}$. The recurrence equation $S(n)$ describing the space required for $|\Pi(\phi, d)|$, with $n = |\phi|$, is the following:

$$S(n) = \begin{cases} \mathcal{O}(d) & \text{if } n = 1 \\ \mathcal{O}(1) + 2 \cdot S(\frac{n}{2}) & \text{otherwise} \end{cases}$$

By unrolling the equation for i steps, we have that $S(n) = \mathcal{O}(i) + 2^i \cdot S(\frac{n}{2^i})$. For $i = \log_2 n$, the equation amounts to:

$$\begin{aligned} S(n) &= \mathcal{O}(\log_2 n) + 2^{\log_2 n} \cdot S(\frac{n}{2^{\log_2 n}}) \\ &= \mathcal{O}(\log_2 n) + n \cdot S(1) \\ &= \mathcal{O}(n) \end{aligned}$$

Since $\mathsf{pastify}(\phi)$ is defined as $X^d\Pi(\phi, d))$ where $d = D(\phi)$, and since $D(\phi) \leq n$, it holds that $|\mathsf{pastify}(\phi)| \leq \mathcal{O}(n)$. □

Lemma 23 (Strong equivalence for the rules)**.** *Let ψ, ψ_1, ψ_2 and ψ_3 be $\mathsf{LTL}{+}\mathsf{P_P}$ formulas. For all state sequences σ and for all positions $i \in \mathbb{N}$, it holds that:*

R_1: $\sigma, i \models \mathsf{X}(\psi_1 \wedge \psi_2) \Leftrightarrow \sigma, i \models \mathsf{X}\psi_1 \wedge \mathsf{X}\psi_2$

R_2: $\sigma, i \models \psi \,\mathsf{R}\, (\psi_1 \wedge \psi_2) \Leftrightarrow \sigma, i \models \psi \,\mathsf{R}\, \psi_1 \wedge \psi \,\mathsf{R}\, \psi_2$

R_3: $\sigma, i \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\psi_2) \Leftrightarrow$

$$\sigma, i \models \begin{cases} \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2)) & \textit{if } i > j \\ \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathsf{R}\, \psi_2) & \textit{otherwise} \end{cases}$$

R_4: $\sigma, i \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j(\psi_2 \,\mathsf{R}\, \psi_3)) \Leftrightarrow$

$$\sigma, i \models \begin{cases} \mathsf{X}^i(\psi_1 \,\mathsf{R}\, ((\mathsf{Y}^{i-j}\psi_2) \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_3))) \\ \quad \textit{if } i > j \\ \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathsf{R}\, (\psi_2 \,\mathsf{R}\, \psi_3)) \\ \quad \textit{otherwise} \end{cases}$$

R_5: $\sigma, i \models \mathsf{GX}^i\mathsf{G}\psi \Leftrightarrow \sigma, i \models \mathsf{X}^i\mathsf{G}\psi$

R_6: $\sigma, i \models \mathsf{G}\mathsf{X}^i(\psi_1 \,\mathsf{R}\, \psi_2) \Leftrightarrow \sigma, i \models \mathsf{X}^i\mathsf{G}\psi_2$

R_7: $(\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\mathsf{G}\psi_2) \Leftrightarrow$

$$\sigma, i \models \begin{cases} \mathsf{X}^i\mathsf{G}\mathsf{Y}^{i-j}\psi_2 & \text{if } i > j \\ \mathsf{X}^j\mathsf{G}\psi_2 & \text{otherwise} \end{cases}$$

R_{flat}: $\sigma, 0 \models \mathsf{X}^i(\psi_1\mathsf{R}(\psi_2\mathsf{R}(\ldots(\psi_{n-1}\mathsf{R}\psi_n)\ldots))) \Leftrightarrow \sigma, 0 \models \mathsf{X}^i((\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \ldots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i\top)\ldots))) \,\mathsf{R}\, \psi_n)$ $\forall n \geq 3$

Proof. Before starting the proof, we remark that the claim of this lemma not only asks for proving the *equivalence* between the left- and the right-hand side of the rules, but requires to prove the *strong equivalence* between the two, *i.e.*, that for all the state sequences σ and for all the positions i, σ is a model starting from position i of the left-hand formula iff σ is a model starting from position i of the right-hand formula. Equivalence is a special case of strong equivalence by considering only $i = 0$. In our case, the *necessity* of considering strong equivalence is due to the fact that the left-hand side of the rules (except for R_{flat}, for which we require only the equivalence) can appear as subformulas of the original ϕ on which we apply the normalize algorithm, and thus it can be interpreted potentially on any position i. Since we want to maintain the equivalence between ϕ and normalize(ϕ), we have to make sure that each subformulas is strongly equivalent to the one by which it is replaced during the applications of the rules. The only exception is the R_{flat} rule, which is applied only to top-level conjuncts or disjuncts, and thus we can require for it to maintain only the equivalence.

Initially we prove the first two points (*i.e.*, R_1 and R_2). For the R_1 rule, the following steps hold:

$$\begin{aligned} & \sigma, i \models \mathsf{X}(\psi_1 \wedge \psi_2) \\ \Leftrightarrow\ & \sigma, i+1 \models \psi_1 \wedge \psi_2 \\ \Leftrightarrow\ & \sigma, i+1 \models \psi_1 \wedge \sigma, i+1 \models \psi_2 \\ \Leftrightarrow\ & \sigma, i \models \mathsf{X}\psi_1 \wedge \sigma, i \models \mathsf{X}\psi_2 \\ \Leftrightarrow\ & \sigma, i \models \mathsf{X}\psi_1 \wedge \mathsf{X}\psi_2 \end{aligned}$$

Consider rule R_2. We first prove that $\sigma, s \models \psi \,\mathsf{R}\, (\phi_1 \wedge \phi_2)$ implies $\sigma, s \models \psi \,\mathsf{R}\, \phi_1 \wedge \psi \,\mathsf{R}\, \phi_2$, for all state sequences σ and for all positions s. Let σ be a state sequence and let $s \in \mathbb{N}$ be a position such that $\sigma, s \models \psi \,\mathsf{R}\, (\phi_1 \wedge \phi_2)$. We divide in cases:

1. if $\forall i \geq s.(\sigma, i \models \phi_1 \wedge \phi_2)$, then $\forall i \geq s.\sigma, i \models \phi_1$ and $\forall i \geq s.\sigma, i \models \phi_2$. Thus, $\sigma, s \models \psi \,\mathsf{R}\, \phi_1$ and $\sigma, s \models \psi \,\mathsf{R}\, \phi_2$, that is $\sigma, s \models \psi \,\mathsf{R}\, \phi_1 \wedge \psi \,\mathsf{R}\, \phi_2$.

2. if $\exists i \geq s.(\sigma, i \models \psi \wedge \forall s \leq j \leq i.\sigma, j \models (\phi_1 \wedge \phi_2))$ then

$$\begin{aligned}
&\Leftrightarrow \exists i \geq s.(\sigma, i \models \psi \wedge \forall s \leq j \leq i.(\sigma, j \models \phi_1) \wedge \\
&\qquad\qquad \forall s \leq k \leq i.(\sigma, k \models \phi_2)) \\
&\Rightarrow \exists i \geq s.(\sigma, i \models \psi \wedge \forall s \leq j \leq i.(\sigma, j \models \phi_1)) \wedge \\
&\qquad \exists i \geq s.(\sigma, i \models \psi \wedge \forall 0 \leq j \leq i.(\sigma, j \models \phi_2)) \\
&\Leftrightarrow \sigma, s \models \psi \,\mathsf{R}\, \phi_1 \wedge \psi \,\mathsf{R}\, \phi_2
\end{aligned}$$

We now prove the opposite direction, that is $\sigma, s \models \psi \,\mathsf{R}\, \phi_1 \wedge \psi \,\mathsf{R}\, \phi_2$ implies $\sigma, s \models \psi \,\mathsf{R}\, (\phi_1 \wedge \phi_2)$, for all state sequences σ and for all positions s. Let σ be a state sequence and let $s \in \mathbb{N}$ such that $\sigma, s \models \psi \,\mathsf{R}\, \phi_1 \wedge \psi \,\mathsf{R}\, \phi_2$. We divide again in cases:

1. if $\forall i \geq s.(\sigma, i \models \phi_1) \wedge \forall i \geq s.(\sigma, j \models \phi_2)$, then $\forall i \geq s.(\sigma, i \models \phi_1 \wedge \phi_2)$ and thus $\sigma, s \models \psi \,\mathsf{R}\, (\phi_1 \wedge \phi_2)$.

2. if $\forall i \geq s.(\sigma, i \models \phi_1)$ and $\exists i \geq s.(\sigma, i \models \psi \wedge \forall s \leq j \leq i.\sigma, j \models \phi_2)$, then $\exists i \geq s(\sigma, i \models \psi \wedge \forall s \leq j \leq i.\sigma, j \models (\phi_1 \wedge \phi_2))$, that is $\sigma, s \models \psi \,\mathsf{R}\, (\phi_1 \wedge \phi_2)$.

3. if $\exists i \geq s.(\sigma, i \models \psi \wedge \forall s \leq j \leq i.\sigma, j \models \phi_1)$ and $\forall i \geq s.(\sigma, i \models \phi_2)$, then $\exists i \geq s.(\sigma, i \models \psi \wedge \forall s \leq j \leq i.\sigma, k \models \phi_1 \wedge \phi_2)$, that is $\sigma, s \models \psi \,\mathsf{R}\, (\phi_1 \wedge \phi_2)$.

4. consider the case such that $\exists l \geq s.(\sigma, l \models \psi \wedge \forall s \leq j \leq l.\sigma, j \models \phi_1)$ and $\exists k \geq s.(\sigma, k \models \psi \wedge \forall s \leq j \leq k.\sigma, j \models \phi_2)$. Let $i = \min(l, k)$: then $\sigma, i \models \phi$ and $\forall s \leq j \leq i.(\sigma, j \models \phi_1 \wedge \phi_2)$, that is $\sigma, s \models \psi \,\mathsf{R}\, (\phi_1 \wedge \phi_2)$.

This concludes the proof for the R_2 rule.

Before proving the cases of the remaining rules, we define and prove the following auxiliary *strong equivalences*. For all state sequences σ and for all positions i, it holds that:

$\overline{R_1}$: $\sigma, i \models \psi_1 \,\mathsf{R}\, (\mathsf{X}^i \psi_2) \;\leftrightarrow\; \sigma, i \models \mathsf{X}^i((\mathsf{Y}^i \psi_1) \,\mathsf{R}\, \psi_2)$

$\overline{R_2}$: $\sigma, i \models (\mathsf{X}^i \psi_1) \,\mathsf{R}\, \psi_2 \;\leftrightarrow\; \sigma, i \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^i \psi_2))$

$\overline{R_3}$: $\sigma, i \models \mathsf{Y}^i \mathsf{X}^i \psi \;\leftrightarrow\; \sigma, i \models \psi \wedge \mathsf{Y}^i \top$

$\overline{R_4}$: $\sigma, i \models \mathsf{Y}^i(\psi_1 \,\mathsf{R}\, \psi_2) \;\leftrightarrow\; \sigma, i \models (\mathsf{Y}^i\psi_1)\,\mathsf{R}\,(\mathsf{Y}^i\psi_2)$

$\overline{R_5}$: $\sigma, i \models \mathsf{GG}\psi \;\leftrightarrow\; \sigma, i \models \mathsf{G}\psi$

$\overline{R_6}$: $\sigma, i \models \mathsf{G}(\psi_1 \,\mathsf{R}\, \psi_2) \;\leftrightarrow\; \sigma, i \models \mathsf{G}\psi_2$

$\overline{R_7}$: $\sigma, i \models \psi_1 \,\mathsf{R}\, (\mathsf{G}\psi_2) \;\leftrightarrow\; \sigma, i \models \mathsf{G}\psi_2$

These will help proving the cases for R_3-R_7.

