UNIVERSITÀ DI TRENTO
Department of Information Engineering and Computer Science

Doctoral Programme in
Information Engineering and Computer Science

Compositional Reasoning and Model Checking of Asynchronous Systems on Variations of LTL

Alberto Bombardelli

Advisor
Stefano Tonetta
Fondazione Bruno Kessler

October 2025

Abstract

Formal verification is a method that offers mathematically rigorous techniques for verifying system correctness. While it has become a foundational tool in the design and verification of critical systems, its application to modern, real-world systems is increasingly hindered by a combination of fundamental challenges: scalability, expressiveness and system complexity. Scalability issues arise from the explosion of the state space caused by concurrency, interleaving, and system size. Expressiveness is another key difficulty: specification languages must be able to capture arithmetic constraints, real-time specifications, and hyperproperties–i.e., properties relating multiple executions, such as security or diagnosability. Finally, system features such as non-deterministic scheduling, real-time constraints, or shared-memory further complicate modeling and aggravate the other two challenges.

This thesis addresses these three problems as follows.

First, in the context of Verification Modulo Theories, we propose an ad-hoc invariant-based algorithm for verifying a class of LTL modulo theory properties by leveraging the notion of relative safety. This contributes to a more scalable verification strategy for non-trivial properties beyond traditional safety. We also present an open framework supporting both finite and infinite trace semantics.

Second, we develop a compositional framework for asynchronous systems where components are specified in LTL modulo theory. The semantics naturally account for component termination or failure, improving modular reasoning. This approach is extended to real-time systems through Metric Temporal Logic with Skewed Clocks (MTLSK), enabling the verification of time-sensitive distributed behaviours.

Third, we tackle hyperproperty verification via two distinct contributions: a k-induction based method for verifying $\forall\exists$ hyperproperties, and the introduction of an asynchronous temporal logic (GHyperLTLS+C). We identify a decidable, non-prenex fragment expressive enough for diagnosability, and design model checking procedures for the prenex fragment over finite traces.

Together, these contributions advance symbolic verification by improving its scalability, expressiveness, and support for complex system models.

Keywords

Symbolic Model Checking, Linear Temporal Logic,HyperLTL, Asynchronous Composition, Metric Temporal Logic

Contents

List of Tables

List of Figures

Acknowledgements

The list of people I have to thank for their support in this journey is quite long; I hope I don't forget someone.

I would not have even considered the idea of starting a PhD if it weren't for my supervisor, Stefano Tonetta. He introduced me to the fascinating world of model checking and, during these years, has spent his time and energy mentoring me. He gave me the tools to grow as a researcher and also as a person. Stefano has always been available, patient, and supportive. I am grateful to him for that.

I want to thank César Sánchez. It was a pleasure to work with him on hyperproperties; moreover, I was very happy during my visit to IMDEA Software with him and his group.

I would also like to thank the other researchers I worked with during these years: Laura Bozzelli, Marco Bozzano, Alberto Griggio, and Alessandro Cimatti.

I am grateful to Prof. Borzoo Bonakdarpour, Prof. Ezio Bartocci, and Prof. Marius Bozga for reviewing this thesis and for their helpful comments and suggestions.

During these years, I had the opportunity to work with many great people at FBK. My colleagues have always been close friends to me; we had a lot of fun together, and I think it is difficult to find this kind of environment in a workplace. Since there are many, I will name only a few here: Gianni Zampedri, Alberto Bonizzi, Srajan Goyal, Giulia Sindoni, and Guillermo Gomez Arnedo. A special thanks go to my flatmate, friend, and colleague, Daniele Giuliani for the fun time together with aperitifs, dinners, and chats.

Sharing the PhD journey with Anna Becchi made this experience much better. She helped me a lot, and we had a lot of fun over these years.

I believe that without the support of good friends like Alessandro, these years would have been much harder. I really want to thank him for being there for me; our phone calls kept me mentally sane during very difficult moments of this journey.

I am grateful to my family, who supported me throughout this and many other steps of my life. I want to thank my father, my mother, my sister, Cinzia, Daniele, and

Mirco.

Finally, I would also like to thank my guinea pigs (Calliope, Melpomene, Urania, and Talia) and my family dog (Capsula) for their fundamental emotional support. Although they are no longer here with me, I think about them every day.

Chapter 1

Introduction

1.1 Context and Formal Problem

Modern society increasingly relies on complex interconnected systems, including autonomous vehicles, aerospace control software, and distributed infrastructures. These systems bring significant benefits–enhancing safety, optimizing performance, and improving quality of life. However, when such systems are designed or implemented incorrectly, the consequences can be severe. In safety-critical domains such as automotive, aerospace, and avionics, even minor faults can lead to catastrophic outcomes. Similarly, failures in aerospace systems or communication protocols can compromise safety, security, and trust. Several real-world incidents have demonstrated how subtle design errors can result in critical failures in deployed systems. These cases emphasize the need for rigorous methods to ensure that systems behave as intended, especially under all possible operating conditions.

Traditionally, software testing has been the standard approach to increase confidence in system correctness. Testing involves executing the system with selected inputs in an attempt to uncover bugs. While valuable, testing is fundamentally limited: it can show that errors exist but cannot guarantee their absence. It explores only a finite subset of behaviours, which is insufficient for systems with concurrency, nondeterminism, or unbounded inputs. To overcome these limitations, the field of Formal Methods offers mathematically rigorous techniques for verifying system correctness. Among these, various approaches exist–such as theorem proving, model checking and runtime verification. In this thesis, we focus on Model Checking, a fully automated method that has seen widespread adoption in both academia and industry.

Model checking provides an automatic, algorithmic solution to the problem of ver-

ifying that a system satisfies a given specification. From a high-level perspective, a model checker takes as input a formal model of a system–such as a symbolic transition system or a timed automaton–and a property expressed in a temporal logic, such as a Linear Temporal Logics (LTL) formula or an invariant. The task is to determine whether the system satisfies the property; if not, the model checker typically returns a counterexample trace illustrating the violation. While model checking has become a foundational tool in the design and verification of critical systems, its application to modern, real-world systems is increasingly hindered by a combination of fundamental challenges. These challenges stem not only from the complexity of the systems and the expressiveness of the properties but also from how these aspects interact and compound one another–making verification both theoretically and practically more difficult.

The first major obstacle is scalability. As systems grow in size, the number of possible behaviours to explore increases exponentially. This state explosion is especially severe in concurrent or asynchronous systems, where independently executing components produce a vast space of interleaved traces. Symbolic methods such as BDDs or SAT/SMT-based encodings help mitigate this growth, but are often insufficient in practice. The problem is compounded when properties are expressed in rich specification languages—particularly those involving multiple traces (e.g., hyperproperties) or interpreted over background theories (e.g., arithmetic, real-time constraints). In many of these cases, the verification task becomes undecidable, making full automation difficult. Furthermore, verifying properties over infinite executions, such as liveness or fairness, is inherently harder than checking safety properties, which can typically be analyzed using finite trace semantics. This distinction introduces another layer of complexity that affects both algorithm design and tool performance.

A second, tightly related challenge is that of *expressiveness*. Many correctness requirements in modern systems go beyond the capabilities of classical temporal logics such as LTL. A dimension of expressiveness arises when temporal logics are extended with *first-order features* or *theories* such as Linear Integer Arithmetic, real-time constraints, or continuous dynamics. These extensions allow the specification of more realistic, data-rich properties. Another orthogonal dimension comes from *security properties* and *diagnosability properties*, which relate multiple executions and require reasoning about *hyperproperties*. Formalizing such requirements demands logics such as HyperLTL, which quantify over multiple traces. Nevertheless, the increase in expressiveness typically results in higher computational complexity and heavier solver dependencies, making the verification process more fragile and less scalable.

The third major difficulty arises from the inherent *semantic complexity* of modern

systems. Features such as *nondeterministic scheduling*, *shared memory*, and *real-time constraints* introduce subtle interactions and timing dependencies that are difficult to capture and reason about. A particularly critical aspect is *asynchronicity*, where components evolve at different speeds, without synchronized clocks or fixed schedules. Asynchronicity amplifies the number of possible executions and weakens assumptions about time alignment across components. This has severe consequences: many hyperproperties that are decidable in a synchronous setting become *undecidable* when interpreted asynchronously. Moreover, asynchronicity significantly complicates the application of *compositional reasoning* and abstraction techniques, both of which are crucial for managing the scalability problem.

It is important to emphasize that these challenges are not isolated. The need for expressive logics (Challenge 2) often increases the difficulty of verification from both computational and representational standpoints, aggravating the state space explosion identified in Challenge 1. Likewise, system-level semantic features such as asynchronicity (Challenge 3) undermine the tractability of scalable verification and interact poorly with expressive specifications–leading, for example, to undecidability results that do not arise in simpler, synchronous models.

To summarize, model checking in modern systems must contend with:

1. *Scalability* due to the explosion of the state space from interleaving, system size, and the complexity of expressive formalisms;

2. *Expressibility* required to capture data-aware specifications, properties involving arithmetic or continuous time, and hyperproperties;

3. *System complexity* such as asynchronicity, shared memory, and real-time behaviour, which not only complicate modeling but exacerbate the other two challenges.

In this thesis, we explore these challenges and contribute modest advancements in each of these areas, aiming to improve our understanding and provide practical techniques for addressing some of the limitations currently faced in the verification of complex systems.

1.2 Contributions

In this section, we provide a high-level and informal overview of the contributions of this thesis. Detailed technical results, proofs, and formalizations are deferred to the

individual chapters. The contributions lie within the domain of model checking for Linear Temporal Logic (LTL) and its various extensions. We outline how the work presented addresses the main challenges in model checking previously identified.

Although the contributions are organized separately, they are conceptually interconnected. Many of the techniques and insights developed in one context contribute to solving problems in others. Likewise, the problems themselves are often intertwined–progress on one front often facilitates advances on another. This dissertation addresses these fundamental challenges by advancing symbolic verification techniques across three interconnected dimensions. We aim to enhance scalability through compositional reasoning and specialized algorithms, while simultaneously expanding expressiveness to cover complex data, asynchronous behaviours, and relational properties.

The thesis contributions are structured into three parts, each contributing to our overall objective:

- Part II: *Verifying Trace Properties.*

  This part lays the foundation by extending classical Linear Temporal Logic (LTL) model checking to its variant modulo background theories, denoted as LTL($\mathcal{T}$). We develop a symbolic automata-theoretic framework that uniformly supports both infinite and finite trace semantics. In addition, we propose a specialized invariant-based algorithm capable of verifying properties that go beyond traditional safety.

  While the conceptual separation between the automata-theoretic structure and the underlying theory is not novel, our contribution lies in providing a fully formalized, open framework–complete with theoretical guarantees–and an open-source library implementing these ideas with a general and extensible format.

  Furthermore, we unify the treatment of safety properties and variants of LTL with finite-trace semantics (LTL$_f$ and Truncated Weak LTL), presenting a common framework for both. Building on safety verification techniques, we extend the approach to cover a larger class of non-safety properties by leveraging the notion of *relative safety*. This allows us to reduce the verification of certain non-safety specifications to invariant checking, under specific assumptions, which in turn enhances scalability as shown in a experimental evaluation against state-of-the-art algorithms.

  Overall, this part contributes a flexible and extensible foundation for verifying both LTL($\mathcal{T}$) and LTL$_f$($\mathcal{T}$) specifications.

- Part III: *Asynchronously Composing Trace Properties.*

  Building on the foundations of LTL($\mathcal{T}$), this part addresses the scalability challenges posed by concurrent and distributed systems through compositional reasoning. In Chapter 5, we introduce a general framework for the asynchronous composition of LTL properties, where local specifications are interpreted over local traces, taking into account interruptions, scheduling constraints, and data communication.

  A central element of our approach is the use of a *truncated weak semantics*, which supports both finite and infinite executions of individual components. This makes the framework well-suited to practical settings, such as safety analysis in systems with early termination or preemption. By treating asynchrony as decoupled from the local specifications, our compositional method not only improves scalability but also helps manage the increased complexity arising from system concurrency and interaction. This contribution has been implemented in a state-of-the-art compositional verification tool.

  In Chapter 6, we extend this compositional approach to the real-time domain, focusing on Distributed Real-Time Systems (DRTS). Here, we introduce methods to reason about systems with skewed and resettable clocks by developing a novel logic, *Metric Temporal Logic with Skewed and Resettable Clocks* (MTLSK). We also define an encoding of MTLSK into LTL modulo theories, enabling verification via symbolic techniques.

  Besides naturally improving scalability through compositionality, the introduction of this type of composition and the definition of MTLSK also enhance the *expressiveness* of the formalism, enabling the specification of more nuanced timing and coordination properties than those previously supported.

- Part IV: *Hyperproperties for Comparing Traces.* This part delves into the verification of hyperproperties, which are essential for properties relating multiple execution traces (e.g., security, diagnosability, robustness). In Chapter 7, we present a novel approach to verify $\forall\exists$-hyperproperties by adapting induction and k-induction techniques, introducing concepts such as hyper-(k)-inductive invariants and refinement strategies to strengthen properties. This approach is compared with existing techniques and shows promising results on a consistent class of benchmarks.

  In Chapter 8, we introduce Generalized HyperLTL with Stuttering and Contexts

	Ch. 4	Ch. 5	Ch. 6	Ch. 7	Ch. 8
Scalability	✓	✓	✓	✓	×
Expressibility	×	×	✓	×	✓
System Complexity	×	✓	✓	×	✓

Table 1.1: Overview of the research challenges *directly* addressed in this thesis and their corresponding chapters. A ✓ indicates that the chapter directly contributes to the respective challenge, while × denotes it is not directly covered.

(GHyperLTL$_{\mathsf{S+C}}$), a rich asynchronous temporal logic that unifies and extends prior asynchronous hyperlogics. GHyperLTL$_{\mathsf{S+C}}$ supports both past temporal modalities and non-prenex quantification, enabling the specification of complex properties such as diagnosability in distributed and asynchronous systems. We identify a meaningful decidable fragment of the logic capable of expressing several practically relevant properties.

To address finite-trace scenarios, we define a simplified variant called SC-HyperLTL, tailored to deal with the challenge of combining finite semantics with multi-trace asynchronicity. For this fragment, we develop model-checking procedures based on a variation of self-composition for the quantifier-alternation-free case, and an adaptation of bounded model checking techniques when quantifier alternation is present. These techniques are implemented in a proof-of-concept tool, showing the feasibility of the approach.

In summary, this dissertation presents a comprehensive and integrated approach to address the inherent challenges of scalability and expressiveness in formal verification. By combining extensions of classical temporal logics with advanced compositional and inductive techniques, tailored for asynchronous and multi-trace reasoning, we contribute towards a more unified, efficient, and theory-aware methodology for verifying complex modern systems. To provide a high-level overview, Table 1.1 maps each part of the thesis to the specific challenges outlined in the previous section, indicating which contributions directly address which aspects. For a more detailed breakdown of the technical topics covered in each chapter, we refer the reader to Table 1.2.

Relation Between Contributions While each part of the thesis addresses a distinct problem, the contributions are far from isolated. Each part is at least partially

[1]Only the safety fragment.
[2]Only theoretical contribution.
[3]Primarly for the distributed version of the logics. Indirectly, also for MTL.

		Trace Properties		**Hyperproperties**	
		Sync MC	Async Comp.	Sync MC	Async MC
Discrete	Fin. Trace	Ch. 4	Ch. 5	Ch. 7	Ch. 8
	Inf. Trace	Ch. 4	Ch. 5	Ch. 7 ([1])	Ch. 8([2])
Timed	Fin. Trace	×	×	×	×
	Inf. Trace	Ch. 6 ([3])	Ch. 6	×	×

Table 1.2: Overview of the model checking configurations considered in this thesis. Each entry either points to the chapter discussing the corresponding case, or is marked with × if the subcase is not within the scope of this work.

connected to a subset of the others, both in terms of the challenges tackled and the methods developed. After presenting the individual contributions, we highlight a few of these key interconnections.

Notably, the symbolic techniques and automata constructions developed for verifying LTL modulo theories (LTL($\mathcal{T}$)) are also employed in the compositional reasoning framework, where they are used to verify local proof obligations. Similarly, these constructions reappear in the verification of hyperproperties, particularly in the quantifier-alternation free fragment of HyperLTL, where verification can be reduced to checking a self-composed system using LTL($\mathcal{T}$)-based methods.

There is also a meaningful connection between the treatment of relative safety in LTL model checking and the compositional verification framework. The formulae supported by the relative safety extension often take the form of implications, which are a natural fit for compositional verification techniques. This enables the scalable verification of properties that, while not safety in the strict sense, can be handled effectively through the same mechanisms.

Finally, the asynchronous composition framework developed for trace properties has proven unexpectedly useful in the verification of asynchronous hyperproperties. Specifically, the verification of finite trace asynchronous hyperproperties in the quantifier-alternation free case can be reduced to a variant of our asynchronous composition problem, thus broadening the applicability of the developed techniques.

Taken together, the detailed relations above can be viewed within a larger verification landscape, where foundational solving and invariant techniques support progressively more expressive reasoning layers. Figure 1.1 provides an overview of this structure, illustrating both the technical dependencies and the conceptual connections among the different parts of the thesis.

1.3 Structure of the Thesis

The remainder of the thesis is divided in five parts.

In Part I, we provide background notions and provide an overview of the state-of-the-art in formal verification. In Chapter 2, we introduce background notions used in the core parts of this dissertation; in Chapter 3, we discuss developments in formal verification relevant to this thesis.

In Part II, we discuss symbolic model checking of trace properties. This part is composed only by Chapter 4, which discuss model checking of fragments of LTL modulo theory.

In Part III, we discuss asynchronous composition of trace properties. In Chapter 5, we introduce a compositional technique to verify LTL modulo theory properties in the asynchronous setting; in Chapter 6, we extend our compositional framework considering timed logics in the distributed real time setting.

In Part IV, we present our contributions in the context of hyperproperties. In Chapter 7, we present a novel technique to verify $\forall\exists$ temporal safety hyperproperties with a novel adaptation of k-induction for hyperproperties; in Chapter 8, we discuss a unified logic combining Stuttering HyperLTL and Context HyperLTL and its model checking over infinite and finite traces.

Finally, in Part V, we draw conclusions of the thesis. This part is composed only by Chapter 9, which discuss the contribution and possible future extensions of this work.

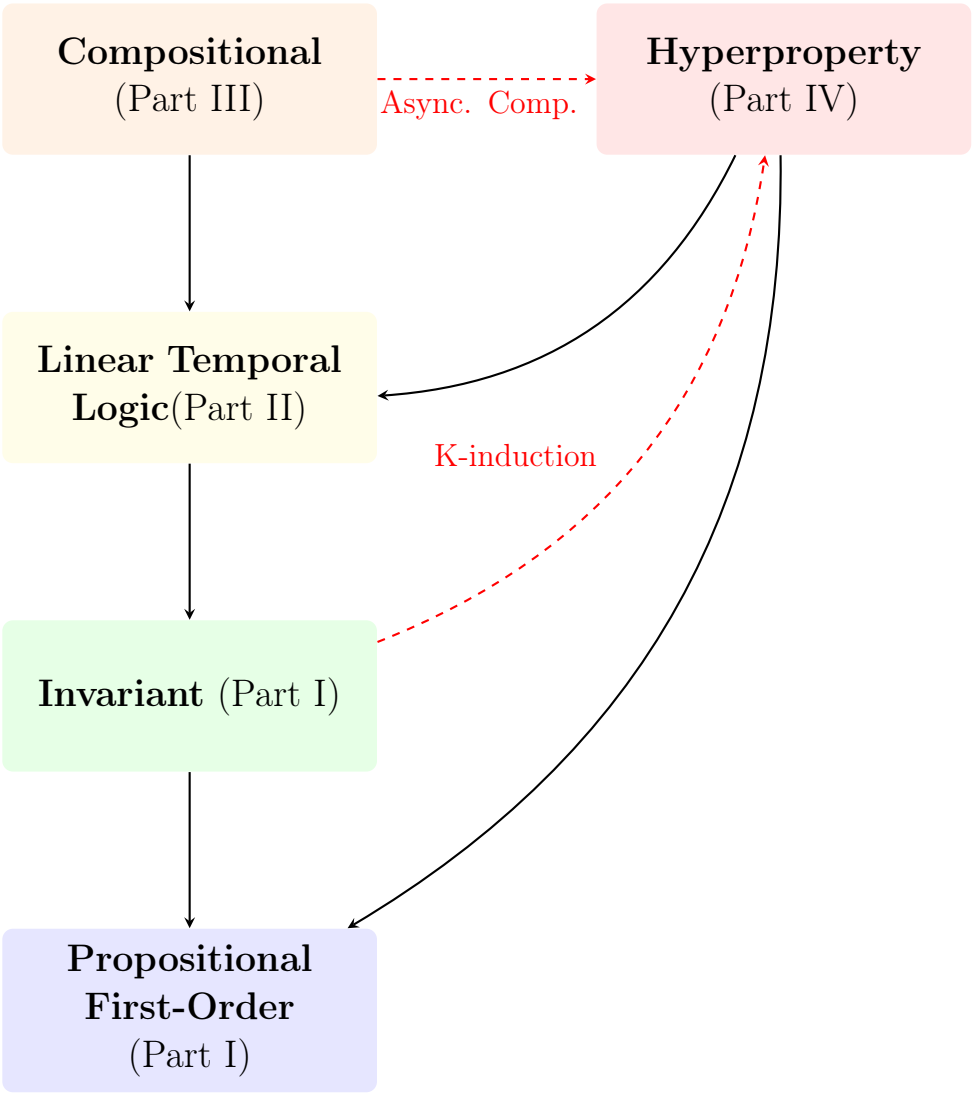


Figure 1.1: Each colored node represents a main conceptual or methodological layer, from propositional and first-order reasoning at the base, through invariant and temporal reasoning, up to compositional and hyperproperty verification. Solid arrows represent technical dependencies, meaning that each layer builds directly on the verification mechanisms of the one below it (for example, compositional reasoning relies on LTL model checking, which in turn depends on invariant generation and SAT solving). Dashed red arrows represent conceptual influences, showing where ideas or techniques developed in one layer have been adapted and extended in another (for instance, k-inductive reasoning from invariant verification is generalized to the hyperproperty setting). Together, these relations illustrate how the thesis evolves from foundational reasoning techniques to expressive logics and verification frameworks for asynchronous and relational systems.

Part I

Background and State of the Art

Chapter 2

Background

This chapter presents the technical background necessary to understand the content of the thesis. It introduces the key concepts, formalisms, and verification techniques that are used throughout the remainder of this dissertation.

We begin in Section 2.1 with an introduction to *Satisfiability Modulo Theories* (SMT), including the basic notions, the distinction between satisfiability and validity, and the role of interpolation and solvers. Section 2.2 then introduces the notions of *state* and *trace*, which serve as the basis for describing system executions. Section 2.3 discusses *trace properties*, focusing in particular on the distinction between safety and liveness. Section 2.4 defines *symbolic transition systems*, which offer a compact representation for reasoning symbolically about system behaviour. Section 2.5 presents *Linear Temporal Logic* (LTL), including its syntax, semantics, safety fragments, and automata-theoretic techniques. Section 2.6 surveys key SAT/SMT-based model checking methods such as invariant checking, bounded model checking, and k-induction, as well as techniques for handling liveness. Section 2.7 introduces *timed systems* and *Metric Temporal Logic* (MTL), formalizing how time is treated in traces and specifications. Finally, Section 2.8 covers *HyperLTL*, a temporal logic that expresses *hyperproperties*—relations over multiple traces that are essential for capturing security, diagnosability, and similar properties. Together, these sections provide the necessary background for the thesis.

2.1 Satisfiability Modulo Theories (SMT)

2.1.1 SMT Basics

Satisfiability Modulo Theories (SMT) [18, 17] is the decision problem of determining whether a logical formula is satisfiable with respect to a background theory. It gen-

eralizes the Boolean satisfiability problem (SAT) by allowing predicates and functions interpreted over richer domains, such as integers, reals, bit-vectors, or arrays.

SMT formulae are typically expressed in *first-order logic (FOL)* with theories. They are constructed from logical connectives (e.g., $\wedge$, $\vee$, $\neg$, $\rightarrow$), a set of variables V, and a signature Σ that includes function and predicate symbols.

A *theory* $\mathcal{T}$ consists of a signature $\Sigma_{\mathcal{T}}$ and a class of structures $\mathcal{M}_{\mathcal{T}}$ that define interpretations for the symbols in $\Sigma_{\mathcal{T}}$. A formula ϕ is said to be *satisfiable modulo* $\mathcal{T}$ if there exists a model (i.e., a structure and assignment) in $\mathcal{M}_{\mathcal{T}}$ that satisfies ϕ.

We write $\langle \tau, \mathcal{M} \rangle \models \phi$ to denote that the formula ϕ is satisfied by the assignment τ under the structure $\mathcal{M}$.

If τ_1 and τ_2 are assignments to two disjoint variable sets V_1 and V_2, respectively, we write $\tau_1 \cdot \tau_2$ for their combination, which maps each $v_i \in V_i$ to $\tau_i(v_i)$.

Common SMT Theories Two commonly used SMT theories in verification are:

Linear Integer Arithmetic (LIA) that includes integer variables and allows linear constraints over integers. For example:

$$\phi_{\text{LIA}} \equiv (x + 2y \leq 5) \wedge (x \geq 0) \wedge (y \in \mathbb{Z})$$

This formula is satisfiable in LIA; for instance, $x = 1$, $y = 2$ is a satisfying assignment.

Linear Real Arithmetic (LRA) uses real-valued variables with linear constraints over $\mathbb{R}$. For instance:

$$\phi_{\text{LRA}} \equiv (x + y > 3.5) \wedge (2x - y \leq 1)$$

A satisfying assignment is $x = 2.5$, $y = 1.5$.

2.1.2 Satisfiability, Validity, and Interpolation

We assume the reader is familiar with propositional and first-order logic, including the notions of *satisfiability*, *validity*, *unsatisfiability* (UNSAT), and *entailment*. We use the satisfaction symbol $\models$, and overload it depending on the logic in context:

- $\models A$: formula A is valid.
- $A \models B$: formula A entails B, equivalent to $\models A \rightarrow B$.

Craig Interpolation

Given formulae A, B, and I such that $A \wedge B$ is UNSAT, I is a *Craig interpolant* for A and B if:

1. $A \models I$,
2. I refers only to symbols common to both A and B, and
3. $I \wedge B$ is UNSAT.