Consider the case for rule $\overline{R_1}$. We first prove that $\sigma, s \models \psi_1 \,\mathsf{R}\, (\mathsf{X}^i\psi_2)$ implies $\sigma, s \models \mathsf{X}^i((\mathsf{Y}^i\psi_1)\,\mathsf{R}\,\psi_2)$, for all state sequences σ and all positions s. Let σ be a state sequence and let $s \in \mathbb{N}$ such that $\sigma, s \models \psi_1 \,\mathsf{R}\, (\mathsf{X}^i\psi_2)$. We divide in cases:

1. if $\forall j \geq s.\sigma, j \models \mathsf{X}^i\psi_2$, then

$$\begin{aligned}
&\Leftrightarrow \forall j \geq s + i.\sigma, j \models \psi_2 \\
&\Rightarrow \sigma, s + i \models (\mathsf{Y}^i\psi_1)\,\mathsf{R}\,\psi_2 \\
&\Leftrightarrow \sigma, s \models \mathsf{X}^i((\mathsf{Y}^i\psi_1)\,\mathsf{R}\,\psi_2)
\end{aligned}$$

2. if $\exists j \geq s.(\sigma, j \models \psi_1 \wedge \forall s \leq k \leq j.\sigma, k \models \mathsf{X}^i\psi_2)$, then $\exists j \geq s.(\sigma, j+i \models \mathsf{Y}^i\psi_1 \wedge \forall s+i \leq k \leq j+i.\sigma, k \models \psi_2)$, which in turn means that $\sigma, s+i \models (\mathsf{Y}^i\psi_1)\mathsf{R}\psi_2$, that is $\sigma, s \models \mathsf{X}^i((\mathsf{Y}^i\psi_1)\mathsf{R}\psi_2)$.

We now prove the opposite direction, that is $\sigma, s \models \mathsf{X}^i((\mathsf{Y}^i\psi_1)\,\mathsf{R}\,\psi_2)$ implies $\sigma, s \models \psi_1 \,\mathsf{R}\, (\mathsf{X}^i\psi_2)$, for all state sequences σ and all positions s. Let σ be a state sequence and let $s \in \mathbb{N}$ such that $\sigma, s \models \mathsf{X}^i((\mathsf{Y}^i\psi_1)\,\mathsf{R}\,\psi_2)$. We divide again in cases:

1. if $\forall j \geq s + i.(\sigma, j \models \psi_2)$, then $\forall j \geq s.(\sigma, j \models \mathsf{X}^i\psi_2)$ and thus $\sigma \models \psi_1 \,\mathsf{R}\, (\mathsf{X}^i\psi_2)$.

2. if $\exists j \geq s + i.(\sigma, j \models \mathsf{Y}^i\psi_1 \wedge \forall s + i \leq k \leq j.\sigma, k \models \psi_2)$, then:

$$\begin{aligned}
&\Leftrightarrow \exists j \geq s + i.(\sigma, j - i \models \mathsf{X}^i\mathsf{Y}^i\psi_1 \wedge \forall s \leq k \leq j - i.\sigma, k \models \mathsf{X}^i\psi_2) \\
&\Leftrightarrow \exists j \geq s + i.(\sigma, j - i \models \psi_1 \wedge \forall s \leq k \leq j - i.\sigma, k \models \mathsf{X}^i\psi_2) \\
&\Leftrightarrow \sigma, s + i \models \mathsf{Y}^i(\psi_1 \,\mathsf{R}\, (\mathsf{X}^i\psi_2)) \\
&\Leftrightarrow \sigma, s \models \psi_1 \,\mathsf{R}\, (\mathsf{X}^i\psi_2)
\end{aligned}$$

This concludes the proof for the rule $\overline{R_1}$. The proof for the $\overline{R_2}$ rule is specular.

Consider the $\overline{R_3}$ case. We first prove that $\sigma, s \models \mathsf{Y}^i\mathsf{X}^i\psi$ implies $\sigma, s \models \psi \wedge \mathsf{Y}^i\top$, for all state sequences σ and all positions s. Let σ be a state sequence such that $\sigma, s \models \mathsf{Y}^i\mathsf{X}^i\psi$ for a given $s \in \mathbb{N}$. We divide in cases:

(i) if $s < i$, then $\sigma, s \not\models \mathsf{Y}^i\mathsf{X}^i\psi$, but this is a contradiction with our hypothesis;

(ii) then it has to be the case that $s \geq i$. It holds that:

$$\begin{aligned} \sigma, s \models \mathsf{Y}^i\mathsf{X}^i\psi &\Leftrightarrow \sigma, s-i \models \mathsf{X}^i\psi \\ &\Leftrightarrow \sigma, s-i+i \models \psi \\ &\Leftrightarrow \sigma, s \models \psi \wedge \mathsf{Y}^i\top \qquad \text{since } s \geq i \end{aligned}$$

We prove the opposite direction, that is $\sigma, s \models \psi \wedge \mathsf{Y}^i\top$ implies $\sigma, s \models \mathsf{Y}^i\mathsf{X}^i\psi$, for all state sequences σ and all positions s. Let σ be a state sequence such that $\sigma, s \models \psi \wedge \mathsf{Y}^i\top$ for a given $s \in \mathbb{N}$. We divide in cases:

(i) if $s < i$, then $\sigma, s \not\models \mathsf{Y}^i\top$, but this is a contradiction with our hypothesis;

(ii) then it has to be the case that $s \geq i$. It holds that:

$$\begin{aligned} \sigma, s \models \psi \wedge \mathsf{Y}^i\top &\Leftrightarrow \sigma, s-i \models \mathsf{X}^i\psi \qquad \text{since } s \geq i \\ &\Leftrightarrow \sigma, s-i+i \models \mathsf{Y}^i\mathsf{X}^i\psi \\ &\Leftrightarrow \sigma, s \models \mathsf{Y}^i\mathsf{X}^i\psi \end{aligned}$$

This concludes the proof for $\overline{R_3}$.

Consider now the $\overline{R_4}$ case. We first prove the left-to-right direction, that is $\sigma, s \models \mathsf{Y}^i(\psi_1 \,\mathsf{R}\, \psi_2)$ implies $\sigma, s \models (\mathsf{Y}^i\psi_1) \,\mathsf{R}\, (\mathsf{Y}^i\psi_2)$, for all state sequences σ and all positions s. Let σ be a state sequence such that $\sigma, s \models \mathsf{Y}^i(\psi_1 \,\mathsf{R}\, \psi_2)$ with $s \geq i$ (obviously, it can't be that $s < i$). It holds that $\sigma, s-i \models \psi_1 R \psi_2$. Now, we divide in cases:

1. if $\forall k \geq s-i.\sigma, k \models \psi_2$, then $\forall k \geq s.\sigma, k \models \mathsf{Y}^i\psi_2$ and thus $\sigma, s \models (\mathsf{Y}^i\psi_1) \,\mathsf{R}\, (\mathsf{Y}^i\psi_2)$.

2. if $\exists k \geq s-i.(\sigma, k \models \psi_2 \wedge \forall s-i \leq l \leq k.\sigma, l \models \psi_1)$, then $\exists k \geq s.(\sigma, k \models \mathsf{Y}^i\psi_2 \wedge \forall s \leq l \leq k.\sigma, l \models \mathsf{Y}^i\psi_1)$, and thus $\sigma, s \models (\mathsf{Y}^i\psi_1) \,\mathsf{R}\, (\mathsf{Y}^i\psi_2)$.

Now we prove the opposite direction. Suppose that $\sigma, s \models (\mathsf{Y}^i\psi_1)\,\mathsf{R}\,(\mathsf{Y}^i\psi_2)$ where $s \geq i$. We divide in cases:

1. if $\forall k \geq s.\sigma, k \models \mathsf{Y}^i\psi_2$, then:

$$\begin{aligned}\forall k \geq s - i.\sigma, k \models \psi_2 &\Leftrightarrow \sigma, s - i \models \psi_1\,\mathsf{R}\,\psi_2\\ &\Leftrightarrow \sigma, s \models \mathsf{Y}^i(\psi_1\,\mathsf{R}\,\psi_2)\end{aligned}$$

2. if $\exists k \geq s.(\sigma, k \models \mathsf{Y}^i\psi_1 \wedge \forall k \leq l \leq k.\sigma, l \models \mathsf{Y}^i\psi_2)$, then:

$$\begin{aligned}\exists k \geq s - i.(\sigma, k \models \psi_1 \wedge \forall s - i \leq l \leq k.\sigma, l \models \psi_2) &\Leftrightarrow \sigma, s - i \models \psi_1\,\mathsf{R}\,\psi_2\\ &\Leftrightarrow \sigma, s \models \mathsf{Y}^i(\psi_1\,\mathsf{R}\,\psi_2)\end{aligned}$$

This concludes the proof for the $\overline{R_4}$ case.

The case for $\overline{R_5}$ is simple, and it consists in the following steps. For all state sequences σ and for all positions s, it holds that:

$$\begin{aligned}\sigma, s \models \mathsf{GG}\psi &\Leftrightarrow \forall i \geq s.\sigma, i \models \mathsf{G}\psi\\ &\Leftrightarrow \forall i \geq s.\forall j \geq i.\sigma, j \models \psi\\ &\Leftrightarrow \forall i \geq s.\sigma, i \models \psi\\ &\Leftrightarrow \sigma, s \models \mathsf{G}\psi\end{aligned}$$

Consider the $\overline{R_6}$ strong equivalence. We first prove the left-to-right direction. Suppose that $\sigma, s \models \mathsf{G}(\psi_1\,\mathsf{R}\,\psi_2)$, for a given state sequence σ and a given position s. It holds that $\forall i \geq s.\sigma, i \models \psi_1\,\mathsf{R}\,\psi_2$. We divide in cases, depending on the semantics of the *release* operator:

1. if $\forall i \geq s.\forall j \geq i.\sigma, j \models \psi_2$. In this case we have that $\forall i \geq s.\sigma, i \models \psi_2$, that is $\sigma, s \models \mathsf{G}\psi_2$.

2. otherwise, $\forall i \geq s.\exists j \geq i.(\sigma, j \models \psi_1 \wedge \forall i \leq k \leq j.\sigma, k \models \psi_2)$. In particular, for $k = i$, we have that $\forall i \geq s.\sigma, i \models \psi_2$, that is $\sigma, s \models \mathsf{G}\psi_2$.

We prove the right-to-left direction for the $\overline{R_6}$ case. Suppose that $\sigma, s \models \mathsf{G}\psi_2$, for a given state sequence σ and position s. It holds that:

$$\begin{aligned}\sigma, s \models \mathsf{G}\psi_2 &\Leftrightarrow \forall i \geq s.\sigma, i \models \psi_2\\ &\Leftrightarrow \forall i \geq s.\forall j \geq i.\sigma, j \models \psi_2\\ &\Rightarrow \forall i \geq s.\sigma, i \models \psi_1\,\mathsf{R}\,\psi_2\\ &\Leftrightarrow \sigma, s \models \mathsf{G}(\psi_1\,\mathsf{R}\,\psi_2)\end{aligned}$$

Finally, consider the case for the $\overline{R_7}$ strong equivalence. We first prove the left-to-right direction. Suppose that $\sigma, s \models \psi_1 \,\mathsf{R}\, (\mathsf{G}\psi_2)$ for a given state sequence σ and position s. We divide in cases, depending on the semantics of the *release* operator:

1. if $\forall i \geq s.\sigma, i \models \mathsf{G}\psi_2$, then for $i = s$ we have that $\sigma, s \models \mathsf{G}\psi_2$.
2. otherwise, $\exists i \geq s.(\sigma, i \models \psi_1 \wedge \forall s \leq j \leq i.\sigma, j \models \mathsf{G}\psi_2)$. In particular, for $j = s$, $\sigma, s \models \mathsf{G}\psi_2$.

Therefore, in both cases we have that $\sigma, s \models \mathsf{G}\psi_2$. For the right-to-left direction, suppose that $\sigma, s \models \mathsf{G}\psi_2$. Then, $\forall i \geq s.\sigma, i \models \mathsf{G}\psi_2$. This implies that $\sigma, s \models \psi_1 \,\mathsf{R}\, (\mathsf{G}\psi_2)$. This concludes the proof of all the auxiliary strong equivalences.

We can now prove the remaining rules R_3-R_7. Consider first R_3 in the case $i > j$: we have to prove that $\sigma, s \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\psi_2) \;\leftrightarrow\; \sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2))$, for all states sequences σ and all positions s. This can be simply done by means of the auxiliary rules $\overline{R_2}$ and $\overline{R_3}$:

$$
\begin{aligned}
& \sigma, s \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\psi_2) \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^i\mathsf{X}^j\psi_2)) && \text{by rule } \overline{R_2} \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}(\mathsf{Y}^j\mathsf{X}^j\psi_2))) \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}(\psi_2 \wedge \mathsf{Y}^j\top))) && \text{by rule } \overline{R_3} \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2 \wedge \mathsf{Y}^{i-j+j}\top)) \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2 \wedge \mathsf{Y}^i\top)) \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2))
\end{aligned}
$$

Consider now the rule R_3 in the case $i \leq j$. We have to prove that $\sigma, s \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\psi_2) \;\leftrightarrow\; \sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathsf{R}\, \psi_2)$. This can be done using the auxiliary equivalences $\overline{R_1}$ and $\overline{R_3}$:

$$
\begin{aligned}
& \sigma, s \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\psi_2) \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^j((\mathsf{Y}^j\mathsf{X}^i\psi_1) \,\mathsf{R}\, \psi_2) && \text{by rule } \overline{R_1} \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}(\mathsf{Y}^i\mathsf{X}^i\psi_1)) \,\mathsf{R}\, \psi_2) \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}(\psi_1 \wedge \mathsf{Y}^i\top)) \,\mathsf{R}\, \psi_2) && \text{by rule } \overline{R_3} \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1 \wedge \mathsf{Y}^{j-i+i}\top) \,\mathsf{R}\, \psi_2) \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1 \wedge \mathsf{Y}^j\top) \,\mathsf{R}\, \psi_2) \\
\leftrightarrow\; & \sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathsf{R}\, \psi_2)
\end{aligned}
$$

Consider the R_4 rule in the case $i > j$. It holds that:

$$\begin{aligned}
&\sigma \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\mathsf{X}^j(\psi_2\,\mathsf{R}\,\psi_3)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1\,\mathsf{R}\,(\mathsf{Y}^i\mathsf{X}^j(\psi_2\,\mathsf{R}\,\psi_3))) && \text{by rule } \overline{R_2} \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1\,\mathsf{R}\,(\mathsf{Y}^{i-j}\mathsf{Y}^j\mathsf{X}^j(\psi_2\,\mathsf{R}\,\psi_3))) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1\,\mathsf{R}\,(\mathsf{Y}^{i-j}(\psi_2\,\mathsf{R}\,\psi_3 \wedge \mathsf{Y}^j\top))) && \text{by rule } \overline{R_3} \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1\,\mathsf{R}\,(\mathsf{Y}^{i-j}(\psi_2\,\mathsf{R}\,\psi_3) \wedge \mathsf{Y}^i\top)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1\,\mathsf{R}\,(\mathsf{Y}^{i-j}(\psi_2\,\mathsf{R}\,\psi_3))) \wedge \mathsf{X}^i(\psi_1\,\mathsf{R}\,\mathsf{Y}^i\top) && \text{by rule } R_1 \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1\,\mathsf{R}\,(\mathsf{Y}^{i-j}(\psi_2\,\mathsf{R}\,\psi_3))) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1\,\mathsf{R}\,((\mathsf{Y}^{i-j}\psi_2)\,\mathsf{R}\,(\mathsf{Y}^{i-j}\psi_3))) && \text{by rule } \overline{R_4}
\end{aligned}$$

Finally, consider the R_4 rule in the case $i \leq j$. It holds that:

$$\begin{aligned}
&\sigma \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\mathsf{X}^j(\psi_2\,\mathsf{R}\,\psi_3)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^j((\mathsf{Y}^j\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\psi_2\,\mathsf{R}\,\psi_3)) && \text{by rule } \overline{R_1} \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\mathsf{Y}^i\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\psi_2\,\mathsf{R}\,\psi_3)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}(\psi_1 \wedge \mathsf{Y}^i\top))\,\mathsf{R}\,(\psi_2\,\mathsf{R}\,\psi_3)) && \text{by rule } \overline{R_3} \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1 \wedge \mathsf{Y}^j\top)\,\mathsf{R}\,(\psi_2\,\mathsf{R}\,\psi_3)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1)\,\mathsf{R}\,(\psi_2\,\mathsf{R}\,\psi_3))
\end{aligned}$$

Consider the R_5 rule. It can be proven by means of the rules R_4 and $\overline{R_5}$ as follows. For all state sequences σ and all positions s, it holds that:

$$\begin{aligned}
&\sigma, s \models \mathsf{G}\mathsf{X}^i\mathsf{G}\psi \\
\Leftrightarrow\ &\sigma, s \models (\mathsf{X}^0\bot)\,\mathsf{R}\,(\mathsf{X}^i(\bot\,\mathsf{R}\,\psi)) && \text{by definition of } \textit{globally} \text{ operator} \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i((\mathsf{Y}^i\bot)\,\mathsf{R}\,(\bot\,\mathsf{R}\,\psi)) && \text{by rule } R_4 \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\bot\,\mathsf{R}\,(\bot\,\mathsf{R}\,\psi)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\mathsf{G}\mathsf{G}\psi) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i\mathsf{G}\psi && \text{by rule } \overline{R_5}
\end{aligned}$$

Consider the R_6 rule. It can be prove by means of the rules R_4 and

$\overline{R_6}$ as follows. For all state sequences σ and positions s it holds that:

$$\begin{aligned}
&\sigma, s \models \mathsf{G}\mathsf{X}^i(\psi_1 \,\mathsf{R}\, \psi_2) \\
\Leftrightarrow\ &\sigma, s \models ((\mathsf{X}^0\bot) \,\mathsf{R}\, (\mathsf{X}^i(\psi_1 \,\mathsf{R}\, \psi_2))) && \text{by definition of } \textit{globally} \text{ operator} \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i((\mathsf{Y}^i\bot) \,\mathsf{R}\, (\psi_1 \,\mathsf{R}\, \psi_2)) && \text{by rule } R_4 \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\bot \,\mathsf{R}\, (\psi_1 \,\mathsf{R}\, \psi_2)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\mathsf{G}(\psi_1 \,\mathsf{R}\, \psi_2)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\mathsf{G}\psi_2) && \text{by rule } \overline{R_6}
\end{aligned}$$

Consider the R_7 rule. It can be proven by means of the rules R_4 and $\overline{R_7}$ as follows. Let σ be a state sequence and let s be a position. We divide in cases. If $i > j$, then:

$$\begin{aligned}
&\sigma, s \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\mathsf{G}\psi_2) \\
\Leftrightarrow\ &\sigma, s \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j(\bot \,\mathsf{R}\, \psi_2)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, ((\mathsf{Y}^{i-j}\bot) \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2))) && \text{by rule } R_4 \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\bot \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2))) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{G}(\mathsf{Y}^{i-j}\psi_2))) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{G}(\mathsf{Y}^{i-j}\psi_2))) && \text{by rule } R_7 \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^i\mathsf{G}\mathsf{Y}^{i-j}\psi_2
\end{aligned}$$

Otherwise, it holds that $i \leq j$ and:

$$\begin{aligned}
&\sigma, s \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j\mathsf{G}\psi_2) \\
\Leftrightarrow\ &\sigma, s \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, (\mathsf{X}^j(\bot \,\mathsf{R}\, \psi_2)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathsf{R}\, (\bot \,\mathsf{R}\, \psi_2)) && \text{by rule } R_4 \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathsf{R}\, (\mathsf{G}\psi_2)) \\
\Leftrightarrow\ &\sigma, s \models \mathsf{X}^j\mathsf{G}\psi_2 && \text{by rule } \overline{R_7}
\end{aligned}$$

This concludes the case for the rules R_1-R_7.

It remains the case for the R_{flat} rule, for which we have to prove only equivalence. We first prove the left-to-right direction, for all $n \geq 3$. Suppose that:

$$\begin{aligned}
&\sigma, 0 \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\psi_2 \,\mathsf{R}\, (\ldots (\psi_{n-1} \,\mathsf{R}\, \psi_n) \ldots))) \\
\Leftrightarrow\ &\sigma, i \models \psi_1 \,\mathsf{R}\, (\psi_2 \,\mathsf{R}\, (\ldots (\psi_{n-1} \,\mathsf{R}\, \psi_n) \ldots))
\end{aligned}$$

This formula contains exactly n *release* operators. Each of these can be satisfied in two ways: (i) *universally*, that is if for all the future positions the right-hand side formula holds, or (ii) *existentially*, if there exists a position in the future where the left-hand side formula holds and the right-hand side formula holds until then. Therefore, we have a total of 2^{n-1} cases.

We consider first the cases in which there exists a *release* operator that is universally satisfied. These correspond to $2^{n-1}-1$ cases. Let m be the index of the outermost between these operators. Let $k_1 = i$. We have that:

$$\begin{aligned}
&\exists j_1 \geq k_1.(\sigma, j_1 \models \psi_1 \wedge \forall k_1 \leq k_2 \leq j_1. \\
&\exists j_2 \geq k_2.(\sigma, j_2 \models \psi_2 \wedge \cdots \wedge \forall k_{m-1} \leq k_{m-1} \leq j_{m-2}. \\
&\forall k_m \geq k_{m-1}.(\sigma, k_m \models \psi_m \,\mathsf{R}\, (\ldots(\psi_{n-1} \,\mathsf{R}\, \psi_n)\ldots)))\ldots)
\end{aligned}$$

Which is equivalent to:

$$\begin{aligned}
&\exists j_1 \geq k_1.(\sigma, j_1 \models \psi_1 \wedge \forall k_1 \leq k_2 \leq j_1. \\
&\exists j_2 \geq k_2.(\sigma, j_2 \models \psi_2 \wedge \cdots \wedge \forall k_{m-1} \leq k_{m-1} \leq j_{m-2}. \\
&(\sigma, k_{m-1} \models \mathsf{G}(\psi_m \,\mathsf{R}\, (\ldots(\psi_{n-1} \,\mathsf{R}\, \psi_n)\ldots)))\ldots))
\end{aligned}$$

By the repeated application of the $\overline{R_6}$ auxiliary rule $n - m$ times, we have that:

$$\begin{aligned}
&\exists j_1 \geq k_1.(\sigma, j_1 \models \psi_1 \wedge \forall k_1 \leq k_2 \leq j_1. \\
&\exists j_2 \geq k_2.(\sigma, j_2 \models \psi_2 \wedge \cdots \wedge \forall k_{m-1} \leq k_{m-1} \leq j_{m-2}.(\sigma, k_{m-1} \models \mathsf{G}\psi_n)\ldots))
\end{aligned}$$

that is:

$$\begin{aligned}
&\exists j_1 \geq k_1.(\sigma, j_1 \models \psi_1 \wedge \forall k_1 \leq k_2 \leq j_1. \\
&\exists j_2 \geq k_2.(\sigma, j_2 \models \psi_2 \wedge \cdots \wedge \forall k_{m-1} \leq k_{m-1} \leq j_{m-2}. \\
&\forall k \geq k_{m-1}.(\sigma, k \models \psi_n)\ldots))
\end{aligned}$$