Example: Let $A = (a \wedge c)$ and $B = (a \rightarrow b) \wedge \neg b$. Then $I = a$ is a Craig interpolant of A and B.

2.1.3 SMT Tooling and Solvers

In this manuscript, we use SAT and SMT solvers as black-box tools for solving verification problems. Over the years, many efficient solvers have been developed. While not exhaustive, the following list includes widely used tools: **Z3** [139], **MathSAT5** [105], **cvc5** [15], **Yices** [146],**PicoSAT** [43] and **MiniSAT** [279].

These solvers support the SMT-LIB standard [16], a unified language for SMT benchmarks and theory definitions. They also offer APIs in various programming languages, enabling easy integration into verification tools and frameworks. Libraries such as **PySMT** [175] provide a common interface for interacting with multiple SMT solvers, abstracting away implementation differences and simplifying prototyping and experimentation.

Quantified Boolean Formulae (QBF) [185] In some cases, we use quantifiers over propositional variables without requiring richer theories. In such cases, *Quantified Boolean Formula (QBF)* are considered. Basically QBF formulae extends propositional formulae allowing quantifiers of Boolean variables. Formally, the formulae are as follows:

$$\phi ::= \exists v.\phi \mid \forall v.\phi \mid \neg\phi \mid \phi \vee \phi \mid v$$

where v is a Boolean variable. Often solvers require QBF formulae to be expressed in prenex normal form i.e. with quantifiers only occurring in the outermost parts of the formula. While QBF is less expressive than SMT with rich theories, it supports nested quantification and is useful for applications such as Bounded Model Checking of hyperproperties [199], planning, games and equivalence checking [273]. This limitation

permit remaining decidable and while still treating relevant problems. Currently, there are a bunch of tools for the verification of QBF formulae such as **QuAbs** [193] and **DepQBF** [233].

2.2 State and Trace

In this manuscript, we will use the terms *path*, *trace* or, *word* to finite or infinite sequence of *states*, which are represented by first-order assignments over set of variables.

Given a structure $\mathcal{M}$ and a set of variables V, a *state* s is an assignment from V to the domain of $\mathcal{M}$. Given a state s and a variable $v \in V$, we denote $s(v)$ as the assignment of v in state s. Moreover, let $\overline{V} \subseteq V$, we denote $\overline{s} \doteq s(\overline{V})$ as the state over the variables $\overline{V}$ s.t. for each $v \in \overline{V}$ $s(v) = \overline{s}(v)$.

A [finite] *trace* is an infinite sequence $\sigma = s_0, s_1, \ldots$ [resp., finite sequence $s_0, s_1, \ldots, s_k$] of states. Given a finite or infinite trace $\sigma = s_0 s_1 \ldots$, we denote by $|\sigma|$ the size of σ. Thus, $|\sigma| = +\infty$ if σ is infinite. We denote by $\sigma(i)$ the $i+1$-th state s_i of the trace and by $\sigma_{\geq i}$ the suffix of σ starting from $\sigma(i)$. Given a finite trace σ_1 and a finite or infinite trace σ_2, we denote by $\sigma_1\sigma_2$ their concatenation. Given an infinite trace σ, we denote by $\mathit{Pref}(\sigma)$ the set of prefixes, i.e., $\mathit{Pref}(\sigma) = \{\sigma_1 \mid \sigma = \sigma_1\sigma_2\}$.

If σ_1 and σ_2 are traces of the same length over two disjoint sets of variables V_1 and V_2, we denote with $\sigma_1 \cdot \sigma_2$ the trace over $V_1 \cup V_2$ such that for all i, $0 \leq i < |\sigma_1|$, $(\sigma_1 \cdot \sigma_2)(i) = \sigma_1(i) \cdot \sigma_2(i)$.

Let $V' \subseteq V$, we define $\sigma' \doteq \sigma(V')$ as the trace over variables in V' such that $\sigma'(i) = \sigma(i)(V')$. Finally, we denote by $\mathcal{L}(V)$ and $\mathcal{L}_{fin}(V)$ the set of all possible respectively infinite and finite traces over the variable set V.

2.3 Trace Property

In this dissertation, we discuss of various type of properties expressed in different formalisms. To give a generic views on the properties of interest, we introduce the notion of *trace property* in their most-basic form.

Given a set of variables V interpretable over a theory $\mathcal{T}$ over some first-order structure $\mathcal{M}$. A trace property P is a set of infinite traces $P \subseteq \mathcal{L}(V)$.

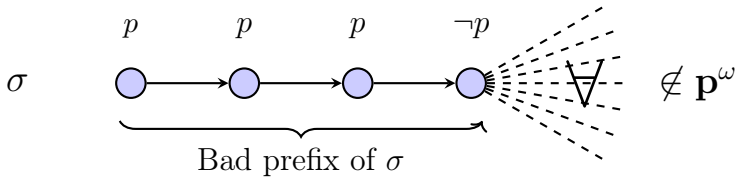


Figure 2.1: Graphical representation of safety property violations with a bad prefix. Any extensions of σ_f cannot satisfy the property stating that p always holds.

2.3.1 Safety Property

Informally, a property is considered as a safety property if it stipulates that nothing bad happens. If something bad happens, it occurs in a finite prefix of the execution; meaning that a finite view of trace can be sufficient to speculate that a property is falsified. A classic example of safety property is the *invariant*, if I stipulate that the Boolean variable b shall always be *true*, any finite trace that assigns in an arbitrary position b to *false* will be sufficient to disprove the property. Figure 2.1 provides a graphical intuition of a safety property. Formally we can define the notion of *safety property* with the following definition.

Definition 2.1 (Safety Property) *Let P be a property over a set of variables V, P is a safety property iff*

$$\textit{for all } \sigma \in \mathcal{L}(V) \textit{s.t. } \sigma \notin P, \textit{ there exists } \sigma_f \in \textit{Pref}(\sigma) \textit{ s.t. for all } \sigma^\omega \in \mathcal{L}(V) \sigma_f \sigma^\omega \notin P.$$

Furthermore, we denote σ_f as a bad prefix *of P.*

2.3.2 Liveness Property

Informally, liveness properties stipulate that some condition will occur infinitely often. When a property is liveness, it is not possible to stipulate whether the property holds looking at its finite prefixes. Figure 2.2 shows a graphical view of a liveness property.

Definition 2.2 (Liveness Property) *Let P be a property over a set of variables V, P is a liveness property iff*

$$\textit{for all } \sigma \in \mathcal{L}(V), \textit{ for all } \sigma_f \in \textit{Pref}(\sigma), \textit{ there exists } \sigma^\omega \in \mathcal{L}(V) \textit{ s.t. } \sigma_f \sigma^\omega \in P$$

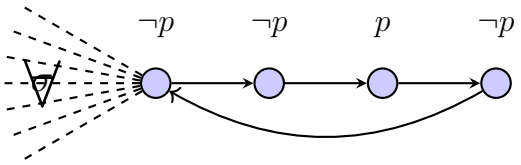


Figure 2.2: Graphical representation of set of traces satisfying the liveness property stating that p holds infinitely often. The loop representing infinite trace ensure that independently on the prefix of the trace, the property infinitely often p will be satisfied.

2.4 Symbolic Transition System

A Symbolic Transition System, called for brevity STS is a formal model for representing the behaviour of a system using logic formulae.

An STS is defined as the tuple $M \doteq \langle V, I, T \rangle$ where

- V is the set of variables interpretable over a first-order structure $\mathcal{M}$.
- $I(V)$ is a formula over V symbolically representing the initial states.
- $T(V, V')$ is a formula over $V \cup V'$ symbolically representing the transition relation.

where $\cdot'$ be a bijective function that maps a variable v to a variable v' and $V' \doteq \{v' | v \in V\}$.

Moreover, we generalize the STS considering Symbolic Transition System with fairness constraints as $M \doteq \langle V, I, T, F \rangle$ where F is a set of formulae over V. An STS without fairness can be represented as an STS with fairness where $F = \emptyset$.

Definition 2.3 (Path) *We denote a path π of $M = \langle V, I, T, F \rangle$ as a sequence of states $\pi \doteq s_0, \ldots$ such that $s_0 \models I$ and, for all $0 \leq i < |\pi| - 1$: $s_i \cdot s'_{i+1} \models T$.*

Moreover, we say that a path π is a fair path *of M if and only if π is an infinite path and for all $f \in F$: for each i there is a $k \geq i$ $\pi, k \models f$.*

Definition 2.4 (Reachable and Fair States) *We denote a state s of M as an assignment over the variables V. We say that a state s is* reachable *iff there exists a path of length $n \in \mathbb{N}$ $\pi = s_0 s_1, \ldots, s_n$ s.t. π is a path of M. Moreover, we say that a state s is* fair *iff it is* reachable *and there exists an infinite path π^ω s.t. $\pi\pi^\omega$ is a* fair *path of M.*

Definition 2.5 (Deadlock and Livelock) *We say that a state s of M is a deadlock state iff there is no successor of s in M i.e. there is $s \not\models \exists V'.T(V, V')$. We say that M is deadlock-free, if no reachable state s of M is a deadlock state.*

Moreover, we consider the more general notion of Livelocks for Fair STS. We say that a state s of a Fair STS M (i.e. STS with fairness constraints) is a livelock iff there is no infinite path starting from s that reaches each $f \in F$ infinitely often. We say that M is livelock-free if no s of M is a livelock state. If M is livelock-free we can easily observe that it is also deadlock-free even though the converse is not true in general.

Definition 2.6 (STS Language) *Let M be an STS, we define the infinite language as $\mathcal{L}(M) \doteq \{\pi | \pi$ is a fair path of $M\}$. Moreover, we defined the finite language as $\mathcal{L}_{fin}(M) \doteq \{\pi | \pi$ is a finite path of $M\}$.*

Definition 2.7 (STS Synchronous Composition) *Let $M_1 = \langle V_1, I_1, T_1, F_1 \rangle$ and $M_2 = \langle V_2, I_2.T_2, F_2 \rangle$ be two STS. We define the synchronous composition of M_1 and M_2 as follows*

$$M_1 \times M_2 \doteq \langle V_1 \cup V_2, I_1 \wedge I_2, T_1 \wedge T_2, F_1 \cup F_2 \rangle$$

2.5 Linear Temporal Logic

In formal verification, system properties are often specified using temporal logics such as Linear Temporal Logic (LTL) [257]. LTL enables the specification of constraints over sequences of events and message exchanges between components. Formally, LTL formulae are interpreted over traces–that is, sequences of variable assignments representing system states over time.

LTL extends propositional logic with temporal operators, including the "next" operator $\mathbf{X}$ and the "until" operator $\mathbf{U}$. The semantics of these operators are intuitive: $\mathbf{X}\varphi$ holds if φ is true in the next state, while $\varphi_1 \,\mathbf{U}\, \varphi_2$ holds if φ_1 continues to hold until φ_2 becomes true.

An extension of LTL with past-time operators was introduced in [228]. This extension adds the "yesterday" operator $\mathbf{Y}$ and the "since" operator $\mathbf{S}$, whose semantics mirror those of their future-time counterparts, with the key distinction that $\mathbf{Y}\varphi$ is false in the initial state.

While past-time operators often allow for more concise and readable property specifications, they do not increase the expressive power of the logic [171].

To illustrate the range of behaviours that LTL can express, we include a subset of the specification patterns defined by Dwyer et al. [147] in Table 2.1.

In the following, we define the semantics of LTL for both the *infinite* and the *finite* semantics. For the finite semantics, we use the semantics of LTL_f presented in [137].

Pattern Name	LTL property
Absence Globally	$\mathbf{G}\neg p$
Absence Between q and r	$\mathbf{G}((q \wedge \neg r \wedge \mathbf{F}r) \rightarrow (\neg p \,\mathbf{U}\, r))$
Bounded existence after q	$\mathbf{F}q \rightarrow (\neg q \,\mathbf{U}\, (q \wedge (\neg p \,\mathbf{W}\, (p \,\mathbf{W}\, (\neg p \,\mathbf{W}\, (p \,\mathbf{W}\, \mathbf{F}\neg p))))))$
Response Globally	$\mathbf{G}(p \rightarrow \mathbf{F}s)$
Precedence between q and r	$\mathbf{G}((q \wedge \neg r \wedge \mathbf{F}r) \rightarrow (\neg \,\mathbf{U}\, (s \vee r)))$
Response chain globally	$\mathbf{G}(s \wedge \mathbf{XF}t \rightarrow \mathbf{X}(\mathbf{F}(t \wedge \mathbf{F}p)))$
Response chain between q and r	$\mathbf{G}((q \wedge \mathbf{F}r) \rightarrow (s \wedge \mathbf{X}(\neg r \,\mathbf{U}\, t) \rightarrow \mathbf{X}(\neg r \,\mathbf{U}\, (t \wedge \mathbf{F}p))) \,\mathbf{U}\, r)$
Constrained Chain Pattern after q	$\mathbf{G}(q \rightarrow \mathbf{G}(p \rightarrow (s \wedge \neg z \wedge \mathbf{X}(\neg z \,\mathbf{U}\, t))))$

Table 2.1: Subset of properties from the LTL Dwyer pattern[147].

2.5.1 Syntax and Semantics

The syntax of LTL extended with past is defined as follows:

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \,\mathbf{U}\, \varphi \mid \mathbf{X}\varphi \mid \mathbf{Y}\varphi \mid \varphi \,\mathbf{S}\, \varphi$$

Given a trace σ over the symbols of an LTL formula φ. The semantics of an LTL formula is defined inductively as follows:

$$\begin{aligned}
&\sigma, i \models \top \\
&\sigma, i \models p && \Leftrightarrow \sigma(i)(p) = \top \\
&\sigma, i \models \neg\varphi && \Leftrightarrow \sigma, i \not\models \varphi \\
&\sigma, i \models \varphi_1 \vee \varphi_2 && \Leftrightarrow \sigma, i \models \varphi_1 \text{ or } \sigma, i \models \varphi_2 \\
&\sigma, i \models \mathbf{X}\varphi && \Leftrightarrow i < |\sigma| - 1 \text{ and } \sigma, i+1 \models \varphi \\
&\sigma, i \models \mathbf{Y}\varphi && \Leftrightarrow i > 0 \text{ and } \sigma, i-1 \models \varphi \\
&\sigma, i \models \varphi_1 \,\mathbf{U}\, \varphi_2 && \Leftrightarrow \text{ there is } i \leq k < |\sigma| - 1, \sigma, k \models \varphi_2 \text{ and for all } i \leq j < k, \sigma, j \models \varphi_1 \\
&\sigma, i \models \varphi_1 \,\mathbf{S}\, \varphi_2 && \Leftrightarrow \text{ there is } 0 \leq k \leq i, \sigma, k \models \varphi_2 \text{ and for all } i \geq j > k, \sigma, j \models \varphi_1
\end{aligned}$$

Finally, we have that $\sigma \models \varphi$ iff $\sigma, 0 \models \varphi$. To better distinguish between finite and infinite trace, in the remainder of this thesis, we will use $\models_f$ to denote the satisfiability of a finite trace with the finite semantics of LTL (and subsequently other logics). The problem of model checking an LTL formula considers induced traces from paths of M to the variables of φ (denoted as $V(\varphi)$). Formally, let M be an STS,

$$M \models \varphi \text{ iff for all } \sigma \in \mathcal{L}(M) : \sigma(V(\varphi)) \models \varphi.$$
$$M \models_f \varphi \text{ iff for all } \sigma \in \mathcal{L}_{fin}(M) : \sigma(V(\varphi)) \models_f \varphi.$$

We use the following standard abbreviations: $\varphi_1 \wedge \varphi_2 := \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \,\mathbf{R}\, \varphi_2 := \neg(\neg\varphi_1 \,\mathbf{U}\, \neg\varphi_2)$ (φ_1 releases φ_2), $\mathbf{F}\varphi := \top \,\mathbf{U}\, \phi$ (sometime in the future φ), $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$

(always in the future φ), $\mathbf{O}\varphi := \top\,\mathbf{S}\,\varphi$ (once in the past φ), $\mathbf{H}\varphi := \neg\mathbf{O}\neg\varphi$ (historically in the past φ), $\mathbf{Z}\varphi := \neg\mathbf{Y}\neg\varphi$ (yesterday φ or at initial state), $\mathbf{X}^n\varphi := \mathbf{X}\mathbf{X}^{n-1}\varphi$ with $\mathbf{X}^0\varphi := \varphi$, $\mathbf{Y}^n\varphi := \mathbf{Y}\mathbf{Y}^{n-1}\varphi$ with $\mathbf{Y}^0\varphi := \varphi$, $\mathbf{Z}^n\varphi := \mathbf{Z}\mathbf{Z}^{n-1}\varphi$ with $\mathbf{Z}^0\varphi := \varphi$, $\mathbf{F}^{\leq n}\varphi := \varphi \vee \mathbf{X}\varphi \vee \cdots \vee \mathbf{X}^n\varphi$, $\mathbf{G}^{\leq n}\varphi := \neg\mathbf{F}^{\leq n}\neg\varphi$, $\mathbf{O}^{\leq n}\varphi := \varphi \vee \mathbf{Y}\varphi \vee \cdots \vee \mathbf{Y}^n\phi$, $\mathbf{H}^{\leq n}\varphi := \neg\mathbf{O}^{\leq n}\neg\varphi$.

When referring to formulae interpreted over finite trace, we introduce $\mathbf{N}$ as the weaker dual of $\mathbf{X}$, formally $\mathbf{N}\varphi := \neg\mathbf{X}\neg\varphi$. When interpreted over infinite traces, $\mathbf{N}$ is instead identical to $\mathbf{X}$.

We introduce now the normalization for LTL formula. A formula is in Negation Normal Form (NNF) if the negation occurs only in front of the atomic formulae. We define the rewriting NNF for LTL.

If we consider the operators $\vee, \mathbf{R}, \mathbf{T}, \mathbf{Z}$ defined by the abbreviations introduced above as primitive, every LTL formula can be converted into an equivalent one in NNF. In particular, $\text{NNF}(\varphi)$ is obtained by applying the following rewriting to every occurrence of negations in φ:

$$\begin{aligned}
\text{NNF}(\neg(\varphi_1 \vee \varphi_2)) &:= \text{NNF}(\neg\varphi_1) \wedge \text{NNF}(\neg\varphi_2), & \text{NNF}(\neg(\varphi_1 \wedge \varphi_2)) &:= \text{NNF}(\neg\varphi_1) \vee \text{NNF}(\neg\varphi_2),\\
\text{NNF}(\neg(\varphi_1\,\mathbf{R}\,\varphi_2)) &:= \text{NNF}(\neg\varphi_1)\,\mathbf{U}\,\text{NNF}(\neg\varphi_2), & \text{NNF}(\neg(\varphi_1\,\mathbf{U}\,\varphi_2)) &:= \neg\text{NNF}(\neg\varphi_1)\,\mathbf{R}\,\text{NNF}(\neg\varphi_2),\\
\text{NNF}(\neg(\varphi_1\,\mathbf{T}\,\varphi_2)) &:= \text{NNF}(\neg\varphi_1)\,\mathbf{S}\,\text{NNF}(\neg\varphi_2), & \text{NNF}(\neg(\varphi_1\,\mathbf{S}\,\varphi_2)) &:= \text{NNF}(\neg\varphi_1)\,\mathbf{T}\,\text{NNF}(\neg\varphi_2),\\
\text{NNF}(\neg(\mathbf{Z}\varphi)) &:= \mathbf{Y}\text{NNF}(\neg\varphi), & \text{NNF}(\neg(\mathbf{Y}\varphi)) &:= \mathbf{Z}\text{NNF}(\neg\varphi),\\
& & \text{NNF}(\neg(\mathbf{X}\varphi)) &:= \mathbf{X}\text{NNF}(\neg\varphi).
\end{aligned}$$

If we also consider $\mathbf{N}$ as a primitive operator, then every LTL_f formula can be converted into an equivalent one in NNF. The corresponding rewriting $\text{NNF}_f(\varphi)$ is defined similarly to NNF, except for the last case that is replaced by $\text{NNF}_f(\neg(\mathbf{X}\varphi)) := \mathbf{N}\text{NNF}_f(\neg\varphi)$ and $\text{NNF}_f(\neg(\mathbf{N}\varphi)) := \mathbf{X}\text{NNF}_f(\neg\varphi)$.

2.5.2 Safety Fragments of LTL

SafetyLTL is a fragment of Linear Temporal Logic which disallows positive occurrence of until. We present the logic in negation normal form.

Definition 2.8 (SafetyLTL) *The syntax of SafetyLTL is defined as follows:*

$$\varphi := \top \mid \bot \mid p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi\,\mathbf{R}\,\varphi \mid \boldsymbol{X}\varphi \mid \boldsymbol{Y}\varphi \mid \boldsymbol{Z}\varphi \mid \varphi\,\mathbf{S}\,\varphi \mid \varphi\,\mathbf{T}\,\varphi$$

Note that we use include also past operators in SafetyLTL although this is typically defined only with the future ones.

Another safe fragment of LTL is full-past LTL. Although the syntax of the fragment is far stricter than SafetyLTL, the two logics have the same expressiveness.

Definition 2.9 (Full past LTL) *The syntax of full-past LTL is of the form* $\phi := \boldsymbol{G}\beta$ *where* β *is as follows:*

$$\beta := \beta \wedge \beta \mid \neg\beta \mid \boldsymbol{Y}\beta \mid \beta\, \mathbf{S}\, \beta \mid p$$

where $p \in V$ *is a Boolean variable.*

It was proved in [89] that both fragments, SafetyLTL (even without past operators) and full-past LTL express all and only the safety properties of LTL.

2.5.3 Automata-Theoretic Approach

Reduction to Liveness and Invariant Checking The automata-theoretic approach [286] to LTL model checking is to transform an LTL formula φ over V into a Büchi automaton, which is here represented symbolically by an STS $M^l_{\neg\varphi}$ with a fairness condition $f^l_{\neg\varphi}$ such that, for every infinite trace σ, $\sigma \models \neg\varphi$ iff there exists an infinite trace σ' of $M^l_{\neg\varphi}$ with $\sigma'_{|V} = \sigma$ visiting $f^l_{\neg\varphi}$ infinitely often. Thus, $M \models \varphi$ iff $M \times M^l_{\neg\varphi} \models \mathbf{FG}\neg f^l_{\neg\varphi}$.

Similarly, for LTL_f, the automata-theoretic approach [137] transforms a formula φ over V into an automaton on finite traces, which is here represented by an STS $M^s_{\neg\varphi}$ with a final condition $f^s_{\neg\varphi}$ such that, for every finite trace σ, $\sigma \models \neg\varphi$ iff there exists a finite σ' of $M^s_{\neg\varphi}$ with $\sigma'_{|V} = \sigma$ reaching $f^s_{\neg\varphi}$. Thus, $M_{fin} \models \varphi$ iff $M \times M^s_{\neg\varphi} \models_f \mathbf{G}\neg f^s_{\neg\varphi}$.

Finally, for SafetyLTL, the automata-theoretic approach [219] transforms a SafetyLTL formula φ into an automaton over finite traces, here represented by an STS $M^b_{\neg\varphi}$ with a final condition $f^b_{\neg\varphi}$ such that, for every infinite trace σ, $\sigma \models \neg\varphi$ iff there exists a finite trace σ of $M^b_{\neg\varphi}$ with $\sigma_{|V} = \sigma$ and with a bad prefix reaching $f^b_{\neg\varphi}$. Thus, if M is deadlock free, $M \models \varphi$ iff $M \times M^b_{\neg\varphi} \models_f \mathbf{G}\neg f^b_{\neg\varphi}$.

Note that the above construction where the safety fragment of LTL can be reduced to invariant checking over finite traces can be extended also beyond the safety fragment, in particular considering the notion of relative safety, first introduced in [194]. In fact, in Section 4.3, we will show that under specific assumptions, the reduction can be extended to formulae of the form $\alpha \rightarrow \varphi$, where φ is safety.

2.5.4 Symbolic Compilation of Full LTL

There are various automata compilation approaches [117, 212, 114]. The one proposed back in [117] is still used in tools such as nuXmv [87]. With small variants, this produces an STS that works for all the cases mentioned above.

Definition 2.10 *Given an LTL formula ϕ, the STS $ltl2sts(\phi) = \langle V_\phi, I_\phi, T_\phi \rangle$ is defined as follows:*

- $V_\phi = V \cup \{v_{\boldsymbol{X}\beta} \mid \boldsymbol{X}\beta \in Sub(\phi)\} \cup \{v_{\boldsymbol{X}(\beta_1 \mathbf{U} \beta_2)} \mid \beta_1 \mathbf{U} \beta_2 \in Sub(\phi)\} \cup \{v_{\boldsymbol{Y}\beta} \mid \boldsymbol{Y}\beta \in Sub(\phi)\} \cup \{v_{\boldsymbol{Y}(\beta_1 \mathbf{S} \beta_2)} \mid \beta_1 \mathbf{S} \beta_2 \in Sub(\phi)\}$
- $I_\phi = Enc(\phi) \wedge \bigwedge_{v_{\boldsymbol{Y}\beta} \in V_{\neg\phi}} \neg v_{\boldsymbol{Y}\beta}$
- $T_\phi = \bigwedge_{v_{\boldsymbol{X}\beta} \in V_\phi} v_{\boldsymbol{X}\beta} \leftrightarrow Enc(\beta)' \wedge \bigwedge_{v_{\boldsymbol{Y}\beta} \in V_{\neg\phi}} Enc(\beta) \leftrightarrow v'_{\boldsymbol{Y}\beta}$

where Sub is a function that maps a formula ϕ to the set of its subformulae, and Enc is defined recursively as follows.

- $Enc(\top) = \top$
- $Enc(v) = v$
- $Enc(\phi_1 \wedge \phi_2) = Enc(\phi_1) \wedge Enc(\phi_2)$
- $Enc(\neg\phi_1) = \neg Enc(\phi_1)$
- $Enc(\boldsymbol{X}\phi_1) = v_{\boldsymbol{X}\phi_1}$
- $Enc(\phi_1 \mathbf{U} \phi_2) = Enc(\phi_2) \vee (Enc(\phi_1) \wedge v_{\boldsymbol{X}(\phi_1 \mathbf{U} \phi_2)})$
- $Enc(\boldsymbol{Y}\phi_1) = v_{\boldsymbol{Y}\phi_1}$
- $Enc(\phi_1 \mathbf{S} \phi_2) = Enc(\phi_2) \vee (Enc(\phi_1) \wedge v_{\boldsymbol{Y}(\phi_1 \mathbf{S} \phi_2)})$

Intuitively, the variables in V_ϕ are proof obligations for the future states in the trace that must satisfy ϕ; they are initially set to a value according to $Enc(\phi)$ so that ϕ is satisfied in the initial state and are propagated by the transition relation if needed.