In particular, for $k_1 = k_2 = \cdots = k_{m-2} = k_{m-1}$, we have that:

$$\forall k \geq k_1.\sigma, k \models \psi_n$$

Since by definition $k_1 = i$, we have that $\forall k \geq i.\sigma, k \models \psi_n$, and thus $\sigma, 0 \models \mathsf{X}^i((\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \ldots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i\top)))\ \mathsf{R}\ \psi_n)$. The

remaining case is when *all* the *release* operators are existentially satisfied. Suppose that:

$$\exists j_1 \geq k_1.(\sigma, j_1 \models \psi_1 \wedge \forall k_1 \leq k_2 \leq j_1.$$
$$\exists j_2 \geq k_2.(\sigma, j_2 \models \psi_2 \wedge \cdots \wedge \forall k_{n-1} \leq k_{n-1} \leq j_{n-2}.$$
$$\exists j_{n-1} \geq k_{n-1}.(\sigma, j_{n-1} \models \psi_{n-1} \wedge \forall k_{n-1} \leq k_n \leq j_{n-1}.\sigma, k_n \models \psi_n)) \dots)$$

where $k_1 = i$. This implies that:

$$\exists j_1 \geq i.(\sigma, j_1 \models \psi_1 \wedge$$
$$\exists j_2 \geq j_1.(\sigma, j_2 \models \psi_2 \wedge \cdots \wedge$$
$$\exists j_{n-1} \geq j_{n-2}.(\sigma, j_{n-1} \models \psi_{n-1} \wedge \forall i \leq k \leq j_{n-1}.\sigma, k \models \psi_n) \dots))$$

This is equivalent to:

$$\exists j_{n-1} \geq i.(\sigma, j_{n-1} \models \psi_{n-1} \wedge$$
$$\exists i \leq j_{n-2} \leq j_{n-1}.(\sigma, j_{n-2} \models \psi_{n-2} \wedge \cdots \wedge$$
$$\exists i \leq j_1 \leq j_2.(\sigma, j_1 \models \psi_1) \dots) \wedge \forall i \leq k \leq j_{n-1}.\sigma, k \models \psi_n)$$

This in turn is equivalent to:

$$\exists j_{n-1} \geq i.(\sigma, j_{n-1} \models \psi_{n-1} \wedge$$
$$\exists 0 \leq j_{n-2} \leq j_{n-1}.(\sigma, j_{n-2} \models \psi_{n-2} \wedge \cdots \wedge$$
$$\exists 0 \leq j_1 \leq j_2.(\sigma, j_1 \models \psi_1 \wedge \mathsf{Y}^i \top) \dots) \wedge \forall i \leq k \leq j_{n-1}.\sigma, k \models \psi_n)$$

This is the definition of the existential semantics of the formula $(\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \dots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i \top)))\ \mathsf{R}\ \psi_n$, starting from position i. Therefore, $\sigma, 0 \models \mathsf{X}^i((\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \dots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i \top)))\ \mathsf{R}\ \psi_n)$.

We now prove the right-to-left direction for R_{flat}. Suppose that $\sigma, 0 \models \mathsf{X}^i((\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \dots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i \top)))\ \mathsf{R}\ \psi_n)$. Therefore, $\sigma, i \models (\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \dots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i \top)))\ \mathsf{R}\ \psi_n$. We divide in cases:

1. if $\forall j \geq i.\ \sigma, j \models \psi_n$, then $\sigma, 0 \models \mathsf{X}^i(\psi_1\ \mathsf{R}\ (\psi_2\ \mathsf{R}\ (\dots(\psi_{n-1}\ \mathsf{R}\ \psi_n) \dots)))$
2. otherwise, $\exists j \geq i.(\sigma, j \models \psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \dots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i \top) \dots) \wedge \forall i \leq k \leq j.\sigma, k \models \psi_n)$.

With the former case, we are done. Instead, the latter is equivalent to:

$$\exists j_{n-1} \geq i.(\sigma, j_{n-1} \models \psi_{n-1} \wedge$$
$$\exists 0 \leq j_{n-2} \leq j_{n-1}.(\sigma, j_{n-2} \models \psi_{n-2} \wedge \dots$$
$$\exists 0 \leq j_1 \leq j_2.(\sigma, j_1 \models (\psi_1 \wedge \mathsf{Y}^i \top)) \dots) \wedge \forall i \leq k \leq j_{n-1}.\sigma, k \models \psi_n)$$

In turn, this is equivalent to:

$$\begin{aligned}&\exists j_{n-1} \geq i.(\sigma, j_{n-1} \models \psi_{n-1} \wedge \\ &\exists i \leq j_{n-2} \leq j_{n-1}.(\sigma, j_{n-2} \models \psi_{n-2} \wedge \dots \\ &\exists i \leq j_1 \leq j_2.(\sigma, j_1 \models \psi_1) \dots) \wedge \forall i \leq k \leq j_{n-1}.\sigma, k \models \psi_n)\end{aligned}$$

This is equivalent to:

$$\begin{aligned}&\exists j_1 \geq i.(\sigma, j_1 \models \psi_1 \wedge \\ &\exists j_2 \geq j_1.(\sigma, j_2 \models \psi_2 \wedge \dots \\ &\exists j_{n-1} \geq j_{n-2}.(\sigma, j_{n-1} \models \psi_{n-1}) \dots) \wedge \forall i \leq k \leq j_1.\sigma, k \models \psi_n)\end{aligned}$$

which implies that:

$$\begin{aligned}&\exists j_1 \geq i.(\sigma, j_1 \models \psi_1 \wedge \forall i \leq k_1 \leq j_1. \\ &\exists j_2 \geq j_1.(\sigma, j_2 \models \psi_2 \wedge \dots \wedge \forall k_{n-2} \leq k_{n-1} \leq j_{n-1}. \\ &\exists j_{n-1} \geq j_{n-2}.(\sigma, j_{n-1} \models \psi_{n-1} \wedge \forall k_{n-1} \leq k \leq j_{n-1}.\sigma, k \models \psi_n) \dots))\end{aligned}$$

This is the definition of the *existential* semantics of the formula $\psi_1 \,\mathsf{R}\, (\psi_2 \,\mathsf{R}\, (\dots (\psi_{n-1} \,\mathsf{R}\, \psi_n) \dots))$, starting from position i. Therefore, $\sigma, 0 \models \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\psi_2 \,\mathsf{R}\, (\dots (\psi_{n-1} \,\mathsf{R}\, \psi_n) \dots)))$. This concludes the proof of Lemma 23. □

Lemma 24. *Let ψ_1, ψ_2 and ψ_3 be $\mathsf{LTL}{+}\mathsf{P_P}$ formulas. Let ϕ be a formula of type $\mathsf{X}^j\psi_2$, $\mathsf{X}^j\mathsf{G}\psi_2$ or $\mathsf{X}^j(\psi_2 \mathsf{R} \psi_3)$. For each state sequence σ and position i, it holds that:*

1. $\sigma, i \models \mathsf{G}\phi \;\leftrightarrow\; \sigma, i \models \texttt{\textit{resolve\_globally}}(\phi)$
2. $\sigma, i \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, \phi \;\leftrightarrow\; \sigma, i \models \texttt{\textit{resolve\_release}}(\mathsf{X}^i\psi_1, \phi)$

Proof. We prove the second point, for the *release* operator. The subroutine `resolve_release` divides in cases, depending on the structure of ϕ:

- if $\phi = \mathsf{X}^j\psi_2$ and $i > j$, then:

  $$\texttt{resolve\_release}(\mathsf{X}^i\psi_1, \mathsf{X}^j\psi_2) \coloneqq \mathsf{X}^i(\psi_1 \,\mathsf{R}\, (\mathsf{Y}^{i-j}\psi_2))$$

  By rule R_3 of Lemma 23, we have that $\sigma, i \models (\mathsf{X}^i\psi_1) \,\mathsf{R}\, \phi \leftrightarrow \sigma, i \models \texttt{resolve\_release}(\mathsf{X}^i\psi_1, \phi)$.

- if $\phi = \mathsf{X}^j\psi_2$ and $i \leq j$, then

  $$\texttt{resolve\_release}(\mathsf{X}^i\psi_1, \mathsf{X}^j\psi_2) \coloneqq \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1)\,\mathsf{R}\,\psi_2)$$

  By rule R_3 of Lemma 23, we have that $\sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi \leftrightarrow \sigma, i \models \texttt{resolve\_release}(\mathsf{X}^i\psi_1, \phi)$.

- if $\phi = \mathsf{X}^j(\psi_2\,\mathsf{R}\,\psi_3)$ and $i > j$, then

  $$\texttt{resolve\_release}(\mathsf{X}^i\psi_1, \mathsf{X}^j(\psi_2\,\mathsf{R}\,\psi_3)) \coloneqq \mathsf{X}^i(\psi_1\,\mathsf{R}\,((\mathsf{Y}^{i-j}\psi_2)\,\mathsf{R}\,(\mathsf{Y}^{i-j}\psi_3)))$$

  By rule R_4 of Lemma 23, we have that $\sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi \leftrightarrow \sigma, i \models \texttt{resolve\_release}(\mathsf{X}^i\psi_1, \phi)$.

- if $\phi = \mathsf{X}^j(\psi_2\,\mathsf{R}\,\psi_3)$ and $i \leq j$, then

  $$\texttt{resolve\_release}(\mathsf{X}^i\psi_1, \mathsf{X}^j(\psi_2\,\mathsf{R}\,\psi_3)) \coloneqq \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1)\,\mathsf{R}\,(\psi_2\,\mathsf{R}\,\psi_3))$$

  By rule R_4 of Lemma 23, we have that $\sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi \leftrightarrow \sigma, i \models \texttt{resolve\_release}(\mathsf{X}^i\psi_1, \phi)$.

- if $\phi = \mathsf{X}^j\mathsf{G}\psi_2$ and $i > j$, then

  $$\texttt{resolve\_release}(\mathsf{X}^i\psi_1, \mathsf{X}^j\mathsf{G}\psi_2) \coloneqq \mathsf{X}^i\mathsf{G}\mathsf{Y}^{i-j}\psi_2$$

  By rule R_7 of Lemma 23, we have that $\sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi \leftrightarrow \sigma, i \models \texttt{resolve\_release}(\mathsf{X}^i\psi_1, \phi)$.

- if $\phi = \mathsf{X}^j\mathsf{G}\psi_2$ and $i \leq j$, then

  $$\texttt{resolve\_release}(\mathsf{X}^i\psi_1, \mathsf{X}^j\mathsf{G}\psi_2) \coloneqq \mathsf{X}^j\mathsf{G}\psi_2$$

  By rule R_7 of Lemma 23, we have that $\sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi \leftrightarrow \sigma, i \models \texttt{resolve\_release}(\mathsf{X}^i\psi_1, \phi)$.

The case for $\texttt{resolve\_globally}(\phi)$ is analogous. □

Lemma 25 (Soundness of $\mathsf{applyR1R7}(\cdot)$)**.** *For any* $\mathsf{Pastified\text{-}LTL_{EBR}{+}P}$ *formula* ϕ*, for any state sequence* σ *and for any position* i*, it holds that* $\sigma, i \models \phi$ *iff* $\sigma, i \models \mathit{applyR1R7}(\phi)$*.*

Proof. Consider the pseudo-code of $\mathsf{applyR1R7}(\cdot)$ as described in Fig. 7.2. We prove this claim by induction on the complexity of formula ϕ.

The base case corresponds to the case when ϕ is a $\mathsf{LTL}+\mathsf{P_P}$ formula. In this case, the $\mathsf{applyR1R7}(\cdot)$ algorithm returns ϕ it self. Obviously, ϕ is strongly equivalent to $\mathsf{applyR1R7}(\phi)$

For the inductive step, we divide in cases. If $\phi := \mathsf{X}\phi_1$, then $\sigma, i+1 \models \phi_1$. By inductive hypothesis $\sigma', i' \models \phi_1$ iff $\sigma', i' \models \mathsf{applyR1R7}(\phi_1)$, for all state sequences σ' and positions i'. Therefore:

$$\begin{aligned}\sigma, i \models \mathsf{X}\phi_1 &\Leftrightarrow \sigma, i+1 \models \phi_1 \\ &\Leftrightarrow \sigma, i+1 \models \mathsf{applyR1R7}(\phi_1) \qquad \text{by inductive hypothesis} \\ &\Leftrightarrow \sigma, i \models \mathsf{X}(\mathsf{applyR1R7}(\phi_1))\end{aligned}$$

In general, $\mathsf{applyR1R7}(\phi_1)$ is a conjunction of formulas of type $\mathsf{X}^j\psi$, $\mathsf{X}^j\mathsf{G}\psi$, $\mathsf{X}^j((\mathsf{X}^k\psi_1)\,\mathsf{R}\,\psi_2)$, that is:

$$\mathsf{applyR1R7}(\phi_1) \coloneqq \phi_2^c \wedge \cdots \wedge \phi_n^c$$

and thus:

$$\sigma, i \models \mathsf{X}\phi_1 \;\Leftrightarrow \sigma, i \models \mathsf{X}(\phi_2^c \wedge \cdots \wedge \phi_n^c)$$

Using rule R_1 of Lemma 23, we have that:

$$\begin{aligned}\sigma, i \models \mathsf{X}\phi_1 &\Leftrightarrow \sigma, i \models \mathsf{X}(\phi_2^c \wedge \cdots \wedge \phi_n^c) \\ &\Leftrightarrow \sigma, i \models \mathsf{X}\phi_2^c \wedge \cdots \wedge \mathsf{X}\phi_n^c \qquad \text{by rule } R_1 \text{ of Lemma 23} \\ \sigma, i \models \phi &\Leftrightarrow \sigma, i \models \mathsf{applyR1R7}(\phi)\end{aligned}$$

This concludes the case for $\phi := \mathsf{X}\phi_1$. Consider the case $\phi := (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi_1$. Since by inductive hypothesis $\sigma', i' \models \phi_1$ iff $\sigma', i' \models \mathsf{applyR1R7}(\phi_1)$, for all state sequences σ' and positions i', we have that:

$$\begin{aligned}\sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi_1 &\Leftrightarrow \sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\mathsf{applyR1R7}(\phi_1)) \\ &\qquad\qquad \sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\phi_2^c \wedge \cdots \wedge \phi_n^c)\end{aligned}$$

where ϕ_i^c is a formula of type $\mathsf{X}^j\psi$, $\mathsf{X}^j\mathsf{G}\psi$, $\mathsf{X}^j((\mathsf{X}^k\psi_1)\,\mathsf{R}\,\psi_2)$, for each $1 < i \leq n$. By rule R_2 of Lemma 23, we have that:

$$\begin{aligned}\sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi_1 &\Leftrightarrow \sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\phi_2^c \wedge \cdots \wedge \phi_n^c) \\ &\Leftrightarrow \sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\phi_2^c) \wedge \cdots \wedge (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\phi_n^c)\end{aligned}$$

Let $\phi_i^r \equiv \texttt{resolve\_release}(\mathsf{X}^i\psi_1, \phi_i^c)$, for all $1 < i \leq n$. By Lemma 24:

$$\begin{aligned}\sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi_1 &\Leftrightarrow \sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\phi_2^c) \wedge \cdots \wedge (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\phi_n^c)\\ &\Leftrightarrow \sigma, i \models (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\phi_2^r) \wedge \cdots \wedge (\mathsf{X}^i\psi_1)\,\mathsf{R}\,(\phi_n^r)\\ &\quad \text{by Lemma 24}\\ &\Leftrightarrow \sigma, i \models \mathsf{applyR1R7}(\phi) \qquad \text{by definition of } \mathsf{applyR1R7}\end{aligned}$$

This concludes the case for $\phi := \phi := (\mathsf{X}^i\psi_1)\,\mathsf{R}\,\phi_1$. The case for the *globally* operator is analogous to the proof for the *release* one. □

Lemma 26 (Soundness of $\mathsf{flatten}(\cdot)$)**.** *For any* Pastified-$\mathsf{LTL}_{\mathsf{EBR}}$+P *formula* ϕ*, it holds that* $\phi \equiv \mathit{flatten}(\phi)$*.*

Proof. We prove this lemma by induction on the number n of top-level conjucts or disjuncts. The base case corresponds to the case of $n = 0$. We divide in cases:

- if $\phi := \mathsf{X}^i(\psi_1\,\mathsf{R}\,(\psi_2\,\mathsf{R}\,(\ldots(\psi_{n-1}\,\mathsf{R}\,\psi_n)\ldots)))$, then $\mathsf{flatten}(\phi) := \mathsf{X}^i((\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \ldots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i\top)\ldots))\,\mathsf{R}\,\psi_n)$. By the R_{flat} rule of Lemma 23, $\phi \equiv \mathsf{flatten}(\phi)$.
- otherwise, the $\mathsf{flatten}$ algorithm falls in the `default` case. In this case, $\mathsf{flatten}(\phi) := \phi$, and obviously $\phi \equiv \mathsf{flatten}(\phi)$.

For the inductive step, we divide in cases as well.

- if $\phi := \phi_1 \wedge \phi_2$, then by inductive hypothesis $\phi_1 \equiv \mathsf{flatten}(\phi_1)$ and $\phi_2 \equiv \mathsf{flatten}(\phi_2)$. Thus $\phi \equiv \mathsf{flatten}(\phi_1) \wedge \mathsf{flatten}(\phi_2)$, that is $\phi \equiv \mathsf{flatten}(\phi)$.
- if $\phi := \phi_1 \wedge \phi_2$, then by inductive hypothesis $\phi_1 \equiv \mathsf{flatten}(\phi_1)$ and $\phi_2 \equiv \mathsf{flatten}(\phi_2)$. Thus $\phi \equiv \mathsf{flatten}(\phi_1) \vee \mathsf{flatten}(\phi_2)$, that is $\phi \equiv \mathsf{flatten}(\phi)$.

□

Lemma 17 (Soundness of $\mathsf{normalize}(\cdot)$)**.** *For any* Pastified-$\mathsf{LTL}_{\mathsf{EBR}}$+P *formula* ϕ*, it holds that* ϕ *and* $\mathit{normalize}(\phi)$ *are equivalent and* $\mathit{normalize}(\phi)$ *is a* Normal-$\mathsf{LTL}_{\mathsf{EBR}}$+P *formula.*

Proof. Recall that $\mathsf{normalize}(\phi)$ is defined as the formula $\mathsf{flatten}(\mathsf{applyR1R7}(\phi))$, where $\mathsf{applyR1R7}$ is the algorithm in Fig. 7.2 and $\mathsf{flatten}$ is the algorithm in Fig. 7.4. By Lemma 25, for each state sequence σ and

position i, we have that $\sigma, i \models \phi$ iff $\sigma, i \models \mathsf{applyR1R7}(\phi)$. In particular, for $i = 0$, this means that $\phi \equiv \mathsf{applyR1R7}(\phi)$. By Lemma 26, we have that $\mathsf{flatten}(\mathsf{applyR1R7}(\phi)) \equiv \mathsf{applyR1R7}(\phi)$, and thus $\phi \equiv \mathsf{flatten}(\mathsf{applyR1R7}(\phi))$, and by definition $\phi \equiv \mathsf{normalize}(\phi)$.

Finally, it is easy to see that all the rules of Lemma 23, except for R_4, replace a formula with a one in $\mathsf{Normal\text{-}LTL_{EBR}{+}P}$. Thus $\mathsf{normalize}(\phi)$ would be a $\mathsf{Normal\text{-}LTL_{EBR}{+}P}$ formula if we did not consider the nested *release* operators. Since this is exactly the case solved by the R_{flat} rule and thus by the $\mathsf{flatten}$ algorithm (which produces a formula in normal form), we have that $\mathsf{flatten}(\mathsf{applyR1R7}(\phi))$, which by definition is $\mathsf{normalize}(\phi)$, is in $\mathsf{Normal\text{-}LTL_{EBR}{+}P}$. □

Lemma 18 (Complexity of $\mathsf{normalize}(\cdot)$)**.** *For any* $\mathsf{Pastified\text{-}LTL_{EBR}{+}P}$ *formula* ϕ*,* $\mathsf{normalize}(\phi)$ *can be built in* $\mathcal{O}(n)$ *time, and the size of* $\mathsf{normalize}(\phi)$ *is* $\mathcal{O}(n)$*, where* $n = |\phi|$*.*

Proof. Since $\mathsf{normalize}(\phi) := \mathsf{flatten}(\mathsf{applyR1R7}(\phi))$, we study the complexity of both $\mathsf{applyR1R7}$ and $\mathsf{flatten}$. At each iteration, algorithm $\mathsf{applyR1R7}(\phi)$ makes at most one recursive call on a formula ϕ' of size $|\phi'| < |\phi|$ and thus it stop at most after $\mathcal{O}(n)$ iterations. The same holds for $\mathsf{flatten}$. At each iteration, $\mathsf{applyR1R7}$ and $\mathsf{flatten}$ produce a formula of constant size with respect to the size of the formula produced by the recursive call; therefore the recurrence equation describing the size of the formula produced by $\mathsf{normalize}(\phi)$ is:

$$S(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1 \\ S(n-1) + \mathcal{O}(1) & \text{otherwise} \end{cases}$$

Therefore:

$$\begin{aligned} S(n) &= S(n-1-i) + i \cdot \mathcal{O}(1) \\ &= S(1) + \mathcal{O}(n) && \text{for } i = n-2 \\ &\in \mathcal{O}(n) \end{aligned}$$

□

A.2 Formalization into Timed Interface Automata (Chapter 9)

We recall here the basic definitions. Let $\mathcal{C}$ be a set of variables over the time domain $\mathbb{Q}$. A *clock constraint* is a boolean combination of

formulae of type $x < c$ or $x \leq y + c$, where $c \in \mathbb{Q}$ and $x, y \in \mathcal{C}$. We denote with $\Xi(\mathcal{C})$ the set of all the clock constraints over $\mathcal{C}$.

Definition 64 (Timed Interface Automata)**.** *A* timed interface automaton
(TIA, for short) $\mathcal{A} = \langle V_\mathcal{A}, v^0_\mathcal{A}, \Sigma^I_\mathcal{A}, \Sigma^O_\mathcal{A}, C_\mathcal{A}, inv^I_\mathcal{A}, inv^O_\mathcal{A}, T_\mathcal{A} \rangle$ *consists of:*

- *a finite set of locations* $V_\mathcal{A}$*;*
- *an initial location* $v^0_\mathcal{A} \in V_\mathcal{A}$*;*
- *an input alphabet* $\Sigma^I_\mathcal{A}$ *and an output alphabet* $\Sigma^O_\mathcal{A}$*, such that* $\Sigma^I_\mathcal{A} \cap \Sigma^O_\mathcal{A} = \varnothing$*; we define the alphabet of* $\mathcal{A}$ *as* $\Sigma_\mathcal{A} = \Sigma^I_\mathcal{A} \cup \Sigma^O_\mathcal{A}$*;*
- *a finite set of clocks* $C_\mathcal{A}$*, where a clock is a real-valued variable;*
- *an* input invariant $inv^I_\mathcal{A} : V_\mathcal{A} \to \Xi(C_\mathcal{A})$ *for each location;*
- *an* output invariant $inv^O_\mathcal{A} : V_\mathcal{A} \to \Xi(C_\mathcal{A})$ *for each location;*
- *a transition relation* $T_\mathcal{A} \subseteq V_\mathcal{A} \times \Sigma_\mathcal{A} \times 2^{C_\mathcal{A}} \times \Xi(C_\mathcal{A}) \times V_\mathcal{A}$*.*

We give the semantics of a TIA in terms of traces, which basically are the same as in timed automata, except that an action can be either an input or an output.