Let $F_\phi = \{Enc(\beta_1 \mathbf{U} \beta_2 \rightarrow \beta_2) \mid \beta_1 \mathbf{U} \beta_2 \in Sub(\phi)\}$[1].

Definition 2.11 (Degeneralization of STS) *Given a set of formulae F over the variables V, we can build an STS $S_{deg}(F) = \langle V \cup V_{deg}, I_{deg}, T_{deg} \rangle$ called degeneralization of F, which is built as follows:*

- $V_{deg} = \{v_f \mid f \in F\}$ *is a set of Boolean variables*
- $I_{deg} = \bigwedge_{v_f \in V_{deg}} \neg v_f$

[1]It should be noted that it is possible to restrict the amount of fairness constraint considering only $\beta_1 \mathbf{U} \beta_2 \in Sub(\phi)$ occurring positively in ϕ.

- $T_{deg} = (\bigwedge_{v_f \in V_{deg}}((\neg v_f \wedge f) \rightarrow v'_f)) \wedge ((\bigwedge_{v_f \in V_{deg}} v_f) \rightarrow (\bigwedge_{v_f \in V_{deg}} \neg v'_f)) \wedge$
  $((\neg \bigwedge_{v_f \in V_{deg}} v_f) \rightarrow (\bigwedge_{v_f \in V_{deg}}(v_f \rightarrow v'_f)))$

We can finally define $M^f_{\neg\varphi} = ltl2sts(\neg\varphi) \times M_{deg}(F_{\neg\varphi})$. Let $f^l_{\neg\varphi} = \bigwedge_{v_f \in V_{deg}} \neg v'_f$.

Theorem 2.1 (Correctness of Full-LTL Symbolic Construction) *For any infinite trace σ, $\sigma \models \neg\varphi$ iff there exists an infinite trace σ of $M^l_{\neg\varphi}$ over σ visiting $f^l_{\neg\varphi}$ infinitely many times.*

In case of LTL_f, the construction is applied to a formula in NNF and the STS is built with a modification on the transition condition, where the double implication is replaced by a single implication.

Definition 2.12 (Symbolic Compilation of LTL_f) *Given an LTL formula ϕ, the STS $ltlf2sts(\phi) = \langle V_\phi, I_\phi, T_\phi \rangle$ is defined as follows:*

- $V_\phi = V \cup \{v_{X\beta} \mid \boldsymbol{X}\beta \in Sub(\phi)\} \cup \{v_{\boldsymbol{X}(\beta_1 \mathbf{U} \beta_2)} \mid \beta_1 \mathbf{U} \beta_2 \in Sub(\phi)\} \cup \{v_{\boldsymbol{Y}\beta} \mid \boldsymbol{Y}\beta \in Sub(\phi)\} \cup \{v_{\boldsymbol{Y}\beta_1 \mathbf{S} \beta_2} \mid \beta_1 \mathbf{S} \beta_2 \in Sub(\phi)\} \cup \{v_{\boldsymbol{N}\beta} \mid \boldsymbol{N}\beta \in Sub(\phi)\} \cup \{v_{\boldsymbol{X}(\beta_1 \mathbf{R} \beta_2)} \mid \beta_1 \mathbf{R} \beta_2 \in Sub(\phi)\} \cup \{v_{\boldsymbol{Z}\beta} \mid \boldsymbol{Z}\beta \in Sub(\phi)\} \cup \{v_{\boldsymbol{Z}\beta_1 \mathbf{T} \beta_2} \mid \beta_1 \mathbf{T} \beta_2 \in Sub(\phi)\}$
- $I_\phi = Enc(\phi) \wedge \bigwedge_{v_{\boldsymbol{Y}\beta} \in V_{\neg\phi}} \neg v_{\boldsymbol{Y}\beta} \wedge \bigwedge_{v_{\boldsymbol{Z}\beta} \in V_{\neg\phi}} v_{\boldsymbol{Z}\beta}$
- $T_\phi = \bigwedge_{v_{\boldsymbol{X}\beta} \in V_\phi} v_{\boldsymbol{X}\beta} \rightarrow Enc(\beta)' \wedge \bigwedge_{v_{\boldsymbol{Y}\beta} \in V_{\neg\phi}} Enc(\beta) \rightarrow v'_{\boldsymbol{Y}\beta} \wedge \bigwedge_{v_{\boldsymbol{N}\beta} \in V_\phi} v_{\boldsymbol{N}\beta} \rightarrow Enc(\beta)' \wedge \bigwedge_{v_{\boldsymbol{Z}\beta} \in V_{\neg\phi}} Enc(\beta) \rightarrow v'_{\boldsymbol{Z}\beta}$

where Sub is a function that maps a formula ϕ to the set of its subformulae, and Enc extends the previous definition as follows:

- $Enc(\phi_1 \vee \phi_2) = Enc(\phi_1) \vee Enc(\phi_2)$
- $Enc(\boldsymbol{N}\phi_1) = v_{\boldsymbol{N}\phi_1}$
- $Enc(\phi_1 \mathbf{R} \phi_2) = Enc(\phi_2) \wedge (Enc(\phi_1) \vee v_{\boldsymbol{N}(\phi_1 \mathbf{R} \phi_2)})$
- $Enc(\boldsymbol{Z}\phi_1) = v_{\boldsymbol{Z}\phi_1}$
- $Enc(\phi_1 \mathbf{T} \phi_2) = Enc(\phi_2) \wedge (Enc(\phi_1) \vee v_{\boldsymbol{Z}(\phi_1 \mathbf{T} \phi_2)})$

Let us define $\psi = \text{NNF}_f(\neg\varphi)$ and $M^s_{\neg\varphi} = ltlf2sts(\psi)$. The condition $f^s_{\neg\varphi}$ is then defined as $\bigwedge_{v_{\mathbf{X}\beta} \in V_\psi} \neg v_{\mathbf{X}\beta}$.

Theorem 2.2 (Correctness of LTL_f Symbolic Construction) *For every finite trace σ, $\sigma \models \neg\varphi$ iff there exists a finite trace σ of $M^s_{\neg\varphi}$ with $\sigma_{|V} = \sigma$ reaching $f^s_{\neg\varphi}$.*

In case of SafetyLTL, similar constructions can be used based on the notion of informative prefix defined in [219]. In the following we consider an extended version with past operators.

Definition 2.13 (Informative Prefix [219]) *Let ψ be an LTL formula in negative normal form, $Sub(\psi)$ be the set of sub-formulae of ψ and let σ be a finite trace of length n over the language of ψ. We say that σ is informative for ψ iff there exists a mapping $L : \{0, \ldots, n\} \to 2^{Sub(\neg\psi)}$ such that:*

1. $\neg\psi \in L(0)$.
2. $L(n) = \emptyset$ *and* $L(-1) = \emptyset$.
3. *For all* $0 \leq i < n$, *for all* $\varphi \in L(i)$:
   - *If* φ *is propositional,* $\sigma, i \models \varphi$.
   - *If* $\varphi = \varphi_1 \vee \varphi_2$, $\varphi_1 \in L(i)$ *or* $\varphi_2 \in L(i)$.
   - *If* $\varphi = \varphi_1 \wedge \varphi_2$, $\varphi_1 \in L(i)$ *and* $\varphi_2 \in L(i)$.
   - *If* $\varphi = \boldsymbol{X}\varphi_1$, $\varphi_1 \in L(i+1)$.
   - *If* $\varphi = \varphi_1 \,\mathbf{U}\, \varphi_2$, $\varphi_2 \in L(i)$ *or* $[\varphi_1 \in L(i)$ *and* $\varphi_1 \,\mathbf{U}\, \varphi_2 \in L(i+1)]$.
   - *If* $\varphi = \varphi_1 \,\mathbf{R}\, \varphi_2$, $\varphi_2 \in L(i)$ *and* $[\varphi_1 \in L(i)$ *or* $\varphi_1 \,\mathbf{R}\, \varphi_2 \in L(i+1)]$.
   - *If* $\varphi = \boldsymbol{Y}\varphi_1$, $i > 0$ $\varphi_1 \in L(i-1)$
   - *If* $\varphi = \varphi_1 \,\mathbf{S}\, \varphi_2$, $\varphi_2 \in L(i)$ *or* $[i > 0$, $\varphi_1 \in L(i)$ *and* $\varphi_1 \,\mathbf{S}\, \varphi_2 \in L(i-1)]$.
   - *If* $\varphi = \varphi_1 \,\mathbf{T}\, \varphi_2$, $\varphi_2 \in L(i)$ *and* $[\varphi_1 \in L(i)$ *or* $i = 0$ *or* $\varphi_1 \,\mathbf{R}\, \varphi_2 \in L(i+1)]$.

2.6 SAT/SMT Based Model Checking of Trace Properties

In this dissertation, we focus on model checking techniques based on SAT and SMT solving. The core idea behind these approaches is to reduce the model checking problem to a series of logical queries: to SAT when analyzing finite-state systems, and to SMT when handling infinite-state systems.

This section introduces the symbolic techniques used to verify both safety and liveness properties. We begin with invariant checking, which lies at the heart of safety verification, and later present approaches for verifying liveness properties.

2.6.1 Invariants and Inductive Invariants

In Section 2.3.1, we introduced the notion of safety properties. In Section 2.5.3, we showed how the verification of certain safety fragments of LTL can be reduced to checking properties of the form $\mathbf{G}\phi$ over finite traces.

The problem of checking whether a symbolic transition system (STS) satisfies a property over finite traces ($\models_f$) of the form $\mathbf{G}\phi$, where ϕ is a propositional formula (in the finite-state case) or a first-order formula (in the infinite-state case), is known as the invariant checking problem.

A formula ϕ is said to be an invariant of an STS M if it holds in all reachable states of M; that is, for every reachable state s, we have $s \models \phi$.

Invariant checking has been extensively studied in the literature. In this dissertation, we focus on symbolic approaches that reduce the verification task to SAT or SMT queries, enabling automated reasoning even in the presence of large or infinite state spaces.

A common and effective technique for proving safety properties [246, 272, 75, 227] involves searching for a stronger invariant that is also inductive. An inductive invariant implies the target property and can be validated using induction over the transition relation, thereby avoiding explicit enumeration of reachable states.

Definition 2.14 (Inductive Invariant) *Let M be an STS and P be an invariant over the variables of M. We say that P is an inductive invariant of M iff the following conditions occur:*

1. $I \models P(V)$ *i.e. Initially the invariant is satisfied (base case)*

2. $T(V, V') \wedge P(V) \models P(V')$ *i.e. if a state satisfies P all its successors satisfy P as well (inductive case)*

Moreover, given another invariant Q over the symbol of M which is not inductive, we say that P is an invariant strengthening of Q iff P is an inductive invariant and $P \models Q$.

It is easy to see that any inductive invariant is satisfied by M. Moreover, if an invariant has an inductive strengthening then it is satisfied by M as well. Finally, we know that an invariant is satisfied only if exists an inductive invariant strengthening, making the search of inductive invariant a key approach for proving safety properties.

2.6.2 Bounded Model Checking

Bounded Model Checking (BMC) [45] considers only initial paths of length up to a certain bound. Given a bound k, an STS M and a formula $\varphi(V)$, the bounded reachability problem is the problem of finding, for some $j \leq k$ an initial finite path $s_0, ..., s_j$ of M such that $s_j \models \varphi$. In BMC, the problem is usually reduced to a satisfiability problem.

In the following, given a set V of variables, we use several copies of V, one for each state of the path we are looking for. We will denote with V_i the i-th copy of V. Thus, $V_i = \{v_i\}_{v \in V}$, where v_i represents the i-th copy of v. For a given bound k, we often use $k+1$ copies $V_0, \ldots, V_k$ and, when k is clear from the context, we denote their union by $\overline{V}$.

Given an STS M, and a bound k, the formula $\mathit{Unroll}_k(\overline{V})$ is defined as follows:

$$\mathit{Unroll}_k(\overline{V}) := \bigwedge_{1 \leq h \leq k} T(V_{h-1}, V_h)$$

Given an STS M, a bound k, and a formula φ, the formula $\mathit{BMC}_k(\overline{V})$ is defined as follows:

$$\mathit{BMC}_k(\overline{V}) := I(V_0) \wedge \mathit{Unroll}_k(\overline{V}) \wedge \bigvee_{0 \leq i \leq k} \varphi(V_i)$$

The formula BMC_k encodes the bounded reachability problem, in the sense that BMC_k is satisfiable if and only if there exists an initial path of length up to k reaching φ.

2.6.3 K-Induction

K-induction [272] is a technique that proves that if a set of states is not reachable in k steps, then it is not reachable at all. Analogous to induction principle, k-induction consists of a *base step*—which solves the bounded reachability problem with a given bound k of steps—and an *inductive step*, which concludes that k is sufficient to solve the (unbounded) reachability problem. The idea of the inductive step is to check whether the target set of states cannot be reached in k steps. Both of these checks can be solved by means of satisfiability.

Given an STS M, and a bound k and a formula φ, the formula KIND_k is defined as follows:

$$\mathit{KIND}_k := \mathit{Unroll}_k(\overline{V}) \wedge \bigwedge_{0 \leq i < j \leq k} \neg \bigwedge_{v \in V} v_i = v_j \wedge \bigwedge_{0 \leq i < k} \neg\varphi(V_i) \wedge \varphi(V_k)$$

If $KIND_k$ is unsatisfiable, then M does not have a non-initialized simple path (i.e., a path without repetitions) reaching φ in k steps.

If BMC_k and $KIND_k$ are both unsatisfiable, then φ is not reachable in M. Thus, in order to prove that φ is not reachable one can check the unsatisfiability of BMC_k and $KIND_k$ for increasing values of k. Usually, these checks can enjoy sharing lemmas discovered trying to prove unsatisfiability, and therefore the checks are performed using incremental SAT solvers.

2.6.4 Checking Liveness

In the following, we present known techniques to assess whether a proposition f will eventually occur forever in an STS M i.e. if $M \models \mathbf{FG}f$. This technique can be used to verify LTL by applying the automata-theoretic construction of Section 2.5.3.

Liveness to Safety The *liveness-to-safety reduction* (L2S) [44] is a technique for reducing an LTL model checking problem on a finite-state transition system to an invariant model checking problem. The idea is to encode the absence of a lasso-shaped path violating the LTL property $\mathbf{FG}f$ as an invariant property.

The encoding is achieved by transforming the original STS M to the STS M_{L2S}, introducing a set $\overline{V}$ of variables containing a copy $\overline{v}$ for each state variable x of the original system, plus additional variables *seen*, *triggered* and *loop*. Assuming that the STS is a finite state STS, there is a lasso-shaped path that visits infinitely often f iff S_{L2S} can visit the guessed state twice along a path in which f was true after the first time the guessed state is seen. This condition can be expressed as an invariant property. Let $M \doteq \langle V, I, T\rangle$. L2S transforms the STS in $M_{\mathrm{L2S}} \doteq \langle V_{\mathrm{L2S}}, I_{\mathrm{L2S}}, T_{\mathrm{L2S}}\rangle$ so that $M \models \mathbf{FG}\neg f$ if and only if $M_{\mathrm{L2S}} \models \neg bad_{\mathrm{L2S}}$, where:

$$
\begin{aligned}
V_{\mathrm{L2S}} \doteq\ & V \cup \overline{V} \cup \{seen, triggered, loop\} \\
I_{\mathrm{L2S}} \doteq\ & I \wedge \neg seen \wedge \neg triggered \wedge \neg loop \\
T_{\mathrm{L2S}} \doteq\ & T \wedge \big[\textstyle\bigwedge_{v\in V} \overline{v} \leftrightarrow \overline{v}'\big] \\
& \wedge\big[seen' \leftrightarrow (seen \vee \textstyle\bigwedge_{v\in V}(v \leftrightarrow \overline{v}))\big] \\
& \wedge\big[triggered' \leftrightarrow (triggered \vee (f \wedge seen'))\big] \\
& \wedge\big[loop' \leftrightarrow (triggered' \wedge \textstyle\bigwedge_{v\in V}(v' \leftrightarrow \overline{v}'))\big] \\
bad_{\mathrm{L2S}} \doteq\ & loop
\end{aligned}
$$

The variables $\overline{V}$ are used to non-deterministically guess a state of the system from which a reachable fair loop starts. The additional variables are used to remember that

the guessed state was seen once and that the signal f was true at least once afterwards.

K-liveness K-liveness [116] is an algorithm that reduces symbolic LTL model checking to invariant checking. In order to prove the validity of an LTL formula ψ in an STS M, the algorithm counts the occurrence of the fairness condition in the composed automaton $M \times M_{\neg\psi}$. If the algorithm is able to find a bound k such that the fairness condition f is visited at most k times, then the property is valid. It should be noted that the algorithm is not able to disprove the property; therefore, it is usually executed in lockstep with BMC.

The formal construction of the problem is as follows:

$$M_e = \langle V_\psi \cup \{c\}, I_\psi \wedge c = 0, T_\psi \wedge (f \rightarrow c' = c + 1) \wedge (\neg f \rightarrow c' = c)\rangle$$
$$M \times M_e \models c \leq k$$

where M_ψ is the automata construction of ψ, c is the counter variable, M_e is the automata of ψ extended with the fairness counter and k is a positive integer constant.

2.7 Timed Systems and MTL

2.7.1 Notion of Time

We start by providing the notion of time model that is going to be used in the trace. We consider a super-dense model of time [6, 236, 7], which interchanges dense time evolution with instantaneous discrete changes.

A time model is a structure $\tau = \langle T, <, \mathbf{0}, v\rangle$ with a temporal domain T, a total order $<$ over T, a minimum element $\mathbf{0} \in T$, and a function $v : T \rightarrow \mathbb{R}_0^+$ that represents the real-time of a time point in T. A *time point* is an element of T. A time interval sequence is a sequence $I_0, I_1, I_2, \ldots$ of intervals of reals such that, for all $i \geq 0$, I_i and I_{i+1} are almost adjacent (subsequent intervals can overlap in at most one point) and $\bigcup_{i\geq 0} I_i = \mathbb{R}_0^+$.

- In discrete time models, $T = \mathbb{N}$, $\mathbf{0}$ and $<$ are the standard zero and order over natural numbers, $v(0) = 0$ and $v(0), v(1), v(2), \ldots$ is a non-decreasing divergent sequence.

- In super-dense time models,

1. $T \subset \mathbb{N} \times \mathbb{R}_0^+$ such that the sequence of sets $I_0, I_1, I_2, \ldots$ where, for all $i \geq 0$, the set $I_i := \{r \mid \langle i, r\rangle \in T\}$, is a time interval sequence,
2. $\langle i, r\rangle < \langle i', r'\rangle$ iff $i < i'$ or $i = i'$ and $r < r'$
3. $\mathbf{0} = \langle 0, 0\rangle \in \mathbb{N} \times \mathbb{R}_0^+$, and
4. $v(\langle i, r\rangle) = r$.

Intuitively, super-dense time models consist of the union of discrete sets of points in the form $\langle i, r\rangle, \langle i+1, r\rangle, \ldots$ (with the same timestamp r) and dense sets (containing an uncountable number of time points with different timestamps) of the form $\langle i, r\rangle, \langle i, r'\rangle, \langle i, r''\rangle, \ldots$ (with the same counter i). If $\langle i, r\rangle, \langle i+1, r\rangle \in T$, we say that there is a discrete step in $\langle i, r\rangle$, and we define a partial function *succ* as $succ(\langle i, r\rangle) = \langle i+1, r\rangle$ if there is a discrete step in $\langle i, r\rangle$. This notion of time guarantees that we consider traces in which time diverges and there is a finite (yet unbounded) amount of discrete steps between each couple of timed intervals. These assumptions are considered to prevent unrealistic counter-examples with impossible behaviours.

2.7.2 Trace

We provide a general notion parametrized by a given time model τ.

We define a trace as a tuple $\sigma = \langle \mathcal{M}, \tau, \overline{\mu}\rangle$ where $\mathcal{M}$ is a first-order structure, τ is a time domain and $\overline{\mu}$ is a mapping from the domain of τ to the assignment over a set of variable $V(\sigma)$ interpreted over the first-order structure $\mathcal{M}$. Given a trace σ and $t \in \tau$, we denote by $\sigma(t)$ the *state* $\langle \mathcal{M}, \overline{\mu}(t)\rangle$. We say that a trace $\sigma = \langle \mathcal{M}, \tau, \overline{\mu}\rangle$ is a *discrete trace* iff τ is a discrete time model. On the other hand, we say that a trace $\sigma = \langle \mathcal{M}, \tau, \overline{\mu}\rangle$ is a *timed trace* iff τ is a super-dense time model.

This notion generalizes the simpler notion of trace provided in the discrete setting in Section 2.2 when referring to infinite traces. Intuitively, it is possible to encode a sequence of states $s_0, \ldots$ with the trace $\sigma_D \doteq \langle \mathcal{M}, \tau_D, \overline{\mu}_D\rangle$ where $\mathcal{M}$ is the first-order structure used in the state sequence, $\tau_D = \langle \mathbb{N}, <, \mathbf{0}, v\rangle$ is a discrete time domain, $\overline{\mu}_D$ map each $i \in \mathbb{N}$ to the states of s_i. Given the set of symbols V, we say that $v \in V$ is a *discrete variable* if it can only change value during discrete transitions. Figure 2.3 shows a graphical representation of a timed trace.

2.7.3 Metric Temporal Logic

Dealing with real-time systems, the most used logic is Metric Temporal Logic (MTL)[215]. In this manuscript, we give an interpretation of MTL over infinite traces with super-

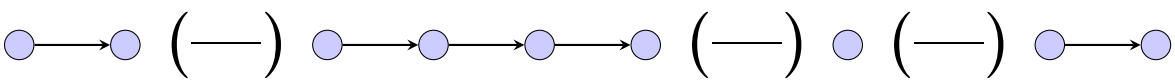


Figure 2.3: Figure showing a graphical view of a timed trace. Circles represent discrete points while the big parenthesis represent open timed intervals.

dense time domain.

This logic enriches LTL temporal operators in two ways, it annotates "until" operator with a timing interval for which it must hold, and it introduces a timed variation of $\mathbf{X}$ called *timed* next ($\tilde{\mathbf{X}}$). The intuition behind the annotated version of until is that it requires the property to be satisfied within the time interval – e.g., $\mathbf{F}_{[0,2]}p$ asks p to be true in the trace within 2 unit of time. Since we interpret formulae over timed traces (i.e. in super-dense time domain), this "bound" can be potentially infinite or even uncountable in the number of time points but still over an unbounded but finite amount of intervals. The other new feature of the logics is timed next ($\tilde{\mathbf{X}}$), such operator is true if a discrete time step is not occurring and if φ holds in the immediate timed interval. Consider Figure 2.3, initially $\tilde{\mathbf{X}}\top$ is not satisfied because there is a discrete transition; however, both in the second step and in the first open interval $\tilde{\mathbf{X}}\top$ is satisfied.

Interestingly, the interpretation of MTL with discrete trace is identical to the one of LTL[2] Intuitively, it extends until and since (for the past case) operators with intervals $\mathcal{I}$ of $\mathbb{R}_0^+$. The syntax is defined as follows.

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \tilde{\mathbf{X}}\varphi \mid \varphi\, \mathbf{U}_{\mathcal{I}}\, \varphi$$

where $p \in V$ is a *discrete* variable. Given a timed trace $\sigma = \langle \mathcal{M}, \tau, \overline{\mu} \rangle$ with a super-dense time model $\tau = \langle T, <, \mathbf{0}, v \rangle$ we inductively define the semantics of each operator over each time point $t \in T$

$\sigma, t \models \mathbf{X}\varphi$ iff	there is a discrete step from t and $\sigma, succ(t) \models \varphi$
$\sigma, t \models \tilde{\mathbf{X}}\varphi$ iff	there is no discrete step from t and there is $t' > t$, for all $t < t'' < t'\sigma, t'' \models \varphi$
$\sigma, t \models \varphi_1\, \mathbf{U}_{\mathcal{I}}\, \varphi_2$ iff	exists $t' > t$, s.t. $v(t') - v(t) \in \mathcal{I}, \sigma, t' \models \varphi_2$, and for all $t \leq t'' < t : \sigma, t'' \models \varphi_1$

where $\mathcal{I}$ is an interval of $\mathbb{R}_0^+$and p is a Boolean variable. Finally, $\sigma \models \varphi$ iff $\sigma, \mathbf{0} \models \varphi$.

[2]Considering intervals over natural, one can construct an LTL formula representing bounded operators with $\mathbf{X}$ and $\mathbf{Y}$. For intervals of the form $[0, n]$ it is sufficient to use the abbreviations like $\mathbf{F}^{\leq n}$.

2.8 HyperLTL

In Section 2.3, we introduced trace properties–properties that can be described as sets of individual system executions (traces). As discussed in Chapter 1, a key limitation of trace properties is their inability to capture relationships between multiple executions of the same system. Such relationships are essential when reasoning about security-related properties like information flow [186, 298] and diagnosability[270, 85, 49, 65].

Hyperproperties [122] address this limitation. They are defined as sets of sets of traces, enabling reasoning over multiple executions simultaneously. To determine whether a hyperproperty is satisfied or violated, one must consider multiple traces together. Typically, quantification over traces is used to express these kinds of "multi-trace" requirements in hyperproperty frameworks.

2.8.1 The Temporal Logic HyperLTL.

HyperLTL [120] is a temporal logic for hyperproperties. The syntax of HyperLTL is:

$$\alpha ::= \exists x.\alpha \mid \forall x.\alpha \mid \varphi \qquad\qquad \varphi ::= a[x] \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi \,\mathbf{U}\, \varphi$$

where x is a *trace variable* from an infinite set VAR of trace variables occurring in α. The intended meaning of $a[x]$ is that state property $a \in V$ holds at the current time in trace x. Trace quantifiers $\exists x$ and $\forall x$ allow reasoning simultaneously about different traces of the computation. Atomic predicates $a[x]$ refer to a single trace x. Given an LTL formula α, we denote by $\alpha[x]$ the formula obtained by substituting every atomic proposition a with $a[x]$.