Definition 65 (Trace of a TIA)**.** *A* trace τ *of a TIA* $\mathcal{A} = \langle V_\mathcal{A}, v^0_\mathcal{A}, \Sigma^I_\mathcal{A}, \Sigma^O_\mathcal{A}, C_\mathcal{A}, inv^I_\mathcal{A}, inv^O_\mathcal{A}, T_\mathcal{A} \rangle$ *is a (either finite or infinite) sequence of* states *of the form:*

$$\langle v_0, \nu_0 \rangle \xrightarrow{\alpha_1} \langle v_1, \nu_1 \rangle \xrightarrow{\alpha_2} \langle v_2, \nu_2 \rangle \xrightarrow{\alpha_3} \dots$$

such that $v_i \in V_\mathcal{A}$*,* $\alpha_i \in \Sigma_\mathcal{A} \cup \mathbb{Q}$ *and* $\nu_i \in \mathcal{V}_{C_\mathcal{A}}$ *for all* $i \geq 0$*, and:*

- (initiation) $v_0 = v^0_\mathcal{A}$*,* $\nu_0(x) = 0$ *for all* $x \in C_\mathcal{A}$*, and* $\nu_0 \models inv^I_\mathcal{A}(v^0_\mathcal{A}) \wedge inv^O_\mathcal{A}(v^0_\mathcal{A})$*;*
- (consecution)*: for all* $i \geq 0$
  - *(timed transition) if* $\alpha \in \mathbb{Q}$*, then* $v_{i+1} = v_i$ *and* $\nu_{i+1} = \nu_i + \alpha$ *and* $\nu_i + \delta \models inv^I_\mathcal{A}(v_i) \wedge inv^O_\mathcal{A}(v_i)$*, for all* $0 \leq \delta \leq \alpha$*;*
  - *(discrete transition) for each player* $\gamma \in \{I, O\}$*, if* $\alpha \in \Sigma^\gamma_\mathcal{A}$ *then there is a tuple* $(v_i, \alpha, R_i, \Xi_i, v_{i+1}) \in T_\mathcal{A}$ *such that: (i)* $\nu_i \models inv^\gamma_\mathcal{A}(v_i) \wedge \Xi_i$*; (ii)* $\nu_{i+1} = \nu_i[R_i \mapsto 0]$*; (iii)* $\nu_{i+1} \models inv^\gamma_\mathcal{A}(v_{i+1})$*.*

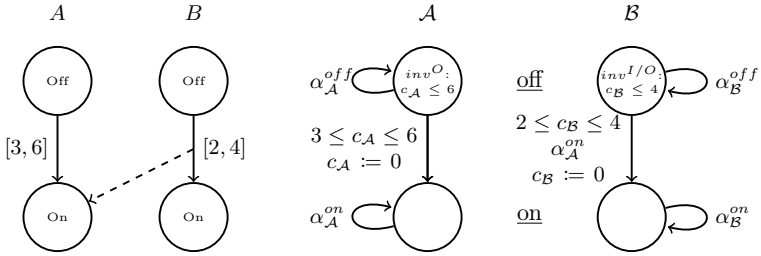
For all $\alpha \in \Sigma_{\mathcal{A}} \cup \mathbb{Q}$, we define $time(\alpha) = \alpha$ if $\alpha \in \mathbb{Q}$, and $time(\alpha) = 0$ otherwise. A trace τ is said to be *diverging* iff $\sum_{k=1}^{|\tau|} time(\alpha_k) = \infty$. A state $\langle v, \nu \rangle$ is said to be *reachable* in $\mathcal{A}$ iff there exists a trace of $\mathcal{A}$ containing it. As in [69], we say that two TIAs $\mathcal{A}$ and $\mathcal{B}$ are *composable* iff $\Sigma^O_{\mathcal{A}} \cap \Sigma^O_{\mathcal{B}} = \varnothing$ and $C_{\mathcal{A}} \cap C_{\mathcal{B}} = \varnothing$. If $\mathcal{A}$ and $\mathcal{B}$ are composable, then we define $shared(\mathcal{A}, \mathcal{B}) = \Sigma_{\mathcal{A}} \cap \Sigma_{\mathcal{B}}$, that is the set of actions on which the two TIAs can synchronize. We now define the product of two TIAs: the two automata will synchronize on the shared actions and in this case they both have to satisfy their clock constraints, and asynchronously interleave on all the other actions (in this case only the clock constraints of the automaton that moves have to be satisfied.)

Definition 66 (Product of TIAs)**.** *Given two composable TIAs $\mathcal{A}$ and $\mathcal{B}$, their product is the TIA $\mathcal{A} \otimes \mathcal{B}$ defined as follows:*

- $V_{\mathcal{A}\otimes\mathcal{B}} = V_{\mathcal{A}} \times V_{\mathcal{B}}$ *and* $v^0_{\mathcal{A}\otimes\mathcal{B}} = (v^0_{\mathcal{A}}, v^0_{\mathcal{B}})$*;*
- $\Sigma^I_{\mathcal{A}\otimes\mathcal{B}} = (\Sigma^I_{\mathcal{A}} \cup \Sigma^I_{\mathcal{B}}) \setminus shared(\mathcal{A}, \mathcal{B})$ *and* $\Sigma^O_{\mathcal{A}\otimes\mathcal{B}} = \Sigma^O_{\mathcal{A}} \cup \Sigma^O_{\mathcal{B}}$*;*
- $C_{\mathcal{A}\otimes\mathcal{B}} = C_{\mathcal{A}} \cup C_{\mathcal{B}}$*;*
- $inv^I_{\mathcal{A}\otimes\mathcal{B}}(v, u) = inv^I_{\mathcal{A}}(v) \wedge inv^I_{\mathcal{B}}(u)$ *and* $inv^O_{\mathcal{A}\otimes\mathcal{B}}(v, u) = inv^O_{\mathcal{A}}(v) \wedge inv^O_{\mathcal{B}}(u)$*;*
- *the transition relation is defined as follows:*

$$
\begin{aligned}
T_{\mathcal{A}\otimes\mathcal{B}} = \{&((v,u), \alpha, R, \Xi, (v', u)) \mid \\
&\quad \exists \alpha \notin shared(\mathcal{A}, \mathcal{B}).(v, \alpha, R, \Xi, v') \in T_{\mathcal{A}}\} \cup \\
&\{((v,u), \alpha, R, \Xi, (v, u')) \mid \\
&\quad \exists \alpha \notin shared(\mathcal{A}, \mathcal{B}).(u, \alpha, R, \Xi, u') \in T_{\mathcal{B}}\} \cup \\
&\{((v,u), \alpha, R, \Xi, (v', u')) \mid \\
&\quad \exists \alpha \in shared(\mathcal{A}, \mathcal{B}).(v, \alpha, R_{\mathcal{A}}, \Xi_{\mathcal{A}}, v') \in T_{\mathcal{A}} \wedge \\
&\quad (u, \alpha, R_{\mathcal{B}}, \Xi_{\mathcal{B}}, u') \in T_{\mathcal{B}} \wedge \\
&\quad R = R_{\mathcal{A}} \cup R_{\mathcal{B}} \wedge \Xi = (\Xi_{\mathcal{A}} \wedge \Xi_{\mathcal{B}})\}
\end{aligned}
$$

The definition of the product between two TIAs inherits the union of the clock resets and the conjunction of the clock constraints from the definition of intersection between timed automata [5], while it inherits the interleaving between no-shared actions from the definition of the product between interface automata [67].



(a) Set of local requirements. (b) Corresponding TIAs.

Example. Consider for example the set of local requirements in Fig. A.1a: each one corresponds to a TIA depicted in Fig. A.1b. We have that $\Sigma_{\mathcal{A}}^O = \{\alpha_{\mathcal{A}}^{off}, \alpha_{\mathcal{A}}^{on}\}$, while $\Sigma_{\mathcal{A}}^I = \varnothing$; instead, $\Sigma_{\mathcal{B}}^O = \{\alpha_{\mathcal{B}}^{off}, \alpha_{\mathcal{B}}^{on}\}$ and $\Sigma_{\mathcal{B}}^I = \{\alpha_{\mathcal{A}}^{on}\}$. Therefore, it holds that $shared(\mathcal{A}, \mathcal{B}) = \{\alpha_{\mathcal{A}}^{on}\}$. In the product TIA $\mathcal{A} \otimes \mathcal{B}$, each state of the form $((\texttt{off}, \texttt{off}), \nu)$ such that $2 \leq \nu(c_{\mathcal{A}}) = \nu(c_{\mathcal{B}}) \leq 4$ is (reachable and) an *immediate illegal state*, since $\alpha_{\mathcal{A}}^{on} \in shared(\mathcal{A}, \mathcal{B})$ and it is accepted by $\mathcal{B}$ in state $(\texttt{off}, \nu)$ but it is not issued by $\mathcal{A}$ in state $(\texttt{off}, \nu)$. In order to remove all these immediate illegal states, we strengthen the input invariant on location `off` of $\mathcal{B}$ to $inv^I : c_{\mathcal{B}} < 2$. Now there exists a strategy for Output player such that for all strategies of Input player the time converges and Input is always to blame: in fact, it suffices that Output chooses a timed move $\alpha \geq 2$. Therefore, every state of the form $((\texttt{off}, \texttt{off}), \nu)$ of $\mathcal{A} \otimes \mathcal{B}$ in which we removed all the immediate illegal states and such that $\nu(c_{\mathcal{A}}) = \nu(c_{\mathcal{B}}) < 2$ is a *time illegal state*, since it is reachable and I-live. In particular, $((\texttt{off}, \texttt{off}), \nu[c_{\mathcal{A}} \mapsto 0, c_{\mathcal{B}} \mapsto 0])$ is a time illegal state, and thus $\mathcal{A}$ and $\mathcal{B}$ are *not* compatible.

Compatibility of Timed Interface Automata In this subsection, we give a characterization of the three problems we want to solve (see Section 9.2) in terms of automata-theoretic problems. In particular, we show the strong relation between local requirements (as defined in Definition 56) and timed interface automata.

As in the original paper on TIAs [69], we consider two types of illegal states: *immediate* and *time*. A state $((v, u), \nu) \in V_{\mathcal{A} \otimes \mathcal{B}} \times \mathcal{V}$ is an *immediate illegal state* iff there exists a *shared* action accepted by $\mathcal{A}$ in (v, ν) but not accepted by $\mathcal{B}$ in (u, ν), or viceversa. Conversely, a *time illegal state* $((v, u), \nu) \in V_{\mathcal{A} \otimes \mathcal{B}} \times \mathcal{V}$ is a state reachable in

$\mathcal{A} \otimes \mathcal{B}$, but not I-live in the TIA $\mathcal{A} \otimes \mathcal{B}$ after removing (*i.e.*, making not reachable [1]) all the immediate illegal states. We call *illegal states* the set of all immediate and time illegal states. Therefore, given a product of TIAs where all the immediate illegal states are not reachable, a time illegal state is a state from which there exists a strategy for the Output player such that for all strategies for the Input player the time converges and the Input player is always to blame beyond some point. We define compatibility between TIAs as follows.