Given a HyperLTL formula φ, we use $\mathsf{TrVars}(\varphi)$ for the set of trace variables quantified in φ. A formula φ is well-formed if for all atoms $a[x]$, x is quantified in φ. Given a set of infinite traces T, the semantics of a HyperLTL formula α is defined in terms of *pointed trace assignments*, which are partial mappings of the form $\Pi : \mathsf{TrVars}(\alpha) \rightharpoonup (T \times \mathbb{N})$. We use $Dom(\Pi)$ for the subset of $\mathsf{TrVars}(\varphi)$ for which Π is defined. Given a trace assignment Π, a trace variable x, a trace σ and a pointer p, we denote by $\Pi[x \mapsto (\sigma, p)]$ the assignment that coincides with Π for every trace variable except for x, which is mapped to (σ, p). The trace assignment with empty domain is denoted by Π_0. Also, we use $\Pi{+}n$ to denote trace assignment Π' such that $\Pi'(x) = \Pi(x) + n$ for all $x \in Dom(\Pi) = Dom(\Pi')$ where $(\sigma, p) + n = (\sigma, p + n)$. Given a partial trace assignment Π and a set of traces T,

the semantics of HyperLTL is as follows (we again omit $\vee$ and $\neg$):

$$
\begin{array}{lll}
\Pi & \models_T \exists x.\alpha & \Leftrightarrow \text{for some } \sigma \in T,\ \Pi[x \mapsto (\sigma, 0)] \models_T \alpha \\
\Pi & \models_T \forall x.\alpha & \Leftrightarrow \text{for all } \sigma \in T,\ \Pi[x \mapsto (\sigma, 0)] \models_T \alpha \\
\Pi & \models_T \varphi & \Leftrightarrow \Pi \models \varphi \\
\Pi & \models a[x] & \Leftrightarrow a \in \sigma(p),\ \text{where } (\sigma, p) = \Pi(x) \\
\Pi & \models \varphi_1 \,\mathbf{U}\, \varphi_2 & \Leftrightarrow \text{for some } j \geq 0\ \ (\Pi + j) \models \varphi_2, \text{and forall } i < j, (\Pi + i) \models \varphi_1 \\
\Pi & \models \mathbf{X}\varphi & \Leftrightarrow (\Pi + 1) \models \varphi
\end{array}
$$

Note that quantifiers assign traces to trace variables and set the pointer to the initial position 0. We say that a set of infinite traces T is a model of a HyperLTL formula φ, denoted $\Pi \models \varphi$ whenever $\Pi_0 \models_T \varphi$. We say that an STS M is a model of a HyperLTL formula φ, denoted by $M \models \varphi$, whenever $\Pi \models_{\mathcal{L}(M)} \varphi$.

A formula φ in HyperLTL is said to be quantifier alternation-free if it contains only universal quantifiers ($\forall$) or only existential quantifiers ($\exists$). We use the notation $\forall^{\cdot}\psi$ and $\exists^{\cdot}\psi$ to denote quantifier alternation-free formulae composed entirely of universal or existential quantifiers, respectively.

It is important to note that the number of quantifier alternations in a formula significantly affects the tractability of its model checking problem. Specifically, when a formula is quantifier-free, the model checking problem can be reduced in polynomial time to standard LTL model checking through self-composition techniques [20], yielding a complexity of PSPACE-complete with respect to the formula size. In contrast, when a formula includes one or more quantifier alternations, the complexity increases substantially, reaching TOWER-complete [241, 260]. Interestingly, even if the model checking problem of HyperLTL is decidable, it has been proved that the satisfiability problem in general is not decidable [169, 161].

A well-known approach for the verification of generic HyperLTL formulae is the *automata-theoretic* approach [162]. Similarly to the approach we presented in Section 2.5.3 for LTL, it reduces the problem of checking if an STS M satisfy an HyperLTL formula φ reducing to the emptiness checking problem for automaton. The downside of this approach is that it requires automata complementation every time a quantifier alternation occur, which is known to be very expensive.

In case of quantifier-alternation free formula, complementation is not needed. It is sufficient to apply self-composition [20].

Theorem 2.3 (Self-Composition [20]) *Let M be an STS and let $\varphi \doteq \mathcal{Q}x_n. \ldots \mathcal{Q}x_1.\psi$*

be a quantifier-alternation free formula and $\mathcal{Q} \in \{\forall, \exists\}$.

$$M \models \varphi \text{ iff } \bigtimes_{x_i \in \{x_1,\ldots,x_n\}} M[x_i] \models_{\mathcal{Q}} \psi_{\mathcal{Q}}$$

where $M[x_i]$ *is the STS obtained replacing each variable* $v \in V$ *with* $v[x_i]$ *in* M, $\models_{\mathcal{Q}} = \models$ *if* $\mathcal{Q} = \forall$, $\not\models$ *otherwise, and* $\psi^{\mathcal{Q}} = \psi$ *is* $\mathcal{Q} = \forall$, $\neg\psi$ *otherwise.*

2.8.2 Examples of Specifications in HyperLTL

Noninterference [186] Noninterference [186] is a foundational security policy that broadly requires that high-security inputs do not influence low-security outputs. Many variants exist, depending on the execution model or specific information flow concerns. A common formulation in HyperLTL [162] expresses that if two computation paths are identical except for high-security inputs, their low-security outputs must also be identical. For example, a basic noninterference property, where $h \in I$ is a high-security input with $I \setminus h$ be the low-security inputs and o is a public output, can be expressed as:

$$\forall x.\forall y.\mathbf{G}\left(\bigwedge_{i \in I \setminus h} i[x] = i[y]\right) \rightarrow \mathbf{G}(o[x] = o[y])$$

This formula states that for all pairs of traces $x$and y, if their low security inputs are always the same, then their output o must also be the same at all times. This ensures that changes in h alone do not cause differences in o.

Generalized Noninterference [242] Generalized noninterference relaxes the strict output determinism of basic noninterference. It allows for nondeterminism in the system's behaviour, while still enforcing that high-security inputs cannot influence low-security outputs. To express this, an existential quantifier is introduced to model the possibility of nondeterministic behaviour in the system's low-observable outputs. The property can be expressed as:

$$\forall x.\forall y.\exists z.\mathbf{G}(h[x] = h[z]) \wedge o[y] = o[z])$$

This formula states that for every pair of traces $x$and y, there exists a third trace $z$that agrees with $x$on high inputs and with $y$on outputs. In other words, it is always possible to find an output behaviour consistent with $y$that could have resulted from the high inputs of x. This captures the idea that low-observable behaviour may be

nondeterministic, but still cannot be influenced by high-security inputs in a way that would be observable by a low-level observer.

Path Planning [292] In the context of path planning, one may want to synthesize strategies that are robust with respect to uncertainty about the system's initial state.

To formally capture the notion of *initial-state robustness*, given a goal *goal*, a set of action A and an initially perturbated set of positions S_0, we can use the following *HyperLTL* formula:

$$\exists x.\forall y.S_0[x] \wedge \mathbf{F}\,goal[x] \wedge \mathbf{G} \bigwedge_{act \in A} (act[x] \leftrightarrow act[y]) \wedge (S_0[y] \rightarrow \mathbf{F}\,goal[y])$$

This formula expresses that there exists a trace that reaches the goal from an initial state in S_0, and that any other trace starting from the perturbed region S_0 will reach the same goal by performing the same sequence of actions. This ensures that the plan is *robust* with respect to variations in the initial state, as it guarantees consistent behaviour despite initial uncertainty.

Chapter 3

State of the Art

The aim of this thesis is to discuss how to improve symbolic verification in a variety of domain. To locate our contribution in the landscape of formal verification research, we need to give an overview of the current research in various aspect of verification.

In this Chapter, we provide an overview of the works covering *symbolic model checking*, *compositional reasoning*, *Linear Temporal Logics* and *Hyperproperties*. The rest of the section is organized as follows. In section 3.1, we discuss leading techniques in the context of Symbolic Model Checking, in particular applied to invariant checking and liveness checking. In Section 3.2, we discuss compositional reasoning techniques, covering approaches such as Assume-Guarantee reasoning, Contract-Based Design, circular composition, and other related methods. In section 3.3, we discuss Linear Temporal Logic and its extensions; we discuss both interesting formalisms, safety and liveness fragments, and verification techniques. Finally, in Section 3.4, we examine hyperproperties, which enable the specification of more complex requirements such as non-interference and diagnosability.

3.1 SAT/SMT based Model Checking

Symbolic model checking is a prominent technique developed to verify large-scale systems by representing their state space symbolically, in contrast to explicit-state model checking, which explores states individually. This symbolic approach enables the analysis of significantly larger systems by avoiding an exhaustive enumeration of states.

For a comprehensive overview of model checking methods, the reader is referred to [13, 118].

Invariant Checking Symbolic model checking traditionally relies on either Binary Decision Diagrams (BDDs) [80] or SAT solvers [46]. A widely adopted SAT-based method is Bounded Model Checking (BMC) [46], which incrementally unrolls the system's transition relation up to a bound k to search for counterexamples. Its key advantage lies in leveraging the efficiency and incrementality of SAT solvers, making it particularly effective for falsification tasks.

BMC can also be applied to liveness properties by encoding infinite behaviours using lasso-shaped counterexamples—paths in which a loop is formed by repeating a state after some point i. This makes BMC suitable for handling LTL specifications under infinite semantics [257]. However, while BMC excels at finding bugs, it is inherently incomplete for proving properties: the absence of a counterexample up to bound k does not imply correctness.

In principle, completeness could be achieved by determining the system's "diameter" d– the longest acyclic path–and checking the property up to this bound. In practice, however, computing such a diameter is often infeasible due to state space explosion.

To address this limitation, BMC has been extended to the K-induction algorithm [272], which generalises BMC to a sound and complete method. The technique involves two components: a base case (ensuring no counterexample exists up to depth k) and an inductive step (proving that if a property holds for k steps, it holds for the next step). Technical details are given in Section 2.6.3. K-induction has seen widespread adoption in industry [256], and has been enriched with techniques like incrementality [149], predicate abstraction [281], abstract interpretation [77], and parallelization [211].

Nonetheless, K-induction is not universally effective. Not all properties are k-inductive, and the required value of k may grow prohibitively large or even become unbounded, limiting its practical utility.

A parallel line of work explores the use of interpolation for invariant checking. McMillan [246] introduced a method based on interpolants extracted from unsatisfiable proofs, which was later extended by [289] to use sequences of interpolants for improved convergence.

A major advancement in invariant checking came with the IC3 algorithm [75, 148]. IC3 constructs inductive invariants by iteratively refining a sequence of over-approximations (called frames) and blocking bad states through inductive reasoning. The approach often outperforms traditional techniques and has been adopted by numerous state-of-the-art tools [87, 88, 277, 78, 213, 210].

Research in this area remains very active, with ongoing work on improving IC3 via variations [227, 224], and infinite-state generalisations [195, 102, 213, 210]. Addi-

tionally, several heuristics have been proposed to boost performance on industrial-scale systems [296, 205]. For an extensive historical perspective on the development of model checking techniques, we refer the reader to the recent survey [42].

Liveness Checking Another active and relevant line of research is *liveness checking*—introduced earlier in Section 2.6.4. Unlike invariant checking, which proves that something always holds, liveness checking aims to determine whether a desirable condition occurs *infinitely often* during system execution.

Traditionally, liveness properties were verified by detecting *strongly connected components* (SCCs) in the state-transition graph [13, 191]. However, such approaches relied on explicit-state exploration, and therefore suffered from scalability issues due to the exponential growth of the state space.

In recent decades, more efficient *symbolic techniques* have been developed. One notable example is the *Liveness-to-Safety* reduction [44], also presented in Section 2.6.4. This method encodes liveness checking as the search for a *lasso-shaped counterexample*—a finite path followed by a loop—indicating that a given property is eventually visited *infinitely often*.

This technique generalises loop-based Bounded Model Checking (BMC): if no such lasso is found, then (under the assumption of a finite-state system), one can conclude that $\mathbf{FG}p$ holds. A key advantage of this approach is its *simplicity*. However, it has two main drawbacks: (1) it duplicates variables, significantly increasing the size of the encoding, and (2) it is a *monolithic* approach that cannot benefit from the incremental solving capabilities typically offered by SAT/SMT solvers.

Another notable method is the *FAIR* algorithm [76], which focuses on fair path detection within symbolic encodings. The idea of fair is to look for a prefix of an infinite execution terminating in a state s, then try to find a continuation terminating in s. If the algorithm finds it, then it is a counter-example; otherwise, it blocks a generalisation of the state. The algorithm terminate with true if it is not able to find new prefixes.

A complementary strategy is provided by *K-liveness* [116], also introduced in Section 2.6.4. This technique tracks how often $\neg p$ occurs, and proves liveness by checking that this counter remains bounded under a threshold k. If so, one can deduce that $\mathbf{FG}p$ holds. In parallel, standard BMC is used to search for lasso-shaped counterexamples. One positive feature of this algorithm is that it is easy to implement. Moreover, K-liveness is particularly efficient when a *small* k is sufficient to guarantee that $\neg p$ does not persist. However, for large or unbounded k, scalability becomes a concern.

FAIR and K-liveness take *dual approaches* to the problem. Their strengths were combined in the *K-FAIR* algorithm [206], which leverages the counterexample detection power of FAIR while exploiting the proof-oriented capabilities of K-liveness. This hybrid approach leads to a more robust and flexible framework.

More recently, a novel method named *rlive* has been proposed [297]. On a high-level view, the approach is conceptually similar to *FAIR*. The algorithm progressively looks for counterexamples with invariant checks. One key difference with respect to *FAIR* is that instead of directly trying to find a lasso with a single invariant query, the algorithm iteratively keeps looking for continuation violating the prefix. If at some point it finds a state he already visited, it means that a counterexample was found; if instead it finds that the path is not extensible, it blocks the region (called shoal in the paper). This algorithm has been proved very efficient and outperforms the other approaches.

All of these techniques have been originally designed for *finite-state systems*. However, due to growing interest in *infinite-state verification*, some of these ideas have been generalised. In particular, [133] extends the Liveness-to-Safety reduction using *implicit predicate abstraction* [281], enabling its application to infinite-state models.

To our knowledge, FAIR and K-FAIR have not yet been adapted to the infinite-state setting. K-liveness, however, can be naturally extended by simply allowing symbolic counters, though this comes at the cost of *incompleteness*: in infinite-state systems, $\neg p$ may occur an unbounded but still finite number of times, which makes it impossible to conclusively verify $\mathbf{FG}p$.

An adaptation of K-liveness to *timed and hybrid systems* is presented in [103], which addresses the additional challenge of *time divergence*. Furthermore, techniques for handling *non-lasso-shaped counterexamples* in infinite-state systems have also been proposed, based on the notion of *funnel loops* [97, 95, 96], broadening the applicability of symbolic liveness checking beyond the standard finite loop model.

All these approaches must account for potentially costly infinite behaviours in order to disprove non-safety properties. This often requires sophisticated techniques such as ranking functions or complex counting mechanisms, which can significantly increase computational overhead. Henzinger introduced the notion of relative safety [194], identifying a class of non-safety properties that, under the assumption that an auxiliary property holds, can be reduced to safety properties. This concept offers a compelling characterization of practically relevant properties in verification. However, to the best of our knowledge, this notion has not yet been incorporated into practical verification algorithms.

3.2 Compositional reasoning

Compositional reasoning is a group of divide-and-conquer techniques defined to tackle the state-space explosion problem caused by the size of the problem.

In the literature, various compositional approaches have been studied [266, 218, 119, 244, 245, 221, 3]. The high-level idea is to verify smaller parts of the system and then prove a global specification exploiting a compositional proof rule.

In [244], McMillan discusses compositional verification of systems with circular dependencies for safety properties. Later, it generalised it for liveness properties of LTL [245]. In [221], Leslie Lamport introduces the Temporal Logic of Action (TLA), a variation of LTL used compositionally. The compositional approach of TLA is defined in [3] and support asynchronous composition with stuttering components (see Section 3.3 for a discussion of the semantic aspect of the logic). In [222], TLA has been extended to support a richer logic (TLA+) that allows reasoning over complex data types and permits to have a high-level abstraction in the language. A detailed overview of classical compositional works can be found in [170].

By construction various systems are inherently component-based. Therefore, in the literature various verification frameworks and approaches have been defined (e.g. [155, 92, 23, 66, 54]). The BIP [23] framework designs a compositional reasoning framework both hierarchical and asynchronous. This framework is build around the three concepts of *Behaviours* of the leaf components, *Interactions* between leaf or compound components and *Priorities* that govern the scheduling of each part of the system. The behavioural blocks are defined by transition systems and, in the timed case by timed automata while the interactions are defined by hierarchical connectors. To tackle down the verification of this system, various techniques have been proposed [30, 51, 276]. D-Finder [30] is a compositional tool for deadlock checking and the verification of safety properties in BIP systems. The approach proposed by DFINDER is compositional. It locally proves invariants and exploit an ad-hoc composition proof rule that considers component interactions to prove invariants over the composed system. On limitation of the tool is that it support only a fragment of the BIP language; in particular, it requires BIP component to interact without passing data.

In [276], a direct encoding from Timed BIP to UPPAAL [28]. In [51], a direct translation to STS over infinite state was proposed. The employed technique for the verification of BIP components used was ESST [108]; a technique that combines explicit-state techniques for the scheduler with symbolic approaches for the local components. The experimental results show that this approach outperforms D-Finder [30] still being

more expressible. We are not aware of any extension of this line of work.

Other systems such as OCRA (Othello Constraint Requirement Analysis) [92] instead consider a hierarchical system in which the composition is based on the notion of Interface Automata [135]. In this case each component can directly share variables, which results in potentially more unconstrained interactions between components, especially when interpreted asynchronously. More detail about the compositional proof system of OCRA will be provided later while discussing contract-based design.

An important technique that emerged during the years is assume-guarantee reasoning[209]. With this approach, specifications are divided into assumptions and guarantees. The assumptions represent what is expected to be guaranteed by the environment while the guarantee is the property that holds if the assumptions are satisfied. Based on assume-guarantee reasoning a prominent technique is Contract Based Design[31, 32, 259] (CBD). Historically, contracts are introduced by Meyer for software programs[248]. In that context, a contract is a couple of preconditions and postconditions on the system. In Contract-Based Design, often contracts are defined over components that can be composed and can be decomposed.

The meta-theory of contract is introduced in [33]. This theory describes abstractly way contracts and the operators to deal with them such as composition and quotient. Very recently, the operators of the algebra of contracts have been included in an ad-hoc tool to reason with contracts operators called Pacti [201].

In [126], contracts are considered as safety properties. In that case, the semantics is that the guarantee is ensured only if historically the assumption hold. This contract construction uses a "weak" assumption since there is no guarantee when the assumption becomes false. This work has been recently extended considering asynchronous systems with fixed scheduling for avionic systems[231]. In contrast, in [112], it is defined a trace-based contract proof system in which assumptions and guarantees are set of discrete or hybrid traces. To entail the correctness of the decomposition, the proof system considers two distinct checks: Implementation check and Environment check. The implementation check takes conjunction with the implications of assumptions over guarantees of each local contract to entail the global contract.

$$(\bigwedge_i (A_i \rightarrow G_i)) \rightarrow (A \rightarrow G)$$

The environment check verifies that the correct global assumptions conjoined with the

rest of the contract implementations entail the assumption.

$$\text{For all } j : (A \wedge \bigwedge_{i \neq j} (A_i \rightarrow G_i)) \rightarrow A_j$$

The approach have been used extensively in various domains [230, 12, 174, 91, 90, 66, 54] and have been implemented in the tool OCRA [92]. Contract-based techniques have also been applied to safety assessment by allowing making contract invalid when a fault occurs [67]. However, a limitation of this approach is that faults are represented strictly. A component fails from the initial point of the trace through all the execution. This means that it is not possible to represent a component functioning correctly up to a time point i and the failing (even for a limited amount of time).

One limitation of classic contract based design is that it requires to manually define the assumption and guarantees. A line of research to deal with this problem consists in the synthesis of assumptions [123, 4, 283, 255, 180, 142]. For instance, in [255, 123], the uses techniques based on the L^* learning algorithm [10]. Other ways to tackle the problem of assumption synthesis includes using abstraction refinement techniques e.g. [180]. More recent works [4, 142] extends the previous techniques permitting assumption synthesis when the composition involves circular reasoning. A generic overview of Contract-based approaches can be found in [280].

Other compositional approaches for CPS have been defined directly for Simulink and Lustre [203, 249]. Finally, in [202], a meta-theory contract-based design has also been defined in the context of Hyperproperties. This theory extends the classical meta-theory of contracts to handle hyperproperties and their more complex structure.

In [131], a compositional approach is applied to verify consensus protocols for distributed systems. The technique used in that work can reduce the verification of the asynchronous distributed system to the verification of a synchronous system. The core idea, in that case, is to decompose message sessions into communication phases. The nature of the consensus problem permits to guarantee that the synchronous system is equivalent. Compositional techniques are also being used in other domains. For instance, in [295], a compositional approach has been used for the verification of Markov Decision Processes. For example in [144], compositional verification techniques are applied to machine learning component verification. Lately, these approaches were applied to stochastic systems[192], and they are extensively used in hardware system verification[127].

3.3 Linear Temporal Logic and its extensions

Linear Temporal Logic In formal verification, properties are typically specified in temporal logics such as Linear Temporal Logic (LTL)[257]. LTL permits the expression of constraints over events and message exchanges of components. Formally, the logic is evaluated over a trace i.e. a series of assignments over the variables. LTL extends propositional logic with the temporal operator **X** (next) and the temporal operator **U** (until). LTL has been extended with past operators in [228]. That extension introduced **Y** (yesterday) and **S** (since) operators. The semantics of such operators is the same as their future counterparts except that **Y** is false in the initial state. Although LTL with operators permit simpler and succinct properties when dealing with the past, it does not increase the expressiveness of the logic[171]. A different result can be shown for succintness. In fact, it was proven that LTL extended with past is exponentially more succint than its counterpart without past [240]. Both the model checking and the satisfiability problem of LTL are known to be decidable, with PSPACE-complete complexity under both infinite and finite trace interpretations [274, 137].

A known limitation of LTL is that it cannot represent ω-regular languages. For example, in LTL it is not possible to state that a condition holds at even positions. To overcome these limitations, Property Specification Language (PSL) [1] and System Verilog Assertion (SVA) [2] were introduced. These logics permit to express omega regular languages and are extended with syntactic sugar features to be easily employable in the industry. An in-depth study of this kind of extensions of LTL has been done by Eisner and Fisman in [150]. More recent works for user-friendly specifications are the one used by NASA with FRET [184].

As described in Section 2.5.3, a standard approach for the satisfiability and Model Checking (recall that Model Checking is trivially reduced to satisfiability and vice-versa) of LTL properties is the reduction to emptiness checking of automata. In the literature there are various constructions both using explicit state and symbolic representation. A complete overview of classical automata construction and model checking of LTL was presented by Rozier and Vardi in [267]. Overall, the most mature explicit-state tool is [145]– which can also be used to ad-hoc transformations of automata e.g., determinization, minimization. However – as noted in [267] – explicit-state approaches tend to suffer significant scalability issues; moreover, they cannot be used to deal with infinite-state systems. When dealing with symbolic approaches, the automata construction (see e.g., [117, 212, 114]) generate an automaton with a property of the form $\mathbf{FG}p$ and then use liveness checking techniques to verify it [76, 206, 116, 44].

A particularly relevant fragment of LTL is its safety fragment. In [219], Kupferman and Vardi studied the verification of safety properties and introduced the notion of informative path to formally characterize computations that violate a safety property. They also proposed a PSPACE procedure to determine whether a given property is a safety property. Later, in [223], the same approach was implemented using a BDD-based algorithm. Notable syntactic fragments of LTL that are safety are SafetyLTL and full-past LTL, both of which are discussed in Section 2.5.2. Although these fragments are equally expressive [89], full-past LTL is often less intuitive for expressing properties. Nevertheless, it has a significant advantage: it allows for the direct construction of a Deterministic Finite Automaton (DFA), which is not generally possible for SafetyLTL. This ability to construct a DFA directly is particularly important in the context of the realizability problem.

Motivated by full past LTL, the authors of [93] introduced a novel safety fragment of LTL. This new fragment maintains the benefit of direct deterministic construction while offering a more natural and intuitive syntax for expressing safety properties. When considering satisfiability instead of realizability, the need for deterministic constructions is less critical. In such cases, the logic can be translated into a Non-deterministic Finite Automaton (NFA) and verified using invariant checking algorithms.

LTL can be combined with First-Order Logic to enable reasoning over complex data and systems [239, 282, 98, 101, 176]. In such extensions, LTL is augmented with functions and predicates, allowing it to express, for example, arithmetic properties [239]. In the general case, the satisfiability and model checking problems of First-Order LTL are undecidable. It is easy in fact to encode two counters by considering for example the theory of integers. The decidability of fragments has been studied for example in [181] for LTL($\mathcal{T}$) and most of the results can be ported to the LTL$_f$($\mathcal{T}$) case.

A notable example of a quantifier-free First-Order extension of LTL is LTL with event-freezing functions [282]. This logic introduces the operators $@\tilde{\mathbf{F}}$ ("at next") and $@\tilde{\mathbf{P}}$ ("at last") to capture values of variables at specific temporal positions. Specifically, the expression $var@\tilde{\mathbf{F}}\varphi$ yields the value of var at the next point in time when the formula φ holds, while $var@\tilde{\mathbf{P}}\varphi$ retrieves its value at the last time φ was true. Although originally defined for timed systems, this logic has also been integrated into the nuXmv symbolic model checker [87] with discrete-time semantics.

Building on the classic automata construction for LTL [117],in [101] it is proposed a general framework for verifying LTL($\mathcal{T}$) properties –where LTL is extended with a background SMT theory $\mathcal{T}$. A key advantage of this approach is its modularity: it is theory-agnostic and cleanly separates the construction of automata, the selection of the

liveness checking algorithm, and the use of the theory solver. A similar strategy has also been explored in the context of LTL synthesis [263, 264].

LTL has also been extended to the quantified fragment of First-Order Logic (FOL) called FOLTL, allowing for both theories and variable quantification [214]. This logic is known to be highly undecidable and in the literature most of the verification tools are restricted to the quantifier-free fragment e.g [87, 176]. A newer line of research is exploring the use of ad hoc automata to represent FOLTL specifications [177].

Finite semantics of LTL Although the standard semantics of LTL is defined for infinite traces, variations of LTL have been defined to deal with finite traces [137, 152, 151, 168, 262, 24, 111].

In [137], De Giacomo and Vardi introduce LTL_f, a finite variation of LTL with the same syntax of LTL but interpreted over finite words. While the main operators of LTL_f are the same as LTL over infinite traces, the finite nature of the trace introduces a new derived operator $\mathbf{N}$(weak next). Intuitively, $\mathbf{N}\phi$ holds if either there is not a next state or ϕ holds in the next state. One of the advantage of LTL_f with respect to the infinite trace semantics of LTL is its simpler verification. As for safety fragments of LTL, to verify a formula it is not required to reduce to liveness algorithms. Another interesting aspect of LTL_f, is that the logic has more convenient tractability when dealing with synthesis [302, 301, 138]. The idea is that – as for safety fragments of LTL– it is possible to reduce the problem to parity games leveraging the reduction from LTL_f to DFA [138], which are far more tractable than LTL games. The area of LTL_f synthesis is very active with an annual track at the annual synthesis competition *SYNTCOMP* [207] has a dedicated track for LTL_f synthesis.

As a negative side, LTL_f is not a natural formalism for model checking. Intuitively, properties like *response* (e.g., $\mathbf{G}(i \rightarrow \mathbf{F}o)$ are satisfied by an STS only when o is valid if and only if i holds.

Another relevant problem is satisfiability of LTL_f. In [136], it is shown a simple reduction from LTL_f to LTL by means of rewriting; the idea is to rewrite the temporal operators with an additional *end* variable that is true at the end of the finite trace. This rewriting corrects previous approaches that left the formula unchanged, interpreting it over infinite traces that eventually stutter forever. This paper shows that this approach is in general incorrect, but it holds over a significant pattern of relevant formulae. It also introduces an automata-based construction for LTL_f which directly reduces to Non-deterministic Finite Automata.

More recent developments produced direct reductions to SAT solving [225, 164].

In [164], it is shown that various fragments of LTL_f can be checked with NP-COMPLETE complexity instead of the general case of PSPACE-complete. This work leverages these theoretical results to provide a more efficient SAT reduction. In [225], a Conflict-Driven LTL_f satisfiability checking technique has been proposed. The idea is to tailor a CDCL SAT like approach to LTL_f. Finally, in [178], a reduction to SAT is done via tableaux construction. Later, the tableaux construction has also been extended to deal with a First Order extension of LTL_f [176]. Interestingly, that work also shows that First Order LTL_f is semi-decidable, even though, its infinite trace extension is not.

In [152], 3 semantics are studied for LTL: weak, neutral and strong semantics. The weak semantics considers any formula true when the trace is over. On the contrary, the strong semantics evaluates the formula as false when the trace is over. Finally, neutral semantics considers a combination of both semantics. The logic semantics was designed to represent bounded aspects of verifications such as BMC or Runtime Verification. Indeed, a similar semantics has been used in the context of Bounded Model Checking of hyperproperties with synchronous and asynchronous semantics [199, 198]. Recently, in [168], an extension of the truncated semantics was proposed for Signal Temporal Logics [237] with applications in the context of system biology.

A slightly less expressive finite variation of LTL is Mission Time LTL (MLTL) [262] which is used in the aerospace domain to represents bounded scenarios. This logic employs a bounded semantics of LTL by disallowing **U** and all the abbreviated variants of it. The logics allow to reason only with bounded operators e.g.$\mathbf{F}^{\leq p}$ which can be reduced to simply using **X**. A simple reduction to LTL_f satisfiability has been presented in [226]. Another simpler reduction to regular-expressions (without Kleene star) has been more recently proposed for MLTL [156, 294].

Other finite LTL semantic variation come from Runtime Verification [21]. The three-valued variation of LTL (LTL_3) [24] extends the evaluation of an LTL formula considering a third truth value "unknown". Another extension of finite semantics of LTL related to this dissertation considers 4 truth values [111]. The overall idea is that – given a property with given assumptions – the four possible monitor results are the following: (i) the current finite trace cannot be extended to an infinite trace satisfying the assumption, meaning that the execution is "out of model"; (ii) the trace is not out of model and all suffixes of the trace satisfying monitor can check whether all possible extensions (assuming that the assumption formula holds) are true. (iii) the trace is not out of model and all suffixes satisfying the assumption violate the property under observation. (iv) the trace is not out of model, and it is not possible to determine whether the property is satisfied or violated yet.

Asynchronous variations of Linear Temporal Logics Due to the importance of asynchronous systems, in the literature there were various extensions and applications of LTL to deal with asynchronous semantics. The most famous one is the Temporal Logics of Action [221] (TLA) that enables component based reasoning and introduces a variation of always **G** with implicit stuttering of a subset of variables. TLA divides systems into components and defines each component's behaviour as a property. TLA provides an implicit notion of stuttering: when a component does not run, a subset of its variables defined in the property does not change. Thus, this approach simplifies the composition of the logic since data persistency is granted by the language itself.

Another relevant work LTL over asynchronous systems is the Temporal Clock operator for LTL [154]. It extends LTL with a clock operator that restrict the evaluation of an LTL subformula to the points in which a Boolean variable – denoted as "clock" – is true. This clocked view can be seen as a projection of the trace over some local position. Similar views of projection in temporal logics have also been recently employed in the context of asynchronous hyperproperties [72] (which is discussed in depth in Section 3.4). This logic can be verified via a simple rewriting to LTL; indeed, the logic is known to have the same expressiveness of LTL. Contrary to TLA, this work does not provide any compositionality, and it does not assume the stuttering of variables. However, this generality permits the expression of local properties of components with shared variables using the clock operator to represent the scheduling of a local component. A similar formalism has been proposed in [299] for the verification of multi-clock systems – distributed systems that are not synchronized and are locally managed using logical clocks and relating clocks via a clock calculus [238]. A shallower relation can be found between the clock operator and the more recent at next operator ($@\tilde{\mathbf{F}}$) [282]. Both operators reason over satisfaction over relevant interesting points. In [282], the evaluation of the operator ($@\tilde{\mathbf{F}}$) is limited to the "next" value while in [154] the evaluation is local to a complete formula. On the other hand, the framework in [282] allow reasoning over theories and first-order extensions of LTL while in [154] the framework is limited to the propositional case.

Metric Temporal Logic Dealing with real-time systems, the most used logic is Metric Temporal Logic (MTL)[215]. This logic enriches LTL temporal operators with bounds to constraint time intervals in which the formula must hold. It extends until and since (for the past case) operators with intervals of $\mathbb{R}_0^+$. This logic is strictly more expressive than propositional LTL. For example, it can express the property "Always when input is true, the output will be true in at most 5 units of time" ($\mathbf{G}(input \rightarrow$

$\mathbf{F}_{[0,5]}(output)$) while LTL cannot.

A prominent and simpler fragment of MTL is Event Clock Temporal Logic (ECTL) [261] which restricts MTL considering only expressing that some event will occur within a given time or after a given time. The logic is decidable and represents the logical counterpart of Event Clock Timed Automata [8].

Another temporal logic extending LTL with time is Timed Propositional Temporal Logic (TPTL)[9]. It extends LTL with explicit time reasoning and thus, allows time to be compared similarly to the work in [98]. MTL have also been extended with parametrized bounds in [141].

Other logics consider time domains where also variables can be continuous; thus, they have to deal with derivatives and complex dynamics e.g., [110, 237]. Such logics are in general harder to verify and their domain of application is Cyber-Physical Systems.

MTL and the other logics dealing with explicit time consider time as a global monotonic variable. However, in real-time distributed systems a model in which time is local to the components is far more realistic. Distributed Event Clock Temporal Logic (DECTL)[253] is a logic that extends ECTL considering local skewed clocks instead of time. This representation of the logic is more realistic since it considers the possibility that clocks might grow at a different rate. In [252], the timed modal logic ML_ν is introduced. In that work, the system was composed of a network of components with skewed clocks. The same occurs in [291], where TPTL is extended considering distributed systems with skewed clocks.

These works come with a limitation ([291, 253, 252]): they do not take into account clock resets. The problem of resets has been studied in [265], where the impact of clock synchronization have been analyzed in the context of patterns of timed automata. The major problem of resets is that it breaks the assumption of time monotonicity. For example, when a clock is synchronized with a value lower than its current. In [86], the problem of non-monotonicity of time is studied for MTL and TPTL over non-monotonic words. That work shows the undecidability of such logics but identifies a decidable fragment of them.

3.4 Hyperproperties

Although LTL and its variations can express relevant properties and behaviours of systems and components, their expressiveness is limited to properties over a single "execution" i.e. an LTL formula represents a set of traces. This limitation does not permit to reason over information flow properties such as *non-interference*[186], *Observation*

determinism [298], *non inference* [243] and *diagnosability* [270, 65, 49, 85]. To deal with these kinds of challenges two different formalisms have been proposed: hyperproperties [122] and Epistemic Temporal Logics [190, 157]. Both formalisms are able to reason over multiple executions of the same systems. Epistemic Temporal Logics considers a knowledge operator **K** that introduces an implicit quantification over the traces. On the other hand, hyperproperties usually achieve that through an explicit quantification over traces. For what regards Epistemic Temporal Logics, works have studied both information-flow problem [14] and the diagnosability problem in various forms [65, 173]. Other works in the context of diagnosability employ ad-hoc techniques such as and looking for ribbon-shaped paths [70, 49]. Recently, the problem of diagnosability has also been studied in [300], where the authors introduced various patterns in HyperLTL [120] and considers also weak variations of diagnosability [85] that require additional existential quantification.

In [71], the author compare HyperCTL* [120] with KCTL* (the epistemic extension of CTL*) and show that the two logics have different expressiveness, and, to fix that, they propose an extended formalism that generalise both under synchronous semantics.

In the realm of model checking for finite-state reactive systems, several temporal logics for hyperproperties have been introduced [143, 120, 71, 260, 161, 124, 188], many of which ensure decidable model checking. Notable examples include HyperLTL[121], HyperCTL*[121], HyperQPTL[260, 124], and HyperPDL$-\Delta$[188]. These logics extend classical temporal frameworks—LTL, CTL*, QPTL[275], and PDL[166], respectively– by incorporating explicit first-order quantification over traces, enabling reasoning over multiple execution traces simultaneously.

All of these logics adopt a synchronous semantics, where temporal modalities are evaluated in lockstep across the traces bound to quantified trace variables. Additional synchronous hyper logics have been developed through alternative formalisms, such as hyper variants of monadic second-order logic over traces or trees [124], and via team semantics applied to standard temporal logics like LTL [216, 234, 288].

Within the linear-time setting, the logic S1S[E][124] (and its first-order fragment FO[<,E][163]) extends the classical monadic second-order logic of one successor (S1S) by introducing the equal-level predicate E, which allows comparison of positions across different traces. More recently, an extension of HyperLTL incorporating second-order quantification over traces has been proposed [40], enabling the expression of common knowledge in distributed multi-agent systems. However, similar to S1S[E], this extension results in highly undecidable model checking problems [40].

Hyperproperties have also been defined for other extensions of LTL. In [5], prob-

abilistic hyperproperties (HyperPCTL, HyperPCTL*) that extend respectively PCTL and PCTL* with trace quantification. In [64], hyperproperties have also been defined for metric temporal logic. In that work, the authors restricted the verification of HyperMTL to the verification of untimed HyperLTL. Similar works have also been proposed for timed hyperproperties considering parametric timed extensions of CTL [290] and hyper extensions of STL [250]. Finally, in [182], a variation of HyperLTL for finite traces was introduced; to deal with the issue of having traces of different lengths, this work extends the shorter trace with padding transition.

Asynchronous Hyperproperties Hyper logics supporting asynchronous features have been introduced recently [189, 26, 72]. These logics allow relating traces at distinct time points which can be arbitrarily far from each other. Asynchronicity is ubiquitous in many real-world systems, for example, in multithreaded environments in which threads are not scheduled lockstepwise, and traces associated with distinct threads progress with different speed. Asynchronous hyperproperties are also useful in information-flow security and diagnosability settings where an observer cannot distinguish consecutive time points along an execution having the same observations. This requires to match asynchronously sequences of observations along distinct execution traces. The first systematic study of asynchronous hyperproperties was done by Gutsfeld et al. [189], who introduced the temporal fixpoint calculus H_μ and its automata-theoretic counterpart for expressing such properties in the linear-time setting.

More recently, three temporal logics [26, 72], which syntactically extend HyperLTL, have been introduced for expressing asynchronous hyperproperties: the logic *Asynchronous HyperLTL* (A-HyperLTL) [26] and *Stuttering HyperLTL* (HyperLTL$_\mathsf{S}$) [72], both useful for asynchronous security analysis, and *Context HyperLTL* (HyperLTL$_\mathsf{C}$) [72], useful for expressing hyper-bounded-time response requirements. A-HyperLTL, which is expressively incomparable with both HyperLTL and HyperLTL$_\mathsf{S}$ [72], models asynchronicity by means of an additional quantification layer over the so called *trajectories* which control the relative speed at which traces progress by choosing at each instant which traces move and which traces stutter. On the other hand, the logic HyperLTL$_\mathsf{S}$ exploits relativized versions of the temporal modalities with respect to finite sets Γ of LTL formulae: these modalities are evaluated by a lockstepwise traversal of the subtraces of the given traces which are obtained by removing "redundant" positions with respect to the pointwise evaluation of the LTL formulae in HyperLTL. On the same line, in [37], the ObsHyperLTL logic is defined. This logic can extend trace quantification with local predicates that mark relevant points in the trace. This logic is

decidable and resembles the decidable fragment of HyperLTL$_\mathsf{S}$. A comparative study on the expressiveness of these logics has been introduced in [73].

Finally, in [22], the formalism of Hypernode Automata is introduced. This formalism considers both finite traces and asynchronous systems. However, the formalism is conceptually different from the logics presented above. In particular, it expressively incomparable with H_μ, which is known to subsume HyperLTL$_\mathsf{S}$ and HyperLTL$_\mathsf{C}$. The formalism allows defining relevant asynchronous hyperproperties and has decidable verification problem.

Model Checking of HyperLTL The HyperLTL model checking problem has been studied following various approaches. In [120, 162] the authors define an automata-based construction for HyperLTL. For the quantifier-alternation free fragment of HyperLTL, this approach reduces to self-composition of the system [20] and LTL model checking. The self-composition approach has been successfully implemented in MCHyper [162], which reduces the verification problem to the efficient technique symbolic model checking. An advantage of this approach is that it can indirectly benefit from any improvement in the verification of trace properties. For instance, when considering HyperLTL model checking without quantifier alternation. One can apply composition and then use state-of-the-art model checking approaches for LTL such as *rlive* [297]. In contrast, AutoHyper [38] implements the automata-based construction using an explicit state approach via language inclusion using inclusion checkers such as Spot [145]. AutoHyper can handle in general any type of HyperLTL formula, independently of the amount of quantifier alternations or temporal nature of the formula. However, due to its explicit state nature, AutoHyper does not scale very well with the size of the system.

Another line of work for verifying HyperLTL was introduced by Coenen et al. [125], who proposed a game-based approach for proving $\forall\exists$ properties using strategies. In this technique, verification is framed as a game between a universal trace and an existential trace. The strategy for the existential trace can either be provided by the user or synthesized automatically, which in [125] is searched using a bounded approach. Because the existential trace is allowed to observe the entire execution of the universal trace, the method requires the use of prophecy variables to anticipate future behaviours. This approach was implemented in an extended version of MCHyper. Subsequently, [36] shows that completeness of this technique can only be guaranteed when using ω-regular prophecy variables. The authors provide a constructive proof of this result and implemented the improved technique in the tool HyPro [36]. The approach has also been applied to asynchronous hyperproperties in [37], where predicate abstraction was used to

handle infinite-state systems. The game-based verification method was adapted to synchronize traces at designated observation points, enabling reasoning over asynchronous executions. Moreover, in this setting, the approach applied predicate abstraction using a given set of predicates, without providing an automated CEGAR procedure to refine them.

In [199], the authors introduce a Bounded Model Checking (BMC) technique for HyperLTL, extending the classic BMC approach [45] to support quantified formulae using QBF solvers. Currently, this technique is implemented in a tool called HyperQB. Unlike the original BMC framework, this method does not support looping conditions and instead adopts a finite semantics akin to the truncated semantics used in LTL [152]. This occurs because looping conditions with quantifier-alternation become very quickly intractable via QBF solving. Therefore, in [200], the authors tackle down this limitation by means of induction. They focus on proving validity of $\forall\exists.\mathbf{G}\phi$ property and $\exists\forall.\phi$ properties through inductive simulation reasoning. The approach tries to check if a form of inductive simulation holds. This approach resembles an adaptation of the proof obligation of the inductive invariants in the context of hyperproperties. This line of work has also been extended to deal with asynchronous hyperproperties [198]. The target logic in that case was A-HLTL [27].

Recent advances have introduced specialized techniques to address the verification of software systems. In [129], symbolic execution is leveraged to efficiently discover counterexamples to asynchronous $\forall\exists$ hyperproperties in software. Other approaches extend classical Hoare logics to reason about $\forall\exists$ hyperproperties as well [34, 134].

In [284], an extended form of Constraint Horn Clauses (CHCs) [50] was proposed for the verification of relational properties—specifically, $\forall\exists$ hyperproperties–over software programs. A more recent advancement is presented in[204], where CHCs are applied to verify ObsHyperLTL [37]. This CHC-based technique has demonstrated improved efficiency compared to traditional game-based approaches. As such, CHCs represent a promising direction for future research in the verification of HyperLTL.

Another line of work investigates the use of abstraction for the verification of Hyperproperties with quantifier-alternation [187]. The idea is that, while classical works for trace properties can apply over-approximations of the system (e.g. [281]), in this case it is required to apply both over-approximation and under-approximation at the same time.

Part II

Verifying Trace Properties

Chapter 4

Verifying LTL modulo theory over finite and infinite traces

In Chapter 1, we outlined the key challenges of verifying complex systems, especially in the presence of rich data domains and intricate temporal properties. This chapter builds upon that foundation by extending classical model checking techniques for Linear Temporal Logic (LTL) to the more expressive and demanding setting of LTL modulo background theories (LTL($\mathcal{T}$)). Our aim is to adapt and generalize symbolic automata-theoretic approaches to handle both infinite and finite traces in this context, while also addressing scalability. To this end, we introduce a specialised invariant-based algorithm that supports fragments beyond standard safety properties, leveraging the notions of relative safety and relative liveness. This contribution forms a step toward a unified, efficient, and theory-aware verification methodology. To situate our contribution within the broader verification landscape, we recall the key ideas behind temporal logics and symbolic model checking. This overview serves to highlight the challenges and opportunities that arise when extending these classical techniques to the richer setting of LTL modulo theories.

Temporal logics are a fundamental area of computer science, for their ability to represent properties of the behaviours of complex dynamical systems. Application fields include requirements analysis [52, 109], formal verification [13], reactive synthesis [53, 160, 94], agent-based systems [167], planning [84], contract-based design [112], robotics [158], runtime verification [25, 111], and test case generation [197]. In the field of design of complex systems (e.g. hardware, software and cyberphysical), key problems are satisfiability, validity, and model checking of temporal logic formulae. Satisfiability and validity allow checking the properties of requirements formalized as temporal

formulae, and to prove contract refinement. Model checking is used to show that a system, modeled as a transition system M, satisfies the requirements, formalized as a temporal formula φ. The paradigm of Linear Temporal Logic (LTL), originally proposed in [257], has become over the years a de facto standard [285], if compared to branching time temporal logics [83]. In the seminal works on finite-state model checking, LTL is propositional, and interpreted over infinite traces. The basic idea in LTL model checking is to tackle the problem $M \models \varphi$, where M is a transition system and φ an LTL formula, by checking whether all (infinite) execution traces of M satisfy φ. The automata-theoretic approach provides an elegant solution to reduce reasoning about LTL to the fair reachability problem [179]. This is done by constructing an automaton $M_{\neg\varphi}$ that accepts the traces violating φ. If the language of the product $M \times M_{\neg\varphi}$ is not empty, we have a counterexample to the original problem. We assume that $M_{\neg\varphi}$ is a degeneralized Büchi automaton, i.e. it has a single acceptance condition $\neg q$, sometimes referred to as the fairness condition.

One of the main advantages of the automata-theoretic approach is that one can focus on conceiving specialised algorithms for the pattern $\mathbf{FG}q$, which is the dual of fair reachability. Thus, for example, efficient algorithms such as the nested depth-first search of [196, 130] have focused on finding or proving the absence of (infinite) traces that visit $\neg q$ infinitely often.

We focus on a symbolic verification paradigm: following [117], both the system model and the automaton are represented as symbolic transition systems described by means of logical formulae. Popular approaches include Bounded Model Checking [46], liveness to safety [44], and k-liveness [116] (refer to Section 2.6 for a detailed overview of these approaches).

Several extensions and variants of LTL have recently gained significant interest. First, we consider LTL interpreted over *finite traces*, as in LTL_f [137, 302, 11]. This interpretation has the advantage that the analysis can be restricted to sets of finite traces (albeit of unbounded length), in contrast to the search for counterexamples of infinite length. Thus, the model checking problem $M \models_f \varphi$ is focused on proving that all finite traces of M satisfy φ. The automata-theoretic approach in this case builds an automaton $M_{\neg\varphi}$ over finite traces, with a final condition $\neg q$ so that $M \models_f \varphi$ if and only if $\neg q$ cannot be reached in $M \times M_{\neg\varphi}$. This is equivalent to the LTL_f pattern $\mathbf{G}q$ and analogously to the case on infinite traces, specialised algorithms for invariant checking can be used.

Another interesting semantics presented over finite traces is the one proposed by Eisner and Fisman in [152, 168], which provides a three-view semantics in which finite

traces can be interpreted weakly, neutrally or strongly. The weak semantics resembles the idea that a violation has not been found yet. Intuitively, this semantics corresponds to the semantics of bounded model checking when no loops are involved.

In general, the interpretation over finite traces is closely related to the semantic notion of safety languages [219], defined under the infinite trace interpretation, where a formula can be decided by simply looking at finite prefixes. The identification of syntactic safety fragments of LTL is particularly relevant because it opens up the possibility of using invariant checking instead of liveness checking algorithms, hence completely disregarding the notion of fairness. In [194], Henzinger introduced the notion of relative safety and relative liveness, which extend the standard definitions considering an assumption property. For example, a property φ is safety relative to an assumption α if any counterexample to φ satisfying α has a bad prefix. This is the case for example of a bounded response property that states that every request is followed by a response within δ time units: this property is safety relative to the assumption that requires δ time units to always eventually elapse (i.e., avoiding Zeno executions).

Second, we consider the case of LTL *modulo theories*, LTL($\mathcal{T}$) [100], where atoms are not limited to Boolean variables. Rather, temporal formulae are built over first-order atoms interpreted with respect to a background theory, following the Satisfiability Modulo Theories [18] paradigm. The problem is particularly relevant and challenging when the first-order atoms can constrain the current and next value of individual variables like in the formula $G(v' > v)$, where v' represents the next value of v. This opens up the possibility to model transition systems with state defined by arrays, integer- and real-valued variables, and complex, nonlinear dynamics, and to represent timed automata, hybrid automata, software, and their combinations. We remark that in the infinite-state case, differently from the finite state case, when a property is violated there is no guarantee that a lasso-shaped counterexample $\alpha \cdot \beta^{\omega}$ exists. This makes the reduction to invariant checking even more appealing, when possible, because it avoids the analysis of infinite traces, and supports the use of practically effective invariant checkers such as [102, 195, 213, 210].

This chapter details the model checking of trace properties over finite and infinite traces by adapting well-known automata constructions for LTL [117] and safetyLTL [219] to LTL modulo theory. Moreover, we generalize safety reductions to deal with commonly used semantics of finite traces of LTL_f [137] proposed by De Giacomo and Vardi Finally, we introduce the model checking of fragments larger that SafetyLTL via a reduction to invariant exploiting the notion of relative safety and relative liveness – both defined by Henzinger in [194]. First, a direct reduction is given assuming specific struc-

ture of the formula and the automaton; then, we also show a loop-based construction to deal with relative-safety LTL($\mathcal{T}$) formula in a generic context.

Contributions In summary, the contributions of this chapter are the following:

- A unified view of the automata construction for LTL modulo theory with both finite and infinite semantics.
- An open-source Python library that reduces LTL($\mathcal{T}$) to liveness and LTL$_f$($\mathcal{T}$) to safety, with benchmarks from the literature covering both logics in VMT-LIB format.
- An algorithmic reduction of non-safety properties to invariant checking exploiting the notion of relative safety [194].

The contributions presented in the first part of this chapter (Section 4.1 and Section 4.2) are an extended version of the work presented in [57] conjoined with the definition of truncated weak semantics adapted from [62].

For what regards the second part (Section 4.3), most of the contribution presented have been introduced in [58]. Some minor results have been introduced in this thesis.

The work presented in this chapter is the result of a collaboration with Alessandro Cimatti, Alberto Griggio, Stefano Tonetta and Marco Zamboni. The first part of this work was a joint collaboration with Alessandro Cimatti, Alberto Griggio and Stefano Tonetta; the PyVMT tool [258] was already under development, and the author of this thesis only marginally contributed to its implementation. The second part of this chapter was mainly done by the author in collaboration with Alessandro Cimatti, Stefano Tonetta and Marco Zamboni, which contributed in the implementation of the automata-based construction in nuXmv [87].

Outline The rest of the chapter is organized as follows. In Section 4.1, we present LTL modulo theory over finite and infinite traces; in Section 4.2, we present an adaptation of the automata-theoretic approach of LTL and LTL$_f$ to LTL($\mathcal{T}$) and LTL$_f$($\mathcal{T}$); finally, in Section 4.3, we study the notion of relative safety [194] for fragments of LTL($\mathcal{T}$) to improve model checking scalability.

4.1 LTL Modulo-Theory over infinite and finite traces

In this section, we consider LTL [257] extended with past operators [229] as well as first-order atomic formulae. The language LTL($\mathcal{T}$) is parametrized by a first-order theory $\mathcal{T}$. Given a set of variables V, the atomic formulae are built over the predicate, function, and constant symbols of $\Sigma_{\mathcal{T}}$ and the variables in $V \cup V'$, where V' represents the value of V in the "next state", after an execution step, as formalized below in the semantics subsection.