Definition 67 (Compatibility between TIAs)**.** *Two composable TIAs* $\mathcal{A}$ *and* $\mathcal{B}$ *are* compatible *(or equivalently* $\mathcal{A} \otimes \mathcal{B}$ *is* compatible*) iff* $((v^0_{\mathcal{A}}, v^0_{\mathcal{B}}), \nu[C_{\mathcal{A}\otimes\mathcal{B}} \mapsto 0])$ *is* not *a time illegal state.*

Finally, we give a brief overview of the translation from our local requirements defined in Definition 56 to TIAs. Intuitively, each local requirement C_i corresponds to a TIA $\mathcal{A}_{C_i}$ such that (i) its output alphabet contains one letter for each phase, and these are issued when $\mathcal{A}_{C_i}$ is in the corresponding phase; (ii) the input letters of $\mathcal{A}_{C_i}$ corresponds to the output letters of the TIAs which $\mathcal{A}_{C_i}$ depends on; (iii) there is only one clock, which is reset after each discrete transition; (iv) each transition is labeled by the clock constraint of the corresponding target phase. Given a system $S = \langle C_1, \ldots, C_n \rangle$, we can now formalize the three problems described in Section 9.2 as problems over the product TIA $\mathcal{A}_S = \mathcal{A}_{C_1} \otimes \cdots \otimes \mathcal{A}_{C_n}$, in this way: (i) *consistency checking*: it corresponds to checking the reachability of state $(|C_1|, \ldots, |C_n|)$ on the product automaton $\mathcal{A}_S$; (ii) *compatibility checking*: it corresponds to checking the *compatibility* of $\mathcal{A}_S$, namely: the set of local requirements S is safely decomposable iff $\mathcal{A}_1 \ldots \mathcal{A}_n$ are compatible; (iii) *synthesis*: it corresponds to the synthesis of the set of all the TIAs $\mathcal{A}'_{C_1}, \ldots, \mathcal{A}'_{C_n}$ such that $\mathcal{A}'_S = \mathcal{A}'_{C_1} \otimes \cdots \otimes \mathcal{A}'_{C_n}$ is compatible and $\mathcal{L}(\mathcal{A}'_S) \subseteq \mathcal{L}(\mathcal{A}_S)$.

[1] A state $((v, u), \nu) \in V_{\mathcal{A}\otimes\mathcal{B}} \times \mathcal{V}$ can be made *not reachable* by strenghtening the input or output invariants of the previous state(s). For example, in Fig. 9.1a we can make the state $((A.1, B.2), 4)$ not reachable by strenghtening the input invariant of location $B.1$ to $c_1^A < 4$.

A.3 Quantifier-free encoding for requirements with only finite bounds (Chapter 9)

In this section, we will see how the restriction of having only finite bounds on each transition of the system can lead to a simpler (but equisatisfiable) encoding, not featuring the (potentially very expensive) double alternation of existential and universal quantifiers in Eq. (9.2). Since the proof in the case of weak semantics is a bit involved, we start with strict semantics first and then reuse the same argument to prove the result for weak semantics as well. From now on, we call *finite-bounds* a system that does not have infinite bounds on any of its transitions.

A.3.1 Positive Dependencies

Each of our dependency formula does *not* contain negations, *i.e.*, it is a boolean combination of theory atoms, which occur always *positive*. This choice was guided by the fact that negation in our case does not add expressive power. In fact, since we have a finite number of phases, a dependency of the type $\phi := \neg(d, j)$ can be rewritten as:

$$\bigvee_{\substack{1 \leq c \leq n \\ 1 \leq i \leq m_c \\ (c,i) \neq (d,j)}} (c, i)$$

Thus negation does not add expressive power, although it can add succinctness. Since our dependencies contain only theory atoms occurring positive and contain no negation, we have that the negation normal form of $\neg\phi$ (which we refer to as $\mathrm{nnf}(\neg\phi)$) contains only negated theory atoms. Therefore, we can rewrite for example the following formula:

$$\phi_i^c[(d, j) \mapsto (r_j^d \wedge s_j^d \prec \overline{s} \preceq s_{j+1}^d)]$$

as the equivalent one:

$$\mathrm{nnf}(\neg\phi_i^c)[\neg(d, j) \mapsto (\neg r_j^d \vee \overline{s} \preceq s_j^d \vee s_{j+1}^d \prec \overline{s})]$$

A.3.2 Strict Semantics

Suppose we add to the S encoding also the constraint (for all components c and phases i)

$$(r^c_{i-1} \wedge \neg r^c_i) \rightarrow (t^c_i = t^c_{i-1} + u^c_{i-1}) \tag{A.1}$$

and modify Eq. (9.2) as follows:

$$(r^c_{i-1} \wedge \neg r^c_i) \rightarrow \text{VIOLATION}(t^c_i) \tag{A.2}$$

We call this encoding S_{fin} and we define $\text{S}^{\text{ill}}_{\text{fin}} \coloneqq \text{S}_{\text{fin}} \wedge \text{ILL}$ and $\text{S}^{\text{cons}}_{\text{fin}} \coloneqq \text{S}_{\text{fin}} \wedge \text{END}$. It is easy to see that S^{cons} is equivalent to $\text{S}^{\text{cons}}_{\text{fin}}$. Now let's prove it for $\text{S}^{\text{ill}}_{\text{fin}}$ as well.

Proposition 20 (Equisatisfiability under strict semantics)**.** S^{ill} *and* $\text{S}^{\text{ill}}_{\text{fin}}$ *are equisatisfiable for finite-bounds systems under strict semantics.*

Proof. Let's first prove that if S^{ill} is satisfiable, then so is $\text{S}^{\text{ill}}_{\text{fin}}$. Let σ be a model of S^{ill}; there surely exists a variable r^c_i such that $\sigma(r^c_i) = \bot$ and thus there exists a value for t^c_{ill} such that (in particular) $\overline{t} = t^c_{i-1} + u^c_{i-1}$ (this because the bound of phase i of component c is finite by hypothesis) and $\text{VIOLATION}(\overline{t})$ is true: let $\text{val}(\overline{t})$ be the aforesaid value of $\overline{t}$. Now, let σ' be equal to σ except that $\sigma'(t^c_i) = \text{val}(\overline{t})$. Since by construction σ' satisfies Eq. (A.1) and Eq. (A.2), we have that $\sigma' \models \text{S}^{\text{ill}}_{\text{fin}}$.

Now let's prove that if $\text{S}^{\text{ill}}_{\text{fin}}$ is satisfiable, then so is S^{ill}. Let σ be a model of $\text{S}^{\text{ill}}_{\text{fin}}$; we show that $\sigma \models \text{S}^{\text{ill}}$ as well. In fact, if we take $t^c_{ill} = \sigma(t^c_i) = \sigma(t^c_{i-1}) + u^c_{i-1}$, then it is easy to see that the body of the $\exists$ in Eq. (9.2) is true and thus Eq. (9.2) itself is true as well. □ □

A.3.3 Weak Semantics

In case of weak semantics, we can give a similar result of the previous subsection, but the proof is more involved. We call W_{fin} the encoding equal to W except that we add constraint Eq. (A.1) and we replace Eq. (9.2) with

$$(r^c_{i-1} \wedge \neg r^c_i) \rightarrow \text{WEAKVIOL}(t^c_i) \tag{A.3}$$

where

$$\begin{aligned}
&\text{WEAKVIOL}(t_i^c) \;\coloneqq && \text{(A.4)}\\
&\quad \neg\phi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq t_i^c < t_{j+1}^d)] \vee && \text{(A.5)}\\
&\quad \neg\psi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq t_i^c)] \vee && \text{(A.6)}\\
&\quad \exists\widetilde{t}(t_{i-1}^c \leq \widetilde{t} \leq t_i^c \wedge \neg\psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \widetilde{t} < t_{j+1}^d)]) && \text{(A.7)}
\end{aligned}$$

We define $\mathrm{W}_{\text{fin}}^{\text{cons}} \coloneqq \mathrm{W}_{\text{fin}} \wedge \text{END}$ and $\mathrm{W}_{\text{fin}}^{\text{ill}} \coloneqq \mathrm{W}_{\text{fin}} \wedge \text{ILL}$. It is straightforward to see that W^{cons} and $\mathrm{W}_{\text{fin}}^{\text{cons}}$ are equivalent. We prove the following result:

Proposition 21 (Equisatisfiability under weak semantics)**.** W^{ill} *and* $\mathrm{W}_{\text{fin}}^{\text{ill}}$ *are equisatisfiable for finite-bounds systems under weak semantics.*

Proof. $\Rightarrow$**.** We first prove that if W^{ill} is satisfiable, than $\mathrm{W}_{\text{fin}}^{\text{ill}}$ is satisfiable as well. Let σ be a model of W^{ill}: there exists surely a variable r_i^c such that $\sigma(r_i^c) = \bot$; without loss of generality, let's suppose it's the only variable set to false among all the boolean variables. Therefore, we have that the body of Eq. (9.2) is true for these particular c and i. We can rewrite it (in a more explicit form) as follows:

$$\begin{aligned}
\exists t_{ill}^c \exists p_{ill}^c \forall\overline{t}\forall\overline{q}\forall\overline{p}\Big(&\big((t_{ill}^c < \overline{t}) \vee (t_{ill}^c = \overline{t} \wedge p_{ill}^c \leq \overline{q})\big)\wedge\\
&\big((\overline{t} < t_{i-1}^c + u_{i-1}^c) \vee (\overline{t} = t_{i-1}^c + u_{i-1}^c \wedge \overline{q} \leq \overline{p})\big) \rightarrow\\
&\text{VIOLATION}(\overline{t},\overline{q})\Big)
\end{aligned}$$

Since σ satisfies this formula, there exists a value $val(t_{ill}^c)$ for t_{ill}^c and a value $val(p_{ill}^c)$ such that the model $\sigma' \coloneqq \sigma[t_{ill}^c \mapsto val(t_{ill}^c), p_{ill}^c \mapsto val(p_{ill}^c)]$ satisfies the following formula:

$$\begin{aligned}
\forall\overline{t}\forall\overline{q}\forall\overline{p}\Big(&\big((t_{ill}^c < \overline{t}) \vee (t_{ill}^c = \overline{t} \wedge p_{ill}^c \leq \overline{q})\big)\wedge\\
&\big((\overline{t} < t_{i-1}^c + u_{i-1}^c) \vee (\overline{t} = t_{i-1}^c + u_{i-1}^c \wedge \overline{q} \leq \overline{p})\big) \rightarrow\\
&\text{VIOLATION}(\overline{t},\overline{q})\Big)
\end{aligned}$$

Now the idea is to remove all the universal quantifiers of this formula. We first focus on $\forall\overline{p}$; since the righthand part of the implication does not contain any free occurrence of $\overline{p}$, we can push the quantification

on the lefthand part; moreover, since the top-level operator is an implication, $\forall\overline{p}$ becomes $\exists\overline{p}$, obtaining:

$$\forall\bar{t}\forall\overline{q}\Big(\exists\overline{p}\Big(\big((t^c_{ill} < \bar{t}) \vee (t^c_{ill} = \bar{t} \wedge p^c_{ill} \leq \overline{q})\big) \wedge \big((\bar{t} < t^c_{i-1} + u^c_{i-1}) \vee (\bar{t} = t^c_{i-1} + u^c_{i-1} \wedge \overline{q} \leq \overline{p})\big)\Big) \rightarrow \text{VIOLATION}(\bar{t}, \overline{q})\Big)$$

Moreover, since there's only one theory atom containing a free occurrence of $\overline{p}$, we can push $\exists\overline{p}$ only in front of it, obtaining $\exists\overline{p}(\overline{q} \leq \overline{p})$; given that this is a true formula, we have that:

$$\forall\bar{t}\forall\overline{q}\Big(\big((t^c_{ill} < \bar{t}) \vee (t^c_{ill} = \bar{t} \wedge p^c_{ill} \leq \overline{q})\big) \wedge \big((\bar{t} \leq t^c_{i-1} + u^c_{i-1})\big)\Big) \rightarrow \text{VIOLATION}(\bar{t}, \overline{q})\Big)$$