Formally, we define LTL($\mathcal{T}$) extending the propositional definition of Section 2.5 with possibly finite semantics and to support predicates, and terms.

Definition 4.1 *(LTL($\mathcal{T}$) Syntax) Given a first-order theory $\mathcal{T}$ and a set of variables V, LTL($\mathcal{T}$) formulae φ are defined by the following syntax:*

$$\begin{aligned}\varphi :=& \top \mid \bot \mid \mathit{Pred}(u, \ldots, u) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \boldsymbol{X}\varphi \mid \varphi \,\mathbf{U}\, \varphi \mid \boldsymbol{Y}\varphi \mid \varphi \,\mathbf{S}\, \varphi \\ u :=& \mathit{const} \mid v \mid \mathit{func}(u, \ldots, u) \mid v'\end{aligned}$$

where Pred is a first-order predicate, func is a first-order function, const is a constant value of the theory, v is a variable and v' is the function counterpart of $\boldsymbol{X}$ also denoted as next.

The logic LTL$_f$($\mathcal{T}$) has the same syntax as LTL($\mathcal{T}$). When $\mathcal{M}_{\mathcal{T}}$ is empty and V is a set of Boolean variables, the logic coincides with the usual propositional case introduced in Section 2.5.

To avoid repetitions, we give a uniform definition of the semantics of LTL and LTL$_f$ given a trace in $\mathcal{L}(V) \cup \mathcal{L}_{fin}(V)$. To do so, we introduce the following notation. Given $i+1 < |\sigma|$ (σ_i is not the last state), we define $\sigma^+(i)$ as the assignment $\sigma(i) \cdot \sigma'(i+1)$. If $i+1 = |\sigma|$ ($\sigma(i)$ is the last state), we define $\sigma^+(i)$ as the assignment that assigns every variable $v \in V$ to $\sigma(i)(v)$ and every variable $v' \in V'$ to a default value d in the domain of $\mathcal{M}$.

In the following, we define the semantics of LTL($\mathcal{T}$) considering as well its finite semantics (LTL$_f$($\mathcal{T}$)).

Definition 4.2 (LTL($\mathcal{T}$) and LTL$_f$($\mathcal{T}$) Semantics) *Given a structure $\mathcal{M}$ and a trace $\sigma \in \mathcal{L}^{\mathcal{M}}(V) \cup \mathcal{L}_f^{\mathcal{M}}(V)$, the semantics of a formula φ is defined as follows:*

- $\sigma, \mathcal{M}, i \models \top$
- $\sigma, \mathcal{M}, i \not\models \bot$

- $\sigma, \mathcal{M}, i \models \mathit{Pred}(u_1, \ldots, u_n)$ *iff* $\langle \sigma^+(i), \mathcal{M} \rangle \models \mathit{Pred}(u_1, \ldots, u_n)$
- $\sigma, \mathcal{M}, i \models \varphi_1 \wedge \varphi_2$ *iff* $\sigma, \mathcal{M}, i \models \varphi_1$ *and* $\sigma, \mathcal{M}, i \models \varphi_2$
- $\sigma, \mathcal{M}, i \models \neg\varphi$ *iff* $\sigma, \mathcal{M}, i \not\models \varphi$
- $\sigma, \mathcal{M}, i \models \varphi_1 \mathbf{U} \varphi_2$ *iff there exists* $k, i \leq k < |\sigma|$, *such that* $\sigma, \mathcal{M}, k \models \varphi_2$ *and for all* $l, i \leq l < k, \sigma, \mathcal{M}, l \models \varphi_1$
- $\sigma, \mathcal{M}, i \models \varphi_1 \mathbf{S} \varphi_2$ *iff there exists* $k, 0 \leq k \leq i$, *such that* $\sigma, \mathcal{M}, k \models \varphi_2$ *and for all* $l, k < l \leq i, \sigma, \mathcal{M}, l \models \varphi_1$
- $\sigma, \mathcal{M}, i \models \boldsymbol{X}\varphi$ *iff* $i < |\sigma|$ *and* $\sigma, \mathcal{M}, i+1 \models \varphi$
- $\sigma, \mathcal{M}, i \models \boldsymbol{Y}\varphi$ *iff* $i > 0$ *and* $\sigma, \mathcal{M}, i-1 \models \varphi$

The interpretations of terms $\sigma^{\mathcal{M}}(i)$ *is defined as follows:*

- $\sigma^{\mathcal{M}}(i)(\mathit{const}) = \mathit{const}^{\mathcal{M}}$
- $\sigma^{\mathcal{M}}(i)(v) = s_i(v)$ *if* $v \in V$
- $\sigma^{\mathcal{M}}(i)(\mathit{func}(u_1, \ldots, u_n)) = \mathit{func}^{\mathcal{M}}(\sigma^{\mathcal{M}}(i)(u_1), \ldots, \sigma^{\mathcal{M}}(i)(u_n))$

Finally, we have that $\sigma, \mathcal{M} \models \varphi$ *iff* $\sigma, \mathcal{M}, 0 \models \varphi$.

$\mathrm{LTL}(\mathcal{T})$ and $\mathrm{LTL}_f(\mathcal{T})$ formulae are evaluated by a structure $\mathcal{M} \in \mathcal{M}_{\mathcal{T}}$ and respectively infinite and finite traces. We use the standard abbreviations defined in Section 2.5.

We remark that the choice of evaluating the next value of a variable on the last state of a trace to a default value is arbitrary, and any other choice has pros and cons. To avoid ambiguity, it is better to guard the use of next variables with $X\top$ so as to enforce the existence of the next state (e.g., $\mathbf{G}(\mathbf{X}\top \rightarrow v' = \mathit{const})$).

In the following Section, we will use the following rewriting to ensure that the next variables occur in this form. Let $\mathit{norm}(\varphi)$ be the formula obtained from φ by substituting every atomic formula p with $(\mathbf{X}\top \wedge p) \vee (\neg\mathbf{X}\top \wedge p[V'/d])$, where $p[V'/d]$ means that every occurrence of $v' \in V$ in *Pred* is replaced by the default value d.

Theorem 4.1 *φ and $\mathit{norm}(\varphi)$ are equivalent.*

Proof: Let σ be a trace $\in \mathcal{L}^{\mathcal{M}}(V) \cup \mathcal{L}^{\mathcal{M}}_{fin}(V)$ where $\mathcal{M}$ is a first-order structure.

For all $0 \leq i < |\sigma| : \sigma, i \models \phi \Leftrightarrow \sigma, i \models norm(\phi)$.

The statement easily entails the theorem. We prove it by induction on the structure of the formula. We ignore the cases with infinite trace because they trivially hold ($\mathbf{X}\top$ is always true for infinite traces).

Base case: $\psi = p$.

$\sigma, \mathcal{M}, i \models norm(p) \Leftrightarrow \sigma, \mathcal{M}, i \models (\mathbf{X}\top \wedge p) \vee (\neg\mathbf{X}\top \wedge p[V', d])$. We consider two cases: if $i = |\sigma| - 1$ and $i < |\sigma| - 1$. In the first case $\sigma, \mathcal{M}, i \models norm(p) \Leftrightarrow \sigma, \mathcal{M}, i \models p[V', d]$. Since i is the last point of the trace, the assignment of the dotted variable is provided using the default d value for each variable; therefore, $\sigma, \mathcal{M}, i \models norm(p) \Leftrightarrow \sigma, \mathcal{M}, i \models p$.

If i is not the last state then $\sigma, \mathcal{M}, i \models \mathbf{X}\top$ which simplifies the formula to p itself completing the proof.

Inductive case: all the inductive case follows from trivial application of induction. □

Weak Truncated Semantics We now introduce an alternative semantics for LTL($\mathcal{T}$) based on the truncated semantics of LTL defined in [152, 168]. This semantics is tailored for both finite and infinite traces. For what regards infinite traces, the semantics is identical to the standard one. When we reason over finite traces, the predicates are interpreted "weakly" i.e. all the predicates are evaluated as true at the end of the trace (position $|\sigma|$). This kind of semantics is useful when reasoning of prefixes of executions such as in Bounded Model Checking of Runtime Verification (see Chapter 3 for a more in depth discussion of finite semantics of LTL).

Another important difference between the standard semantics of LTL$_f$, LTL$_f(\mathcal{T})$, and the truncated semantics lies in the negation. When a formula is negated the interpretation passes from *weak* to *strong* and vice versa. Therefore, a negated formula $\neg\phi$ is satisfied with weak (strong) semantics if and only if ϕ is not satisfied with strong (weak) semantics. This means for instance that a proposition P is evaluated as *true* at the end of a trace (position $|\sigma|$) independent of the polarity of its variables. These different semantics are denoted as $\models_{t^-}$ for weak semantics and as $\models_{t^+}$ for strong semantics; moreover, we use $\models_{t^{sgn}}$ to refer to both interpretations. For what regards the difference with LTL$_f(\mathcal{T})$, LTL$_f(\mathcal{T})$ requires finite traces to satisfies eventualities i.e. $\sigma \models_f \mathbf{F}p$ iff at some point in the trace p is satisfied while with truncated semantics $\mathbf{F}p$ is always satisfied.

Since the original Truncated LTL did not contain past operator, we introduce them here; the semantics of $\mathbf{Y}$ with truncated semantics is a bit subtle because it must take

into account the fact that the formula can be interpreted outside the range of the trace. The original truncated semantics guarantees that, any formula evaluated after $|\sigma|$ is true. However, if we evaluate $\mathbf{Y}p$ at position $|\sigma|$ considering the standard semantics, we would lose that property. Therefore, in our setting, $\mathbf{Y}p$ is always evaluated as true at the end of the trace with weak semantics. On the negative side, doing that we lose the equivalence $\phi \equiv \mathbf{XY}\phi$ that you have in the infinite trace semantics (note that in LTL_f this equivalence does not hold).

Definition 4.3 (Weak semantics of LTL($\mathcal{T}$)) *Given a structure $\mathcal{M}$ and a trace $\sigma \in \mathcal{L}^{\mathcal{M}}(V) \cup \mathcal{L}_f^{\mathcal{M}}(V)$, the weak semantics of a formula φ is defined as follows:*

- $\sigma, \mathcal{M}, i \models_{t^-} \mathit{Pred}(u_1, \ldots, u_n)$ *iff* $|\sigma| \leq i$ *or* $\langle \sigma^+(i), \mathcal{M} \rangle \models \mathit{Pred}(u_1, \ldots, u_n)$
- $\sigma, \mathcal{M}, i \models_{t^+} \mathit{Pred}(u_1, \ldots, u_n)$ *iff* $|\sigma| > i$ *and* $\langle \sigma^+(i), \mathcal{M} \rangle \models \mathit{Pred}(u_1, \ldots, u_n)$
- $\sigma, \mathcal{M}, i \models_{t^{sgn}} \varphi_1 \wedge \varphi_2$ *iff* $\sigma, \mathcal{M}, i \models_{t^{sgn}} \varphi_1$ *and* $\sigma, \mathcal{M}, i \models_{t^{sgn}} \varphi_2$
- $\sigma, \mathcal{M}, i \models_{t^-} \neg\varphi$ *iff* $\sigma, \mathcal{M}, i \not\models_{t^+} \varphi$
- $\sigma, \mathcal{M}, i \models_{t^+} \neg\varphi$ *iff* $\sigma, \mathcal{M}, i \not\models_{t^-} \varphi$
- $\sigma, \mathcal{M}, i \models_{t^{sgn}} \varphi_1 \mathbf{U} \varphi_2$ *iff there exists* $k \geq i, \sigma, \mathcal{M}, k \models_{t^{sgn}} \varphi_2$ *and for all* $l, i \leq l < k, \sigma, \mathcal{M}, l \models_{t^{sgn}} \varphi_1$
- $\sigma, \mathcal{M}, i \models_{t^-} \varphi_1 \mathbf{S} \varphi_2$ *iff* $i \geq |\sigma|$ *or there exists* $k \leq i, \sigma, \mathcal{M}, k \models_{t^-} \varphi_2$ *and for all* $l, k < l \leq i, \sigma, \mathcal{M}, l \models_{t^{sgn}} \varphi_1$
- $\sigma, \mathcal{M}, i \models_{t^+} \varphi_1 \mathbf{S} \varphi_2$ *iff* $i < |\sigma|$ *and there exists* $k \leq i, \sigma, \mathcal{M}, k \models_{t^-} \varphi_2$ *and for all* $l, k < l \leq i, \sigma, \mathcal{M}, l \models_{t^{sgn}} \varphi_1$
- $\sigma, \mathcal{M}, i \models_{t^{sgn}} \boldsymbol{X}\varphi$ *iff* $\sigma, \mathcal{M}, i+1 \models_{t^{sgn}} \varphi$
- $\sigma, \mathcal{M}, i \models_{t^-} \boldsymbol{Y}\varphi$ *iff* $i \geq |\sigma|$ *or* $i > 0$ *and* $\sigma, \mathcal{M}, i-1 \models_{t^-} \varphi$
- $\sigma, \mathcal{M}, i \models_{t^+} \boldsymbol{Y}\varphi$ *iff* $0 < i < |\sigma|$ *and* $\sigma, \mathcal{M}, i-1 \models_{t^+} \varphi$

where $sgn \in \{-, +\}$ and $pred^{\mathcal{M}}$ is the interpretation in $\mathcal{M}$ of the predicate in Σ.
The interpretation of terms $\sigma^{\mathcal{M}}(i)$ is given as for standard LTL($\mathcal{T}$).
Finally, we have that $\sigma, \mathcal{M} \models_t \varphi$ iff $\sigma, \mathcal{M}, 0 \models_{t^-} \varphi$.

4.1.1 Satisfiability and Model checking modulo theories

The problem of LTL [resp., LTL_f] satisfiability modulo theory is the problem of deciding if, given a formula φ, there exists a structure $\mathcal{M}$ and an infinite [resp., finite] trace σ such that $\langle \sigma, \mathcal{M} \rangle \models \varphi$. As usual, the dual validity problem is the problem of deciding if, given a formula φ, for every structure $\mathcal{M}$ and infinite/finite trace σ, $\langle \sigma, \mathcal{M} \rangle \models \varphi$.

In model checking, a temporal property φ of an STS $M = \langle V, I, T \rangle$ is specified over a set $V_\varphi \subseteq V$ of variables. A finite or infinite trace σ of M defines a corresponding trace $\sigma_{|V_\varphi}$ over V_φ given by the projection of the assignments of σ on the subset of variables V_φ.

Given an LTL [resp., LTL_f] formula φ, the LTL model checking problem is the problem to check if $M \models \varphi$ [resp., $M \models_f \varphi$], i.e., if, for every structure $\mathcal{M}$ and every infinite [resp., finite] path π of the STS M, $\mathcal{M}, \pi \models \varphi$.

The validity and model checking problems are equivalent in the sense that one can be reduced to the other and vice versa:

Theorem 4.2 *If φ is an LTL formula over the variables V, φ is valid in LTL($\mathcal{T}$) [resp., in $\text{LTL}_f(\mathcal{T})$] iff $M_U \models \varphi$ [resp., $M_U \models_f \varphi$], where M_U is the universal model defined as $M_U = \langle V, \top, \top \rangle$; vice versa, given an STS $M = \langle V, I, T \rangle$ and an LTL formula φ over V, $M \models \varphi$ [resp., $M \models_f \varphi$] iff $(I \wedge \boldsymbol{G}(\boldsymbol{X}\top \rightarrow T)) \rightarrow \varphi$ is valid in LTL($\mathcal{T}$) [resp., in $\text{LTL}_f(\mathcal{T})$].*

Note that $(I \wedge \mathbf{G}(\mathbf{X}\top \rightarrow T) \rightarrow \varphi$ can be simplified to $(I \wedge \mathbf{G}T) \rightarrow \varphi$ in the case of infinite traces.

Proof: [From validity to model checking:] To prove the equivalence between the model checking of universal model M_U and the validity, we show that the set of infinite (resp. finite) traces of M_U is equal to $\mathcal{L}^{\mathcal{M}}(V)$ (resp. ${\mathcal{L}_{fin}}^{\mathcal{M}}(V)$).

Let $Q = \mathcal{L}(M_U)$ and $Q_f = \mathcal{L}_{fin}(M_U)$. $Q = \mathcal{L}^{\mathcal{M}}(V)$ and $Q_f = {\mathcal{L}_{fin}}^{\mathcal{M}}(V)$

We split the proof in two parts using set inclusion: one direction is trivial $Q \subseteq \mathcal{L}^{\mathcal{M}}(V)$ and $Q_f \subseteq {\mathcal{L}_{fin}}^{\mathcal{M}}(V)$ since the traces of M_U are all traces over V.

We prove the other direction by w.o.c. considering an infinite (resp. finite) trace σ (σ') s.t. $\sigma \in \mathcal{L}^{\mathcal{M}}(V)$ ($\sigma' \in {\mathcal{L}_{fin}}^{\mathcal{M}}(V)$) and $\sigma \notin Q$ ($\sigma' \notin Q_f$). Since σ (σ') is not a trace of M_U then either $\sigma(\sigma'), 0 \not\models I_U$ or for all $0 \leq i < |\sigma|(|\sigma'|) - 1$ $\sigma(\sigma') \not\models T_U$. Since $I_U = T_U = \top$ from the semantics we obtain that no trace violates $\top$ by definition contradicting the claim. Therefore, as expected the universal model contains all the traces over V proving the mapping from validity to model checking. □

Proof: [From model checking to validity:]

($\Rightarrow$)
- We want to prove that: $M \models \varphi \Rightarrow I \wedge \mathbf{G}T \rightarrow \varphi$ is valid over LTL($\mathcal{T}$). By contradiction, we say that $M \models \varphi$ and $I \wedge \mathbf{G}T \rightarrow \varphi$ is not valid over LTL($\mathcal{T}$). Which means that there is an infinite trace σ that violates the formula i.e. $\sigma \models I \wedge \mathbf{G}T \wedge \neg\phi$. However, by satisfaction we derive that $\sigma, 0 \models I$ and for all $i \geq 0 : \sigma, i \models T$ from which we derive that σ is a trace of M; however, by assumption the infinite traces of M satisfy φ (contradiction).
- We want to prove that: $M \models_f \varphi \Rightarrow I \wedge \mathbf{G}(\mathbf{X}\top \rightarrow T) \rightarrow \varphi$ is valid over LTL$_f(\mathcal{T})$.

  By contradiction, we say that $M \models_f \varphi$ and $I \wedge \mathbf{G}(\mathbf{X}\top \rightarrow T) \rightarrow \varphi$ is not valid over LTL$_f(\mathcal{T})$. Which means that there is a finite trace σ that violates the formula i.e. $\sigma \models I \wedge \mathbf{G}(\mathbf{X}\top \rightarrow T) \wedge \neg\varphi$. However, by satisfaction we obtain that $\sigma, 0 \models I$ and for all $i \leq |\sigma| - 1 : \sigma, i \models \mathbf{X}\top \rightarrow T$. We can simplify the latter one as for all $i < |\sigma| - 1$: $\sigma, i \models T$ since $\mathbf{X}\top$ holds when $i < |\sigma| - 1$. As for the case with infinite traces we obtain that σ is a trace of M ending up with a contradiction. It should be noted that without $\mathbf{X}\top$ a finite trace σ disproving the property could have existed; indeed, the $\mathbf{X}\top$ guard is crucial for the correctness of the theorem.

($\Leftarrow$)
- We want to prove that: $I \wedge \mathbf{G}T \rightarrow \varphi$ is valid over LTL($\mathcal{T}$) $\Rightarrow M \models \varphi$. By contradiction we say that $I \wedge \mathbf{G}T \rightarrow \varphi$ is valid over LTL($\mathcal{T}$) and there is an infinite trace of M violating φ. If immediately follows that being a trace of M satisfies I in the initial state and T at each transition; thus contradicting the claim.
- We want to prove that: $I \wedge \mathbf{G}(\mathbf{X}\top \rightarrow T) \rightarrow \varphi$ is valid over LTL$_f(\mathcal{T}) \Rightarrow M \models_f \varphi$.

  By contradiction we say that $I \wedge \mathbf{G}(\mathbf{X}\top \rightarrow T) \rightarrow \varphi$ is valid over LTL($\mathcal{T}$) and there is a finite trace of M violating φ. If immediately follows that being a trace of M satisfies I in the initial state and T at each transition which formally means: $\sigma, 0 \models_f I$ and for all $0 \leq i < |\sigma| - 1$ $\sigma, i \models T$. Since $\mathbf{X}\top$ is valid iff $i < |\sigma| - 1$ then for all $0 \leq i \leq |\sigma| - 1 : \sigma, i \models \mathbf{X}\top \rightarrow T$. From which we show the contradiction.

□

Another important relation exists between the propositional and SMT case. Let P_φ be the set of atomic predicates that occur in the formula $norm(\varphi)$. Let us consider a set $\hat{V}_\varphi$ containing one fresh Boolean variable v_p for each predicate p[1] in P_φ. Let $\hat{\varphi}$ be the propositional formula obtained from φ by substituting every predicate p with v_p.

Given a trace σ over the variables V, we define the trace $\hat{\sigma}$ over the Boolean variables $\hat{V}_\varphi$ as follows: for all i, $0 \leq i < |\sigma|$, $\hat{\sigma}(i)(v_p) = \top$ iff $\sigma, i \models p$. Let $\sigma^e = \sigma \cdot \hat{\sigma}$ be the composition of σ and $\hat{\sigma}$.

Theorem 4.3 (From Propositional to SMT) *φ and $\hat{\varphi} \wedge (\bigwedge_{p \in P_\varphi} v_p \leftrightarrow p)$ are equisatisfiable LTL($\mathcal{T}$). φ and $norm(\varphi) \wedge (\bigwedge_{p \in P_\varphi} v_p \leftrightarrow p)$ are equisatisfiable LTL$_f$($\mathcal{T}$). More specifically, for every infinite trace σ, $\sigma \models \varphi$ iff $\sigma^e \models \hat{\varphi} \wedge (\bigwedge_{p \in P_\varphi} v_p \leftrightarrow p)$; and for every finite trace σ, $\sigma \models \varphi$ iff $\sigma^e \models norm(\varphi) \wedge (\bigwedge_{p \in P_\varphi} v_p \leftrightarrow p)$.*

Proof: [sketch] The proof follows straightforwardly the recursive definition of formulae. The only interesting case is the case in which σ is finite and $\varphi = p$ is atomic. For all i, $0 \leq i < |\sigma|$, $\sigma, i \models p$ iff $\sigma^e \models (\mathbf{X}\top \wedge p) \vee (\neg X \top \wedge p[V'/d])$, which follows directly from the semantics of LTL$_f$($\mathcal{T}$). □

Moreover, if the atomic formulae do not contain the "next" variables, the satisfiability problem can be trivially reduced to the propositional case, by creating an equisatisfiable propositional LTL/LTL$_f$ formula with quantifier elimination applied to $\exists V. \bigwedge_{p \in P_\varphi} (v_p \leftrightarrow p)$. In particular, this can be computed with the ALLSMT procedure [220].

As mention in Chapter 3, in the propositional case, the satisfiability and model checking problems of both LTL and LTL$_f$ are PSPACE-complete [274, 137] while in the general case they are both undecidable.

4.1.2 Relation Between Safety and Finite Semantics

In this section, we show how to reduce relevant fragments and semantics of LTL($\mathcal{T}$) to the LTL$_f$($\mathcal{T}$) semantics via a very simple rewriting.

We enstablish a relation between the notion of *bad prefix* (see Definition 2.1) and the notion of finite trace satisfiability ($\models_f$). In particular, we say that a finite trace is a bad prefix for a SafetyLTL($\mathcal{T}$) formula if the same formula rewritten replacing positive occurrences of $\mathbf{X}$ with $\mathbf{N}$ is satisfied under the LTL$_f$($\mathcal{T}$) semantics. The intuition behind this requirement is that, over finite traces, SafetyLTL formulae are prefix-closed if $\mathbf{X}$

[1] It should be noted that we create distinct variables for the same predicates if it occurs with different terms.

occurs only negatively (by prefix-closed we mean that if a finite trace satisfy ϕ, then all its prefixes must satisfy it as well).

Theorem 4.4 (Bad prefix SafetyLTL($\mathcal{T}$)- LTL$_f$($\mathcal{T}$) relation) *Let σ be a finite trace and φ be a SafetyLTL($\mathcal{T}$) formula.*

$$\sigma \textit{ is a bad prefix of } \varphi \textit{ iff } \sigma \not\models_f \mathcal{S}af2\mathcal{F}(\varphi)$$

where $\mathcal{S}af2\mathcal{F}(\varphi) \doteq \varphi[\boldsymbol{X}/\boldsymbol{N}]$ ensures that every positive occurrence of $\boldsymbol{X}$ in φ get replaced with positive occurrence of $\boldsymbol{N}$.

Proof: We exploit the notion of informative prefix [219] relating the notion of bad prefix with a mapping $L : \mathbb{N} \mapsto 2^{Sub(\varphi)}$ mapping positions in σ to subformulae of φ satisfied at that point of trace (see Definition 2.13). We show that each subformula of the informative prefix is satisfied by any continuation of the bad prefix σ if, and only if σ satisfy with LTL$_f$ semantics the rewriting.

($\Rightarrow$) Since σ is a bad prefix for φ, there is a mapping $L : \mathbb{N} \mapsto Sub(\varphi)$ defined as in Definition 2.13. We can observe that $\neg\mathcal{S}af2\mathcal{F}(\varphi)$ and $\neg\varphi$ are equivalent under the infinite trace semantics. Thus, we will use $\neg\mathcal{S}af2\mathcal{F}(\varphi)$

We prove by induction that for each $0 \leq i < n$, for each $\psi \in Sub(\neg\varphi)$, $\sigma, i \models_f \mathcal{S}af2\mathcal{F}(\psi)$.

Base case:

The base case considers all the propositions (recall that SafetyLTL is presented in NNF, this proof would also work if they were expressed without this restriction). Since all the traces extending σ $\sigma\sigma^\omega, i \models p$ then, by the finite semantics, also $\sigma, i \models_f p$. The same reasoning holds for the negation as well.

Inductive case:

For the inductive case, we recall that, besides the negation occurring initially in $\neg\mathcal{S}af2\mathcal{F}(\varphi)$ the subformula is in NNF. Since L provides a semantic representation, without loss of generality, we consider $\neg\varphi$ in negation normal form. Therefore, we can assume that there are no occurrence of $\mathbf{R}$ and $\mathbf{N}$.

- ($\mathbf{X}$): $\mathbf{X}\psi \in L(i)$ iff $\psi \in L(i+1)$. Therefore, by induction $i+1 < |\sigma|$ and $\sigma, i+1 \models_f \psi$, from which we obtain that $\sigma, i \models_f \mathbf{X}\psi$.

- (**U**): $\psi_1 \mathbf{U} \psi_2 \in L(i)$ iff either $\psi_2 \in L(i)$ or if $\psi_1 \in L(i)$ and $\psi_1 \mathbf{U} \psi_2 \in L(i+1)$. If $\varphi_2 \in L(i)$, then $\sigma, i \models_f \varphi$ by induction thus implying that $\sigma, i \models_f \varphi_1 \mathbf{U} \varphi_2$. Otherwise, $\varphi_1 \in L(i)$ and $\varphi_1 \mathbf{U} \varphi_2 \in L(i+1)$. Since $L(|\sigma|) = \emptyset$, there must be a point $i < k < |\sigma|$ s.t. $\varphi_2 \in (k)$ and for each $i \leq j < k$: $\varphi_1 \in L(j)$. By induction, we obtain that there is $i < k < |\sigma|$ s.t. $\sigma, k \models_f \varphi_2$ and for each $i \leq j < k : \sigma, j \models_f \varphi_1$, which is exactly the semantics of until over finite traces.
- ($\vee/\wedge$): This case follows trivially by induction.
- **S**/**Y**/**Z**/**T**: We omit past proofs for brevity, they can be seen as simpler version of their future counterparts.

($\Leftarrow$) The other direction is specular. One can create an equivalent mapping considering the semantics of LTL_f and from that apply the same reasoning.

□

This relation is very important to keep an equivalence between the model checking problem of finite and the infinite semantics in the safety fragment. From Theorem 4.4, we obtain the following model checking reduction result.

Corollary 4.1 (SafetyLTL to LTL_f reduction) *Let $M = \langle V, I, T \rangle$ be a deadlock-free STS and let φ be a SafetyLTL($\mathcal{T}$) formula.*

$$M \models \varphi \text{ iff } M \models_f \mathcal{S}af2\mathcal{F}(\varphi)$$

Similarly, we can extend the result to the weak and strong semantics $\models_{t^-}, \models_{t^+}$ by also weakening until operator (strengthening release operators):

Theorem 4.5 (Relation between Weak and $\text{LTL}_f(\mathcal{T})$ Semantics) *Let σ be a finite trace violating an LTL($\mathcal{T}$) formula φ with the weak semantics i.e. $\sigma \not\models_{t^-} \varphi$.*

$$\sigma \models_{t^-} \varphi \text{ iff } \sigma \models_f \mathcal{W}eak2\mathcal{F}(\varphi) \text{ and } \sigma \models_{t^+} \varphi \text{ iff } \sigma \models_f \mathcal{S}tr2\mathcal{F}(\varphi)$$

where $\mathcal{W}eak2\mathcal{F}(\varphi) \doteq (\varphi \lceil \mathbf{U}/\mathbf{W} \rfloor \lceil \boldsymbol{X}/\boldsymbol{N} \rfloor$,
$\mathcal{S}tr2\mathcal{F}(\varphi) \doteq \varphi \lceil \mathbf{W}/\mathbf{U} \rfloor \lceil \boldsymbol{N}/\boldsymbol{X} \rfloor$

Proof: We can prove the theorem by induction of the structure of the formula considering both cases $\models_{t^-}, \models_{t^+}$. We observe that $\neg\mathcal{W}eak2\mathcal{F}(\varphi) \doteq \mathcal{S}tr2\mathcal{F}(\neg\varphi)$ and vice

versa; thus, it is sufficient to consider the negation to deal with the derived operators **W** and **N**.

Formally we prove that for each $0 \leq i < |\sigma| : \sigma, i \models_{t^-} \varphi$ iff $\sigma, i \models_f \mathcal{W}eak2\mathcal{F}(\varphi)$ and $\sigma, i \models_{t^+} \varphi$ iff $\sigma, i \models_f \mathcal{S}tr2\mathcal{F}(\varphi)$.

Base case trivially follows from the notion of satisfiability of both semantics in the points before $|\sigma| - 1$.

- ($\models_{t^-}$ **X**) $\sigma, i \models_{t^-} \mathbf{X}\psi$ iff $\sigma, i+1 \models_{t^-} \psi$. If $i < |\sigma| - 1$, by induction $\sigma, i+1 \models_{t^-} \psi$ iff $\sigma, i+1 \models_{t^-} \psi$. If $i = |\sigma| - 1$ then we know that, by weak semantics $\sigma, i \models \mathbf{X}\psi$. From which we derive that $\sigma, i \models_{t^-} \mathbf{X}\psi$ iff $i = |\sigma| - 1$ or $\sigma, i+1 \models_f \psi$ iff $\sigma, i \models_f \mathbf{N}\psi$.

- ($\models_{t^+}$ **X**) This case is specular, any formula is false if evaluated after $|\sigma| - 1$, and **X** is false at the end of the trace with LTL$_f$ semantics.

- ($\models_{t^-}$ **U**) $\sigma, i \models_{t^-} \varphi_1 \mathbf{U} \varphi_2$ iff there is a $k \geq i$ s.t. $\sigma, k \models_{t^-} \varphi_2$ and for all $i \leq j < k$: $\sigma, j \models_{t^-} \varphi_1$. Since each formula is satisfied at the end of the trace, we obtain that it holds iff [there is an $i \leq k < |\sigma|$ such that $\sigma, k \models_{t^-} \varphi_2$ and for all $i \leq j < k$ $\sigma, j \models_{t^-} \varphi$] or for all $i \leq j < |\sigma| : \sigma, j \models_{t^-} \varphi_1$. By induction, we obtain the exact semantics of weak until.

- ($\models_{t^+}$ **U**) As the previous case, with the difference that here $\sigma, |\sigma| \not\models_{t^+} \varphi_2$, and thus, you can reduce to standard until with LTL$_f$.

- ($\vee/\neg$) Follows from induction and satifiability semantics.

- (**Y**/**S**) The semantics of the past is identical when the formulae are interpreted before the end of the trace.

□

Corollary 4.2 (Weak Semantics of LTL($\mathcal{T}$) to LTL$_f$ and LTL$_f$($\mathcal{T}$) reduction) *Let M be an STS and let φ be an LTL($\mathcal{T}$) formula.*

$$M \models_{t^-} \varphi \text{ iff } M \models \varphi \text{ and } M \models_f \mathcal{W}eak2\mathcal{F}(\varphi)$$
$$M \models_{t^+} \varphi \text{ iff } M \models \varphi \text{ and } M \models_f \mathcal{S}tr2\mathcal{F}(\varphi)$$

	Infinite Traces $M \models \varphi$		Finite Traces $M \models_f \varphi$
	LTL	Safety Fragments	LTL_f
Propositional	$M \times M_{\neg\varphi} \models \mathbf{FG}q$	$M \times M_{\neg\varphi} \models_f \mathbf{G}q$	
Modulo Theories	$M \times M_{\neg\hat{\varphi}} \times S_{p_i \leftrightarrow \gamma_i(V)} \models \mathbf{FG}q$	$M \times M_{\neg\hat{\varphi}} \times M_{p_i \leftrightarrow \gamma_i(V)} \models_f \mathbf{G}q$	
Engine	Liveness Checker	Invariant Checker	

Table 4.1: Symbolic automata-theoretic approaches.

4.2 Automata-Theoretic Verification of LTL and LTL_f Modulo-Theory

In this section, we provide a general and comprehensive view of the applicability of the symbolic automata-theoretic approach to LTL and its extensions. See Table 4.1. We start from the classical symbolic automaton construction [117], used to reduce LTL model checking [2] to fair reachability. In case of safety LTL and LTL_f, this can be reduced to checking if q holds invariantly, i.e. the fairness condition $\neg q$ simplifies to a reachability condition. In the infinite-state case of Verification Modulo Theories, the construction is applied to the Boolean abstraction of φ, denoted $\hat{\varphi}$, which is φ where each theory atom p is replaced by a fresh Boolean variable v_p. The definition constraints M_φ^{lift} are then added to the description of the transition system. As detailed in the chapter, additional rewriting is necessary in case of finite words to handle the presence of next variables in the theory atoms.

To illustrate the generality of the approach, we contribute with a library of benchmarks, expressed in the VMT-LIB open format [106], collected from multiple sources from the literature. We also provide a set of tools, integrated in the open source PyVMT framework [258], to reduce the different variants of LTL to the appropriate liveness or invariant checking problems for symbolic transition systems, using a general symbolic automaton construction that supports uniformly LTL, LTL_f and SafetyLTL.

In the remainder of the section, we use the symbolic compilation of LTL and LTL_f introduced in Section 2.5.4, adapting it to support reasoning modulo background theories.

[2] The approach also supports LTL validity and satisfiability checking. $\models \varphi$ is reduced to a model checking problem $M_U \models \varphi$, where M_U is the universal transition system. Satisfiability is obtained by dualization, i.e. φ is satisfiable iff $M_U \not\models \neg\varphi$.

4.2.1 From Propositional to Modulo Theory

In Section 2.5.3, we presented the automata-theoretic approach for the verification of LTL and its safety fragment. The automata-theoretic approaches can be lifted naturally to the "modulo theory" case, by considering the Boolean abstraction of the property and adding constraints to the system. In this Section, we show how to reduce the verification of LTL modulo theory with Boolean abstraction. More specifically, let P_φ, $\hat{V}_\varphi$, and $\hat{\varphi}$ be defined as above. Let $M_\varphi^{\text{lift}} \doteq \langle V \cup \hat{V}_\varphi, \top, \bigwedge_{Pred \in P_\varphi} v_{Pred} \leftrightarrow Pred \rangle$. Then, we can prove the following theorems that lift the previous one to the SMT case.

Theorem 4.6 *$M \models \varphi$ iff $M \times M^l_{\neg\hat{\varphi}} \times M^{lift}_\varphi \models \boldsymbol{FG}\neg f_{\neg\hat{\varphi}}$.*

Theorem 4.7 *$M \models_f \varphi$ iff $M \times M^s_{\neg\hat{\varphi}_n} \times M^{lift}_{\varphi_n} \models_f \boldsymbol{G}\neg f_{\neg\hat{\varphi}_n}$, where $\varphi_n = norm(\varphi)$.*

Theorem 4.8 *If φ is in SafetyLTL and M is deadlock free, $M \models \varphi$ iff $M \times M^b_{\neg\mathcal{S}af2\mathcal{F}(\hat{\varphi})} \times M^{lift}_\varphi \models_f \boldsymbol{G}\neg f_{\neg\mathcal{S}af2\mathcal{F}(\hat{\varphi})}$.*

Proof: [Theorem 4.6] Let σ be a trace of M violating φ. By Theorem 4.3, $\hat{\sigma} \models \neg\hat{\varphi}$. Thus, there exists a trace σ^l of $M^l_{\neg\hat{\varphi}}$ with $\sigma^l_{|\hat{V}_\varphi} = \hat{\sigma}$ satisfying $\neg f_{\neg\hat{\varphi}}$ infinitely many times. Let $\sigma^\times$ be the trace of $M \times M^l_{\neg\hat{\varphi}} \times M^{lift}_\varphi$ obtained by composing σ, $\hat{\sigma}$, and σ^l. Then $\sigma^\times \models \neg\mathbf{FG}\neg f_{\neg\hat{\varphi}}$.

Let $\sigma^\times$ be a trace of $M \times M^l_{\neg\hat{\varphi}} \times M^{lift}_\varphi$ satisfying $\neg\mathbf{FG}\neg f_{\neg\hat{\varphi}}$. Let $\hat{\sigma}$ be $\sigma^\times_{|\hat{V}_\varphi}$ and σ be $\sigma^\times_{|V}$. Then $\sigma^\times_{|V_\varphi} \models \neg\hat{\varphi}$. For all $i \geq 0$ $\hat{\sigma}, i \models v_p$ iff $\sigma, i \models p$. By Theorem 4.3, $\sigma \models \neg\varphi$. □

The proof of the other two theorems is similar.

In the remainder of this dissertation, we will refer to the "lifted" constructions when referring to the automata construction for LTL modulo theory.

4.2.2 Running Example

We showcase the various parts of our construction with a couple of small examples using $\mathcal{LIA}$ (already introduced in Section 2.1) as underlying theory. Let an STS $M = \langle V, I, T \rangle$ with $V := \{i, o, n\}$, $I := \top$ and $T := n' = n \wedge o' = i + 1$. The STS M has an "input" variable i (i.e. a variable in which its primed counterpart is not constrained), an output variable o and a frozen variable n; each transition replicates the value of i incremented by one in the next state.

A liveness Example: We now consider the response property $\varphi_f := \mathbf{G}(i = n \rightarrow \mathbf{F}(o = n+1))$. It is easy to see that this liveness property holds; the transition relation satisfies the sub-formula $\mathbf{F}(o = n+1)$ in one step every time that i is equal to n. We now apply the construction defined in the previous subsections. First, we define a propositional formula and a "lift" transition system mapping the original property to the propositional one. Subsequently, we construct the symbolic compilation for the negation of the property.

We can easily construct $\hat{\varphi_f}$ by replacing each predicates with fresh Boolean variables as $\hat{\varphi_f} := \mathbf{G}(v_{p_0} \rightarrow \mathbf{F}v_{p_1})$. Moreover, we define the "lift" transition system as $M_{\varphi_f}^{lift} := \langle V \cup \hat{V}_{\varphi_f}, \top, \hat{T}_{\varphi_f} \rangle$ where $\hat{V}_{\varphi_f} := \{v_{p_0}, v_{p_1}\}$ and $\hat{T}_{\varphi_f} := (v_{p_0} \leftrightarrow i = n) \wedge (v_{p_1} \leftrightarrow o = n+1)$. With this mapping between the propositional LTL formula and the LTL($\mathcal{T}$) formula we just need to construct the STS corresponding to the propositional formula to prove the property.

We now take the negation of $\hat{\varphi_f}$ without abbreviations. The resulting $\neg\hat{\varphi_f}$ formula is as follows. $\neg\hat{\varphi_f} := \top \mathbf{U} (v_{p_0} \wedge \neg(\top \mathbf{U} v_{p_1}))$. Since our formula contains 2 until temporal operator, $V_{\neg\hat{\varphi_f}} := \{v_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \neg(\top \mathbf{U} v_{p_1})))}, v_{\mathbf{X}(\top \mathbf{U} v_{p_1})}\}$. The initial condition of $STS_{\neg\hat{\varphi_f}}$ is then encoded as $I_{\neg\hat{\varphi_f}} := v_{p_0} \wedge \neg v_{\mathbf{X}(\top \mathbf{U} v_{p_1})} \vee v_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \neg(\top \mathbf{U} v_{p_1}))}$. Either the right side of the outer until holds in the initial state, or it will hold eventually in the future thanks to the prophecy variable. The transition relates the next value of the subformulae with their corresponding prophecy variables. $T_{\neg\hat{\varphi_f}} := (v_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \neg(\top \mathbf{U} v_{p_1}))} \leftrightarrow v'_{p_0} \wedge \neg v'_{\mathbf{X}(\top \mathbf{U} v_{p_1})} \vee v'_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \neg(\top \mathbf{U} v_{p_1}))}) \wedge (v_{\mathbf{X}(\top \mathbf{U} v_{p_1})} \leftrightarrow v'_{p_1} \vee v_{\mathbf{X}(\top \mathbf{U} v_{p_1})})$. Finally, we add the fairness condition forcing either the until prophecy to be false or to witness its right side. $F_{\neg\hat{\varphi_f}} := \{v_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \neg(\top \mathbf{U} v_{p_1}))} \vee (v_{p_0} \wedge \neg(v_{p_1} \vee v_{\mathbf{X}(\top \mathbf{U} v_{p_1})}) \rightarrow (\neg(v_{p_1} \vee v_{\mathbf{X}(\top \mathbf{U} v_{p_1})}), v_{p_1} \vee v_{\mathbf{X}(\top \mathbf{U} v_{p_1})} \rightarrow v_{p_1}\}$.

An LTL$_f$($\mathcal{T}$) Example: We now consider a bounded response property (with bound 1) $\varphi_b := \mathbf{G}(i = n \rightarrow \mathbf{X}(o = n+1))$. Although with infinite semantics we would expect this property to hold, in finite semantics any finite trace terminating with $i = n$ is a valid counterexample of the property.

As before, we apply the construction defined in the previous subsections tailored for LTL$_f$. Since no primed variable occurs in φ_b, $norm(\varphi_b)$ is equivalent to φ_b. Thus, we simply use directly φ_b as our target formula. For brevity, we skip the generation of the lift STS since it is identical to the one of φ_f.

We now take the negation of $\hat{\varphi_b}$ in negative normal form. The resulting $\hat{\varphi_b}' := \text{NNF}_f(\neg\hat{\varphi_b})$ formula is as follows. $\hat{\varphi_b}' := \top \mathbf{U} (v_{p_0} \wedge \mathbf{N}(\neg v_{p_1}))$. Since our formula contains an until and a weak next, $V_{\hat{\varphi_b}'} := \{v_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \mathbf{N}(\neg v_{p_1})))}, v_{\mathbf{N}(\neg v_{p_1})}\}$. The initial condition

of $STS_{\hat{\varphi}_b'}$ is then encoded as $I^s_{\hat{\varphi}_b'} := v_{p_0} \wedge v_{\mathbf{N}(\neg v_{p_1})} \vee v_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \mathbf{N}(\neg v_{p_1})))}$. The transition relates the next value of the subformulae with their corresponding prophecy variables. Since our formula is in negation normal form, we do not need double implication. The resulting transition is encoded as $T^s_{\hat{\varphi}_b} := (v_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \mathbf{N}(\neg v_{p_1})))} \rightarrow v'_{p_0} \wedge v'_{\mathbf{N}\neg v_{p_1}} \vee v'_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \neg(\top \mathbf{U} v_{p_1})))}) \wedge (v_{\mathbf{N}\neg v_{p_1}} \rightarrow \neg v'_{p_1})$. Finally, we construct the reachability condition $f^s_{\hat{\varphi}_b'}$ with the prophecy as $f^s_{\hat{\varphi}_b'} := \neg v_{\mathbf{X}(\top \mathbf{U}(v_{p_0} \wedge \neg(\top \mathbf{U} v_{p_1})))}$. It should be noted that, with such reachability condition, any finite trace terminating with v_{p_0} has a corresponding path with the prophecy variable assigned to false.

4.2.3 LTL($\mathcal{T}$) and LTL$_f$($\mathcal{T}$) Benchmarks

The need for establishing a collection of benchmark problems, ideally written in a simple yet expressive language widely supported by different tools, has long been recognized in several communities in formal verification and automated reasoning. The success of initiatives such as TPTP [278] and SMT-LIB [16] in automated reasoning, or Aiger [48], Btor [251] and SV-COMP [41] in formal verification has been a key factor for the significant practical advancements of the state of the art in their respective fields over the last decades.

Recently, VMT-LIB [106] have been proposed, an extension of the standard SMT-LIB language for SMT solvers, as a language for the representation of invariant checking and liveness checking problems on infinite-state symbolic transition systems. VMT-LIB has a flexible format and clear semantics based on SMT-LIB, and supports the specification of LTL and LTL$_f$ properties with the (extended) syntax introduced in Section 4.1. More specifically, VMT-LIB has metadata tags `:ltl-property` and `:ltlf-property`, with which LTL and LTL$_f$ specifications can be annotated.

A benchmark set for LTL satisfiability and verification. As a first contribution to the creation of a standard benchmark library for LTL model checking and satisfiability, we have collected various benchmark sets from different sources from the literature [225, 165, 271, 69, 51], and converted them to VMT-LIB. The resulting benchmark library consists of about 3000 instances. We make the benchmarks available at `http://www.vmt-lib.org`.

Tool support. As mentioned above, an important feature of the VMT-LIB format is that it is possible to precisely represent LTL and LTL$_f$ properties using the full syntax defined in Section 4.1. This ensures that the LTL benchmarks we collected can be kept

at their original/intended level of abstraction, rather than being reduced to simpler invariant or liveness properties via reductions similar to those described in Section 2.5.3. This is crucial in order to make the benchmark set useful for evaluating techniques that might exploit alternative reductions and/or specialised encodings for particular patterns or kinds of properties. At the same time, however, we have developed a set of algorithms implementing the reductions of Section 2.5.3 in order to convert arbitrary LTL or LTL_f properties into simpler liveness or invariant ones. Therefore, the benchmarks can be used also in the evaluation of core verification algorithms/tools that only target such special kinds of properties. In particular, we have integrated our implementations in PyVMT [258], an open-source Python-based programmatic framework for interacting with model checkers developed as an extension of the PySMT [175] library for SMT solvers. PyVMT provides a lightweight interface for constructing and manipulating transition systems in a symbolic form, with native support for representing and processing $\text{LTL}(\mathcal{T})$ and $\text{LTL}_f(\mathcal{T})$ formulas. It supports the VMT format natively and enables direct model checking via backends such as nuXmv[87], IC3IA[102], and EUForia [82], and also includes support for trace manipulation. The framework is designed to be highly modular, making it straightforward to extend with additional tools or encoding strategies. Further details about the tool can be found in [258].

4.3 Extending Safety Fragment Verification with Relative Safety

In previous Sections, we have shown how the safety fragments of LTL can be checked with a direct reduction to invariant checking (assuming that the transition system is deadlock-free) while the full fragment needs more sophisticated construction to be handled (e.g. liveness to safety, k-liveness). This intuitively occurs because liveness requires reasoning over infinitely many occurrences of some events while for safety properties it is sufficient to prove an invariant over some states. However, many properties that are not strictly safety can be considered as "safety" assuming some known behaviours. One notable example is *bounded response* of real-time systems. Bounded response states that given an input an output will be provided in a bounded amount of time. When interpreting this specification over real-time, we might incur into *Zeno* executions i.e. infinite executions in which time converges to a value. Therefore, in this setting, it is possible to exploit the domain knowledge to reduce a – potentially very complex – liveness checking query into a more tractable *invariant checking* problem. In this

Section, we leverage the notion of relative safety introduced by Henzinger in [194] to provide a more convenient encoding of a significant fragment of LTL($\mathcal{T}$) strictly larger than SafetyLTL($\mathcal{T}$). Furthermore, we show a refinement loop approach to deal with systems that might contain deadlocks and/or livelocks. Finally, we show the benefits of this approach performing an experimental evaluation comparing the loop-approach to the automata-theoretic technique presented in Section 4.2.

Outline: In Section 4.3.1, we introduce the concepts of relative safety and relative liveness, illustrating their significance through concrete examples, including real-time systems and compositional verification; in Section 4.3.2, we present a novel algorithm to verify non-safety fragments of LTL($\mathcal{T}$) exploiting relative safety; in Section 4.3.3, we present a loop approach that blocks non-fair counterexamples until the property is either proved or disproved; finally, in Section 4.3.4, we provide an experimental evaluation that compares our approach with the standard one.

4.3.1 Relative Safety and Relative Liveness

Recall the notion of safety and liveness properties introduced in Section 2.3. The notion of relative safety generalize unconditional safety (or safety for short) relativizing it to another property. Intuitively, if we assume that a property A is satisfied, then every violation of P in this context has a bad prefix. Therefore, if we consider an infinite trace σ violating P i.e. $\sigma \notin P$ satisfying A i.e. $\sigma \in A$; there must be a prefix such that *any* possible extension of the prefix that *satisfies* A also violates P. Figure 4.1 depicts a graphical view of relative safety over LTL properties over a simple example. Note that although Figure 4.1 shows an example in which, if A is false, then P is true; this is not in general the case. Formally, this concept is described by the following definition.

Definition 4.4 *(Relative Safety [194]) Let P and A be two properties over a set of variables V. P is safety relative to A iff*

$$\begin{aligned} &\textit{for all } \sigma \in A \textit{ s.t. } \sigma \notin P, \textit{ there exists } \sigma_f \in \mathit{Pref}(\sigma) \textit{ s.t.} \\ &\textit{for all } \sigma^\omega \in \mathcal{L}(V) \textit{ if } \sigma_f\sigma^\omega \in A \textit{ then } \sigma_f\sigma^\omega \notin P \end{aligned}$$

Another relevant property presented in [194] is relative liveness, which relativize liveness stating that prefixes of traces satisfying another property A are also prefixes of traces of our property P. The intuition is that, if it is assumed that a finite/bounded behaviour occurs (represented by the prefixes of A), then our property P becomes

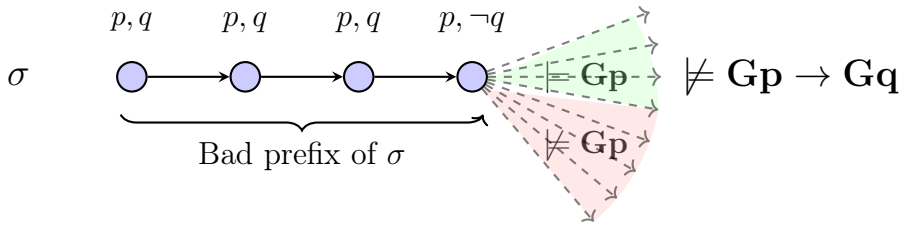


Figure 4.1: Graphical representation of the relative safety concept. Dashed rays represent possible suffixes of σ_f (including σ). Green ones satisfy $\mathbf{G}p$, red ones violate it. This illustrates that $\sigma \not\models \mathbf{G}p \rightarrow \mathbf{G}q$ and there is a prefix σ_f such that any extension still satisfying $\mathbf{G}p$ violates $\mathbf{G}p \rightarrow \mathbf{G}q$.

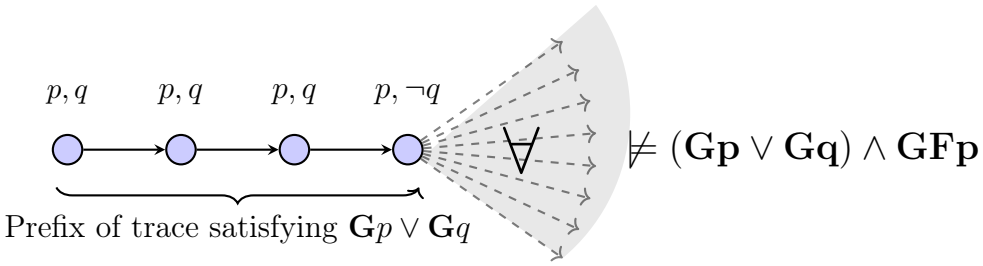


Figure 4.2: Graphical representation of the violation of relative liveness. $(\mathbf{G}p \vee \mathbf{G}q) \wedge \mathbf{GF}q$ is not liveness relative to $\mathbf{G}p \vee \mathbf{G}q$. Dashed rays represent possible suffixes of σ_f. Each extension of σ_f (which is a prefix of $\mathbf{G}p \vee \mathbf{G}q$) violates $(\mathbf{G}p \vee \mathbf{G}q) \wedge \mathbf{GF}q$.