Now we want to remove the $\forall\overline{q}$ quantification. Let $val(\overline{q})$ be any rational number strictly greater than the maximum among the values assigned by σ' to all the (real-values) free variables in the previous formula. Let $\sigma'' \coloneqq \sigma'[\overline{q} \mapsto val(\overline{q})]$. Obviously, it holds that σ'' satisfies the following formula:

$$\forall\bar{t}\Big(\big((t^c_{ill} < \bar{t}) \vee (t^c_{ill} = \bar{t} \wedge p^c_{ill} \leq \overline{q})\big) \wedge \big((\bar{t} \leq t^c_{i-1} + u^c_{i-1})\big)\Big) \rightarrow \text{VIOLATION}(\bar{t}, \overline{q})\Big)$$

Moreover, given that by definition $p^c_{ill} \leq \overline{q}$ is true, σ'' satisfies the following formula as well:

$$\forall\bar{t}\Big((t^c_{ill} \leq \bar{t} \leq t^c_{i-1} + u^c_{i-1}) \rightarrow \text{VIOLATION}(\bar{t}, \overline{q})\Big)$$

Let $val(\bar{t}) \coloneqq \sigma''(t^c_{i-1} + u^c_{i-1})$ and let $\sigma''' \coloneqq \sigma''[\bar{t} \mapsto val(\bar{t})]$. We have

that $\sigma''' \models \text{VIOLATION}(\overline{t}, \overline{q})$, *i.e.*:

$$
\begin{aligned}
&\neg\phi_i^c[(d,j) \mapsto (r_j^d \wedge \\
&\qquad ((t_j^d < \overline{t}) \vee (t_j^d = \overline{t} \wedge p_j^d < \overline{q})) \vee \\
&\qquad ((\overline{t} < t_{j+1}^d) \vee (\overline{t} = t_{j+1}^d \wedge \overline{q} \leq p_{j+1}^d)))] \vee \\
&\neg\psi_i^c[(d,j) \mapsto (r_j^d \wedge ((t_j^d < \overline{t}) \vee (t_j^d = \overline{t} \wedge p_j^d < \overline{q})))] \vee \\
&\neg\forall\widetilde{t}\forall\widetilde{q}\big(\qquad ((t_{i-1}^c < \widetilde{t}) \vee (t_{i-1}^c = \widetilde{t} \wedge p_{i-1}^c \leq \widetilde{q})) \wedge \\
&\qquad ((\widetilde{t} < \overline{t}) \vee (\widetilde{t} = \overline{t} \wedge \widetilde{q} \leq \overline{q})) \rightarrow \\
&\qquad \psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge ((t_j^d < \widetilde{t}) \vee (t_j^d = \widetilde{t} \wedge p_j^d < \widetilde{q})) \wedge \\
&\qquad ((\widetilde{t} < t_{j+1}^d) \vee (\widetilde{t} = t_{j+1}^d \wedge \widetilde{q} \leq p_{j+1}^d)))]\big)
\end{aligned}
$$

By definition of model σ''', $p_j^d < \overline{q}$ is true and $\overline{q} \leq p_{j+1}^d$ is false. Therefore σ''' satisfies

$$
\begin{aligned}
&\neg\phi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \overline{t} < t_{j+1}^d] \vee \\
&\neg\psi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \overline{t}] \vee \\
&\neg\forall\widetilde{t}\forall\widetilde{q}\big(\qquad ((t_{i-1}^c < \widetilde{t}) \vee (t_{i-1}^c = \widetilde{t} \wedge p_{i-1}^c \leq \widetilde{q})) \wedge \\
&\qquad ((\widetilde{t} < \overline{t}) \vee (\widetilde{t} = \overline{t} \wedge \widetilde{q} \leq \overline{q})) \rightarrow \\
&\qquad \psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge ((t_j^d < \widetilde{t}) \vee (t_j^d = \widetilde{t} \wedge p_j^d < \widetilde{q})) \wedge \\
&\qquad ((\widetilde{t} < t_{j+1}^d) \vee (\widetilde{t} = t_{j+1}^d \wedge \widetilde{q} \leq p_{j+1}^d)))]\big)
\end{aligned}
$$

Let $val(\widetilde{q}) \coloneqq \sigma'''(\overline{q})$ and let $\psi \coloneqq \sigma'''[\widetilde{q} \mapsto val(\widetilde{q})]$. Obviously, ψ satisfies the following formula:

$$
\begin{aligned}
&\neg\phi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \overline{t} < t_{j+1}^d] \vee \\
&\neg\psi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \overline{t}] \vee \\
&\neg\forall\widetilde{t}\big(\qquad ((t_{i-1}^c < \widetilde{t}) \vee (t_{i-1}^c = \widetilde{t} \wedge p_{i-1}^c \leq \widetilde{q})) \wedge \\
&\qquad ((\widetilde{t} < \overline{t}) \vee (\widetilde{t} = \overline{t} \wedge \widetilde{q} \leq \overline{q})) \rightarrow \\
&\qquad \psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge ((t_j^d < \widetilde{t}) \vee (t_j^d = \widetilde{t} \wedge p_j^d < \widetilde{q})) \wedge \\
&\qquad ((\widetilde{t} < t_{j+1}^d) \vee (\widetilde{t} = t_{j+1}^d \wedge \widetilde{q} \leq p_{j+1}^d)))]\big)
\end{aligned}
$$

By definition of ψ, we have that both $p_{i-1}^c \leq \widetilde{q}$ and $\widetilde{q} \leq \overline{q}$ are true.

Therefore, ψ satisfies:

$$\begin{aligned}
&\neg\phi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \bar{t} < t_{j+1}^d]\vee\\
&\neg\psi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \bar{t}]\vee\\
&\neg\forall\widetilde{t}\big(\quad (t_{i-1}^c \leq \widetilde{t} \leq \bar{t}) \rightarrow\\
&\qquad \psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge ((t_j^d < \widetilde{t}) \vee (t_j^d = \widetilde{t} \wedge p_j^d < \widetilde{q}))\wedge\\
&\qquad ((\widetilde{t} < t_{j+1}^d) \vee (\widetilde{t} = t_{j+1}^d \wedge \widetilde{q} \leq p_{j+1}^d)))]\big)
\end{aligned}$$

For the same reason, we have that $p_j^d < \widetilde{q}$ is true and $\widetilde{q} \leq p_{j+1}^d$ is false, having that the following formula is satisfied by ψ:

$$\begin{aligned}
&\neg\phi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \bar{t} < t_{j+1}^d]\vee\\
&\neg\psi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \bar{t}]\vee\\
&\neg\forall\widetilde{t}\big(t_{i-1}^c \leq \widetilde{t} \leq \bar{t} \rightarrow \psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \widetilde{t} < t_{j+1}^d)]\big)
\end{aligned}$$

Finally, since by definition $\psi(\bar{t}) = \psi(t_{i-1}^c) + u_{i-1}^c$, we have that ψ satisfies Eq. (A.3) and also:

$$\begin{aligned}
&\neg\phi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq t_i^c < t_{j+1}^d]\vee\\
&\neg\psi_i^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq t_i^c]\vee\\
&\exists\widetilde{t}\big(t_{i-1}^c \leq \widetilde{t} \leq t_i^c \wedge \neg\psi_{i-1}^c[(d,j) \mapsto (r_j^d \wedge t_j^d \leq \widetilde{t} < t_{j+1}^d)]\big)
\end{aligned}$$

that is WEAKVIOL(t_i^c). Therefore, $\psi \models \mathrm{W}_{\mathrm{fin}}^{\mathrm{ill}}$.

$\Leftarrow$**.** Following backward all the steps of direction $\Rightarrow$, it is possible to prove also that if $\mathrm{W}_{\mathrm{fin}}^{\mathrm{ill}}$ is satisfiable, then $\mathrm{W}^{\mathrm{ill}}$ is satisfiable as well. In particular, given a model ψ of $\mathrm{W}_{\mathrm{fin}}^{\mathrm{ill}}$, following the previous steps we can build the model σ (see the opposite direction) such that $\sigma \models \mathrm{W}^{\mathrm{ill}}$. □ □

A.4 Quantifier-free encoding for Convex Dependencies (Chapter 9)

We show how, in case of convex dependencies we can drop the universal quantifier from Eq. (9.1) and consequently also the existential quantifier from Eq. (9.3).

Definition 68 (Convex Dependencies)**.** *A dependency (*i.e.*, an SMT(DL) formula)* $\Gamma(s)$ *is said to be* convex *iff, given two time points* s_i *and* s_j, $\Gamma(s_i)$ *and* $\Gamma(s_j)$ *holds iff* $\Gamma(\overline{s})$ *holds, for all* $s_i \preceq \overline{s} \preceq s_j$.

Proposition 22 (Conjunctive Dependencies)**.** *If* $\Gamma(s)$ *is a conjunctive dependency,* i.e.*, a conjunction of theory atoms of SMT(DL), then* $\Gamma(s)$ *is convex.*

We call $\widehat{\mathrm{W}}$ the encoding equal to W except that, if ψ^c_{i-1} is convex, Eq. (9.1) is replaced by:

$$r^c_i \rightarrow \psi^c_{i-1}[(d,j) \mapsto (r^d_j \wedge s^c_i \preceq s^d_{j+1})] \tag{A.8}$$

and Eq. (9.3) is replaced by:

$$\begin{aligned} \mathrm{VIOLATION}(\overline{s}) \quad := \quad & \neg\phi^c_i[(d,j) \mapsto (r^d_j \wedge s^d_j \prec \overline{s} \preceq s^d_{j+1})] \vee && \text{(A.9)} \\ & \neg\psi^c_i[(d,j) \mapsto (r^d_j \wedge s^d_j \prec \overline{s})] \vee && \text{(A.10)} \\ & \neg\psi^c_{i-1}[(d,j) \mapsto (r^d_j \wedge s^c_i \preceq s^d_{j+1})] && \text{(A.11)} \end{aligned}$$

We can state the following proposition:

Proposition 23 (Equisatisfiability between W and $\widehat{\mathrm{W}}$)**.** W *and* $\widehat{\mathrm{W}}$ *are equisatisfiable.*

Proof. Simple, it is sufficient to apply Definition 68. □

Note that, given what we proved in Section A.3, for a system with only finite bounds and only convex dependencies, we are able to generate a quantifier-free encoding, that we call $\widehat{\mathrm{W}_{\mathrm{fin}}}$: this has a big impact on the performances of our tool.

Quantifier-free encoding for non-convex dependencies. The encoding proposed in Section 9.3.2 for the general case features a universal quantifier in Eq. (9.1), in order to deal with non-convex state dependencies. Despite being more intuitive, it is not necessary, in that we can substitute Eqs. (9.1) and (9.6) with the following contraints, respectively:

$$\begin{aligned} r^c_i &\rightarrow \mathrm{EXITVIOL} \\ (r^c_{i-1} \wedge \neg r^c_i) &\rightarrow \neg\,\mathrm{EXITVIOL} \end{aligned} \tag{A.12}$$

where:

$$\mathrm{EXITVIOL} := \bigwedge_{\substack{1 \le d \le n \\ 1 \le j \le |d|}} \left(t^c_{i-1} \le t^d_j \le t^c_i \rightarrow \psi^c_{i-1}[(e,k) \mapsto (r^e_k \wedge t^e_k < t^d_j \le t^e_{k+1})]\right) \tag{A.13}$$

It is worth noting that for the big-and in Eq. (A.13) is sufficient to range only among the pairs (component,phase) on which phase $C.i$ depends on (including itself); this in general brings to a significanly smaller encoding.