"liveness". It should be noted that, even if we consider traces whose prefixes are prefix of A, we may encounter consider infinite traces that do not satisfy A. As an example consider a STS $M = \langle V, I, T, F\rangle$, let $\phi_M \doteq I \wedge \mathbf{G}T \wedge \bigwedge_{f\in F} \mathbf{GF}f$ be the LTL($\mathcal{T}$) formula representing the STS (see Theorem 4.2 for the equivalence). ϕ_M is liveness relative to $I \wedge \mathbf{G}T$ if and only if STS has no livelock i.e. all reachable states are fair. Figure 4.2 depicts an example of a violation relative liveness condition. Finally, the formal definition of relative liveness is as follows.

Definition 4.5 *(Relative Liveness [194]) Let P and A be two properties over a set of variables V. P is liveness relative to A iff*

$$\textit{for all } \sigma \in A,\ \textit{for all } \sigma_f \in \textit{Pref}(\sigma),\ \textit{there exists } \sigma^\omega \in \mathcal{L}(V) :\ \textit{s.t. } \sigma_f\sigma^\omega \in P$$

To better understand the notion of relative safety and its applicability, in the following we provide meaningful examples in LTL($\mathcal{T}$) properties with this property.

Entailment of Safety Properties An interesting example of a relative safety property is in the classic entailment of two safety properties (also depicted in Figure 4.1).

This is a very natural property that can be used to express that assuming that the environment satisfy a safety property, the system must fulfil the property again. We can express that with the formula $\phi_{AG} \doteq \mathbf{G}\phi_A \rightarrow \mathbf{G}\phi_G$, and we can easily observe that ϕ_{AG} is safety relative to $\mathbf{G}\phi_A$.

Asynchronous Composition of Software Systems In the context of asynchronous software systems, if we want to check a property concerning a set of processes that are executed in interleaving, we might reasonably assume that all the processes are scheduled infinitely often. Therefore, if we pick the property $\phi_{sched} := \bigwedge_{p_i \in PROCS} \mathbf{GF} run_{p_i} \rightarrow \phi_{COMP}$ where ϕ_{COMP} is a safety property over the interleaved system; ϕ_{sched} is safety relative to the fairness constraint implying the safety property.

Timed Bounded Response Under Non-Zenoness Assumption We now port the example provided by Henzinger to our discrete-time setting using an explicit real *time* variable and expressing the properties with LTL($\mathcal{T}$). Therefore, let us consider the property:

$$\psi := \mathbf{G}((p \wedge time = t) \rightarrow \mathbf{F}(q \wedge time \leq t + \delta))$$

for some constant δ.

Let us assume that the values of *time* are diverging over infinite execution traces, for example by posing the following constraints:

$$\alpha := \mathbf{G}(time' \geq time) \wedge \mathbf{GF}(time' - time > \zeta)$$

for some constant ζ.

Every trace σ satisfying α and violating ψ will have a state $\sigma(i)$ in which p holds and *time* has some value τ and a following state $\sigma(j)$ with $j \geq i$ in which *time* has a value greater than or equal to $\tau + \delta$ and for states between $\sigma(i)$ and $\sigma(j)$ q is false. Note that the assumption α is necessary to rule out counterexamples to ψ where *time* is always less than $\tau + \delta$. The prefix of σ up to $\sigma(j)$ violates ψ for any possible suffix. Thus, it is a finite trace witnessing the violation of ψ.

In fact, it can be proved that:

$$\alpha \rightarrow (\psi \leftrightarrow \phi)$$

where

$$\phi := \mathbf{G}((p \wedge time = t) \rightarrow (time \leq t + \delta)\ \mathbf{W}\ (q \wedge time \leq t + \delta))$$

which is safety. Thus, instead of $\alpha \rightarrow \psi$, we can prove $\alpha \rightarrow \phi$.

From these examples it is easy to see that any implication between a generic LTL($\mathcal{T}$) formula α and a SafetyLTL($\mathcal{T}$) formula φ_S is safety relative to α. In the remainder of the section, we will use this observation to define an algorithmic approach to solve these kinds of properties.

4.3.2 Invariant Reduction of Relative Safety Properties

Consider the formula $\alpha := \alpha_S \wedge \alpha_L$ defined as the conjunction of a SafetyLTL($\mathcal{T}$) formula α_S and a liveness formula i.e. a Boolean combination of occurrence of **GF** of predicates over V α_L, and a SafetyLTL($\mathcal{T}$) formula φ. We note that the syntactic restriction for α does not reduce expressiveness because every property can be specified by the conjunction of a safety property and a liveness property. We define a set of conditions such that we have a direct construction from $\alpha \rightarrow \varphi$ to invariant checking.

The first condition we need is related to the extendibility of finite paths of an STS to infinite traces of a formula α. Ideally, we would like that any path of M that is a *prefix* of an infinite trace of an LTL formula α has also an infinite continuation sastisfying α that is an infinite path of M. The intuition behind this property is that –when we find a finite witness that could satisfy α– we are sure that one extension from the STS exists.

Definition 4.6 (Live with Respect to a Property) *A STS $M = \langle V, I, T \rangle$ is live with respect to an LTL property α iff for every finite trace σ and finite path π of M over σ, if there exists a trace σ' such that $\sigma\sigma' \models \alpha$, then there exists an infinite path π^ω of M such that $\pi \in Pref(\pi^\omega)$ and $\pi^\omega \models \alpha$.*

It is easy to see that M is live w.r.t. $\top$ if it does not have deadlocks. Moreover, if we pick a Fair STS that extends M with the fair condition set F, M is live w.r.t. $\bigwedge_{f \in F} \mathbf{GF} f$ iff all the states of the Fair STS are fair. Interestingly, if we look it from the point of view of a $\mathbf{G}\phi$ property, we are requiring that every finite path satisfying ϕ as an invariant has an infinite continuation satisfying ϕ at each point in time. Although similar to the notion of relative liveness, the notion of live w.r.t. is tailored to reason over traces and paths that are defined over a different set of variables. This is needed because the property traces refer to a subset of symbols of the STS finite path.

The second condition we require is that α satisfies liveness relative to α_S. Ideally, we seek a condition where any finite prefix satisfying the safety property assumption can be extended to a trace that continues to satisfy the safety assumption while also fulfilling the liveness guarantees of α_L.

Combining the previous conditions and using Theorem 4.4, we obtain the following reduction to LTL$_f(\mathcal{T})$ model checking.

Theorem 4.9 *Let $M = \langle V, I, T\rangle$ be an STS, let $\alpha \doteq \alpha_S \wedge \alpha_L$ with α_S SafetyLTL($\mathcal{T}$) and α_L liveness and let φ be a SafetyLTL($\mathcal{T}$) formula expressed in negation-normal form. If M is live w.r.t. α and α is liveness relative to α_S then*

$$M \models \alpha \rightarrow \varphi \text{ iff } M \models_f \mathcal{S}af2\mathcal{F}(\alpha_S) \rightarrow \mathcal{S}af2\mathcal{F}(\varphi)$$

Proof: We prove the two cases by way of contradiction.

($\Rightarrow$) Let σ_f be the counterexample trace. From Theorem 4.4, since $\sigma_f \models_f \mathcal{S}af2\mathcal{F}(\alpha_S)$ is a safetyLTL formula then σ_f is not a "bad prefix" of α_S. This means that there is an infinite trace $\sigma \doteq \sigma_f\sigma^\omega$ such that $\sigma \models \alpha_S$ i.e. σ_f is a prefix of an infinite trace satisfying α_S.

By assumption, we know that α is liveness relative to α_S, and consequently, σ_f is a prefix of an infinite trace σ satisfying α.

We now need to prove that σ is also a trace induced by some path in M. This condition follows from the fact that M is live w.r.t. α. Therefore, we obtain that σ is an infinite trace induced by some path in M satisfying α.

Since σ_f violates $\mathcal{S}af2\mathcal{F}(\varphi)$, then σ_f is a bad prefix for φ, meaning that all traces extending σ_f violate φ, which means that also σ violates φ. Finally, proving that $\sigma \models \alpha$ and $\sigma \models \neg\varphi$ proving by w.o.c. the ($\Rightarrow$) case.

($\Leftarrow$) Suppose that $M \models_f \mathcal{S}af2\mathcal{F}(\alpha_S) \rightarrow \mathcal{S}af2\mathcal{F}(\varphi)$, but there is an infinite trace σ of M violating $\alpha \rightarrow \varphi$.

Since φ is a safety property, σ must have a bad prefix σ_f such that any continuation violates φ. From Theorem 4.4, we know derive that $\sigma_f \models_f \neg\mathcal{S}af2\mathcal{F}(\varphi)$. Since by assumption $\sigma_f \models_f \mathcal{S}af2\mathcal{F}(\alpha_S) \rightarrow \mathcal{S}af2\mathcal{F}(\varphi)$, σ_f must violate $\mathcal{S}af2\mathcal{F}(\alpha_S)$, which by Theorem 4.4 entails that σ_f is a bad prefix for α_S. This contradicts the fact that σ_f is a prefix of a trace satisfying $\alpha_S \wedge \neg\varphi$, proving the $\Leftarrow$ case by contradiction. □

From Theorem 4.9, we obtain a direct reduction from a fragment of LTL($\mathcal{T}$) to invariant model checking.

We can now show the potential use of this reduction in use cases.

Consider the example of *Asynchronous Composition of Software Systems* introduced in Sec. 4.3.1. The formula ϕ_{sched} is a liveness assumption and since there is no safety specification assumption we trivially satisfy the liveness relative condition. The formula ϕ_{COMP} is in the SafetyLTL($\mathcal{T}$) fragment for which we can use our algorithm. Finally, we

need to ensure that the composition of the components is live w.r.t. ϕ_{sched}. This means that all reachable states of the automaton allow a possible fair scheduling. Although this condition might not always be ensured, it is possible to compute such states using BDD based approaches.

Consider now the *Timed Bounded Response Under Non-Zenoness Assumption* introduced in Section. 4.3.1. It is easy to see that α is liveness relative to its safety part $\mathbf{G}(time' \geq time)$. Then we can apply our approach assuming Timed Transition Systems without timelocks i.e. timed paths that cannot make time diverge. Indeed, using this approach it is possible to construct an efficient algorithm to check safetyMTL properties assuming the absense of timelocks in Timed Automata.

4.3.3 Counterexample Guided Refinement Loop Algorithm

The reduction to invariant model checking presented in the previous section is limited to specific settings. In the following, we generalize the approach proposed before using a refinement loop approach. Basically, if we get a counterexample to the invariant model checking query, we can test whether it is a real counterexample by looking if the property is extendable to an infinite trace. If so, we know that the premise is indeed a real counterexample; otherwise, we can try to block bad states that are not live w.r.t. the original property and apply the invariant check again. Eventually, either we will find a real counterexample or we end up block all the states that are not live w.r.t. the assumption. The approach is presented in Algorithm 1.

```
1  Function verifyRelativeSafety(M, α, φ):
2      M' ← M × M_α^l;
3      ⟨M_¬φ^b, f_¬φ⟩ ← SafetyLTL2STS(φ);
4      if M' × M_¬φ^b ⊨_f G f_φ then
5          return VALID
6      end
       /* σ_f is a finite cex, try to extend it to α                    */
7      if M' ⊨ GF f_α → G(¬l(σ_f)) then
8          M' ← ⟨V' ∪ V_gen, I', T' ∧ ¬l(σ_f)⟩;
9          goto 4;
10     end
11     return INVALID
```

Algorithm 1: Algorithm to verify $\alpha \rightarrow \varphi$. SafetyLTL2STS refers to the automata construction for SafetyLTL($\mathcal{T}$) while $l(\sigma_f) = \sigma_f(|\sigma_f| - 1)$

Theorem 4.10 *Algorithm 1 is sound.*

Proof: If $M' \times M^b_{\neg\varphi} \models_f \mathbf{G} f_\varphi$, then there is no finite trace of M' violating φ; therefore, since φ is a safety property there is no infinite trace violating φ that satisfies α without fairness and consequently α with fairness. If $M' \models \bigwedge_{f \in F_\alpha}(\mathbf{GF} f) \rightarrow \mathbf{G}\neg l(\sigma_f)$, then $l(\sigma_f)$ is a state in M' that cannot be extended visiting infinitely often F_α; thus, blocking it preserves satisfiability

Finally, if $M' \not\models \bigwedge_{f \in F_\alpha}(\mathbf{GF} f) \rightarrow \mathbf{G}\neg l(\sigma_f)$, then the finite counterexample σ_f is a prefix of M' and can be extended by a suffix σ^ω, which is part of the violation of $\mathbf{GF} f_\alpha \rightarrow \mathbf{G}\neg l(\sigma_f)$; thus, making $\sigma = \sigma_f \sigma^\omega$ a trace of M' and, consequently, a trace of M and a trace of α. Therefore, since σ_f is a bad prefix for φ, σ is a trace of $\alpha \wedge \neg\varphi$ which is a legitimate counterexample of $\alpha \rightarrow \varphi$. □

Theorem 4.11 *If M is a finite state STS, Algorithm 1 is complete.*

Proof: We only need to prove that at some point we stop blocking bad states. Since M is a finite state STS and M_α does not introduce infinite states, the states that cannot be extended to infinite are finite and will eventually be all blocked. Therefore, the algorithm will be forced to exit either with VALID or INVALID result. □

Extending algorithm with look-ahead

The main weakness of Algorithm 1 is the overhead given by the loop iterations for any non-extendable trace. To face this issue, we introduce an optimization that computes a look-ahead of length n on each invariant counterexample. The idea is that the invariant check must compute n successor states after the bad prefix of φ pruning all counterexamples with deadlocks within n steps. The new construction is built on top of $SafetyLTL2STS$ as follows:

Definition 4.7 *We define $\boldsymbol{X}depth(\phi)$ recursively as follows:*
(i) $\boldsymbol{X}depth(p) = 0$ *(ii)* $\boldsymbol{X}depth(\neg\phi) = \boldsymbol{X}depth(\phi)$
(iii) $\boldsymbol{X}depth(\boldsymbol{X}\phi) = \boldsymbol{X}depth(\phi)+1$ *(iv)* $\boldsymbol{X}depth((\phi')) = \boldsymbol{X}depth(\phi)+1$ *(v)* $\boldsymbol{X}depth(\phi_1 \vee \phi_2) = max(\boldsymbol{X}depth(\phi_1), \boldsymbol{X}depth(\phi_2))$
(vi) $\boldsymbol{X}depth(\phi_1 \,\mathbf{U}\, \phi_2) = max(\boldsymbol{X}depth(\phi_1), \boldsymbol{X}depth(\phi_2))$.

We integrate this construction into Algorithm 1. At the first iteration parameter n is initialised heuristically to $\mathbf{X}depth(\alpha) + 1$, then it is incremented by one at each step. Moreover, when we check that a finite counterexample is extensible, instead of

considering the last state of σ_f, we pick the last state of the violation of φ. We observe that all the states after the bad state in σ_f are livelocks because otherwise our violation would be extensible. Therefore, we can safely block those states as well.

4.3.4 Empirical Evaluation

We evaluated the performances of our algorithm by comparing it to the other two well known algorithms used in the reduction of LTL to invariant model checking: k-liveness and liveness to safety (adapted for infinite state systems as proposed in [133]). All the algorithms are implemented inside the nuXmv symbolic model checker. Since the k-liveness algorithm is not able to disprove properties, it has been executed in lockstep with BMC. In our implementation, the algorithms are constructed on top of MathSAT5[105] SMT-solver, which supports combinations of theories such as $\mathcal{LIA}$, $\mathcal{LRA}$ and $\mathcal{EUF}$. Moreover, k-liveness and the algorithms presented in this paper are built on top of an infinite state version of the IC3 algorithm [102].

The experiments[3] were run in parallel on a cluster with nodes with Intel Xeon CPU 6226R running at 2.9GHz with 32CPU, 12GB. The timeout for each run was one hour and the memory cap was set to 1GB.

We carried out the experimental evaluation comparing the execution time of each algorithm on each instance. In the remainder of this section, we denote the relative safety algorithm with the lookahead construction as rels-la, the relative safety algorithm without lookahead construction as rels-no-la, the liveness to safety algorithm as l2s and kliveness algorithm as klive.

Benchmarks

The benchmarks have been taken from different sources: (i) Assume-guarantee contracts models of OCRA[92] representing a simplified Wheel Brake System, Redundant Sensors and other models. (ii) A subset of finite state models from NuSMV examples. (iii) Automatically discretized timed nuXmv benchmarks [98] such as the emergency diesel and a modified version of the Fischer algorithm. (iv) Handcrafted parametrized benchmarks to study the scalability of rels-la compared to kliveness and liveness to safety.

Overall, we collected roughly 850 formulae to be verified by each algorithm. The evaluation considered both valid and invalid properties. Regarding the structure of the

[3]The results of the experimental evaluation can be found at `https://es-static.fbk.eu/people/bombardelli/papers/ifm23/ifm23.tar.gz`

formulae, we considered both pure safety properties (i.e. properties in which $\alpha := \top$) and relative safety properties (with the form $\alpha \rightarrow \varphi$). The structure of α was not limited as well, we considered cases in which α was pure safety, pure liveness and also a generic LTL property. Regarding the type of models, they are both from discrete and timed benchmarks, with variables ranging from *enumeratives*, *Booleans*, *integers* and *reals*.

In the following, we describe the handcrafted parameterised benchmarks.

Asynchronous Discrete Bounded Response: This benchmark considers the asynchronous composition of l bounded response components. Each component receives the same input $trig$ and if it keeps receiving the input for n steps, then in m steps it set the variable res to *true*. When a component is not running it stutters i.e. it ignores inputs and its variable remain unchanged.

The property ψ in form $\alpha \rightarrow \varphi$ is as follows: α is a liveness property that guarantees that each component will be scheduled infinitely often i.e. $\alpha := \mathbf{GF}run_i$. φ asks that if a trigger remains true for g_n steps, then in g_m steps a variable result becomes true i.e $\varphi := \mathbf{G}(\mathbf{G}^{\leq g_n} trig \rightarrow \mathbf{F}^{\leq g_m} res)$. The parameters g_n and g_m depends on the parameters l, n and m. Each parameter n, m, l is instantiated with different values; moreover, the model gives different assignments to g_n and g_m to produce both valid and invalid properties.

Monitor Sensor This benchmark is composed of a sensor and a monitor. The sensor reads a *real* input signal and, if there are no faults, it returns a signal with a perturbation bounded by a constant; otherwise, it replicates its last value forever. The monitor reads the sensor output every q time unit and if it reads the same value twice it raises an alarm.

The property to be verified is as follows. Assuming non-Zenoness of time, if there is a fault an alarm is raised in at most p time units. Parameters p, q are instantiated with different values and as both *integers* and *reals*. Depending on the assignments of p and q the property can be either valid or invalid. Moreover, we considered a variation of the model that introduces livelocks.

Experimental Results and Analysis

Figure 4.3 shows scatter plots comparing the rels-la algorithm with k-liveness, liveness to safety and rels-no-la in terms of execution time. The y-coordinate of each point

represents the execution time of rels-la in that particular instance; conversely, the x-coordinate represents the execution time of the other algorithm. When a point is below the diagonal, it means that rels-la is faster on the given instance. When the point is above the diagonal the other algorithm is faster than rels-la.

Comparison with Liveness to Safety: Figure 4.3b and Figure 4.3e highlight significant differences in the performance between l2s and rels-la. Although there are few instances that can be solved by l2s in a few seconds and are not solved by rels-la, the vast majority of the instances timed out with liveness to safety. This result is due to the weaknesses that liveness to safety has with infinite-state systems because it needs a finite-state abstraction.

Comparison with K-liveness: As Figure 4.3d shows, klive appears to be more performant than rels-la with invalid instances even though rels-la still outperforms klive in several invalid instances and overall is able to solve more invalid instances. We think that in many cases BMC, which runs in lockstep with klive, can be significantly faster in finding counterexample because rels-la has an overhead of time to check the extendibility of σ_f and can still need multiple interactions to converge. On the other hand, when the liveness part of α is significant, rels-la outperforms klive.

Regarding the valid instances, Figure 4.3a shows that rels-la outperforms in most cases klive. Few instances times out with rels-la while they are solved in a short time with the klive algorithm and rels-no-la. Apparently, in these instances, the lookahead counter prevents ic3 to converge. Moreover, we observed that with small variation over parameters of the monitor sensor models neither rels-la nor rels-no-la are able to converge. This result occurs because the algorithm keeps finding counterexamples in an unfair region, and it is not able to generalise them. Moreover, the lookahead construction is not effective in this case because the trace extendibility depends on the assignment of two parameters. When livelocks are removed from the model the algorithm converges easily.

Relative safety algorithm performs significantly better than k-liveness with benchmarks from *Asynchronous discrete bounded response* and with the valid instances of *mutual exclusion* benchmarks. We think that the reasons why this happens are the following. In these benchmarks, φ is slightly complicated; this characteristic penalizes k-liveness because the construction of the model with the counter needs a degeneralization of $M_{\neg\varphi}$ to produce single fairness. This situation hinders ic3 in the process of inductive invariant construction. On the opposite side, the construction of *SafetyLTL2STS* is

quite effective and does not suffer the overhead of the liveness conditions of α; therefore, invariant checking is significantly faster with our algorithm with these instances.

In many instances, the two algorithms appear to have comparable results. We observed that is often the case when φ and α are both simple safety properties, the model does not contain many livelocks and the fairness is either absent or simple. Our theory is that for such instances k-liveness construction is similar to ours and the counter does not provide an overhead with valid formulae. However, when the formulae are invalid, we observe that such instances are easier to verify with BMC, which runs in lockstep with klive.

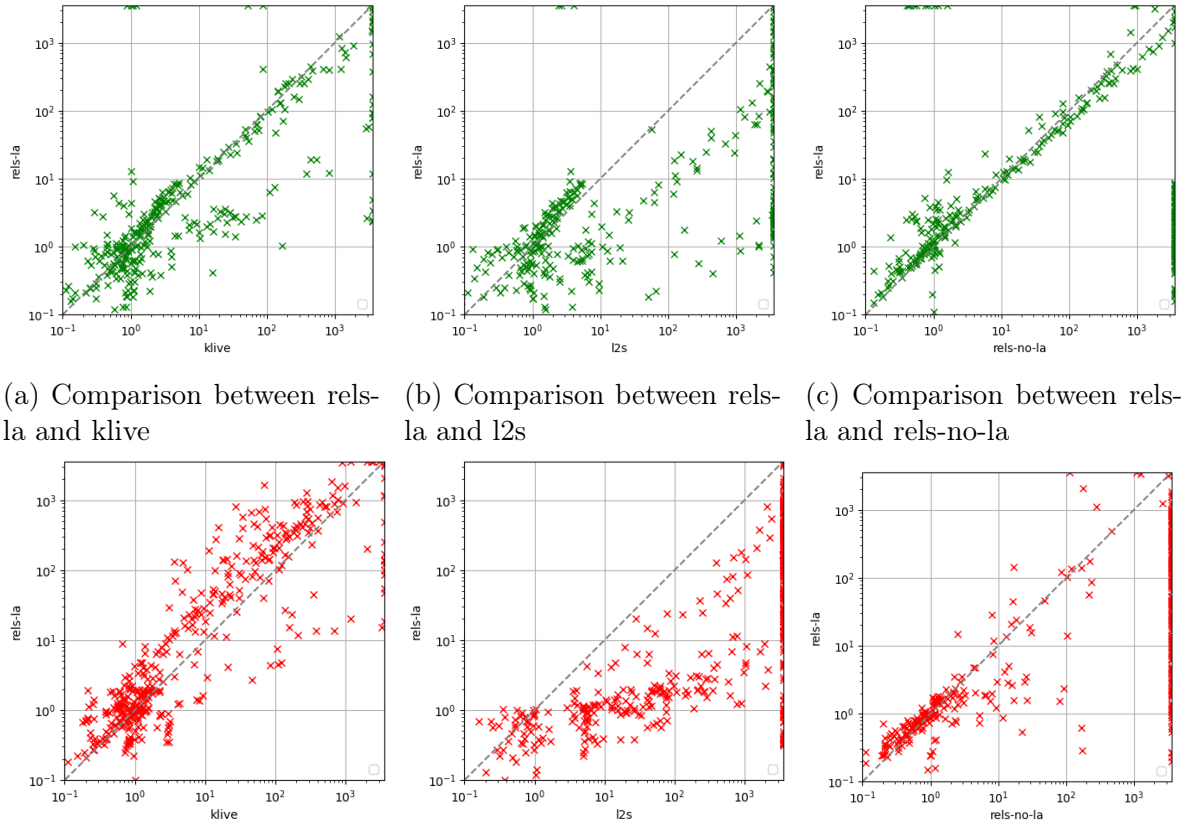
Impact of Lookahead Construction: The lookahead construction forces each finite counterexample to be extended with n steps. As Figure 4.3c and Figure 4.3f shows, this optimization provide a significant performance boost. In particular with the instances that contain several deadlocks or livelocks in M'. The intuition behind this result is that with a lightweight overhead in the invariant verification, the algorithm is able to discard a potentially infinite amount of livelocks that could potentially block the algorithm forever. Moreover, the livelocks caused by prophecy variables for X are all discarded by a sufficiently large lookahead.

4.4 Related Works

This section compares the contribution of this chapter with the related works. A broader view of the state-of-the-art is instead presented in Chapter 3.

The idea of Verification Modulo Theory is not new and has been presented in various works [87, 282, 98, 100]. The merit of this work in that sense is a unified view and consideration of both finite and infinite semantics; in particular, with a reduction to $\mathrm{LTL}_f(\mathcal{T})$ of SafetyLTL($\mathcal{T}$) and the truncated semantics of LTL($\mathcal{T}$). Moreover, as far as we know this is the first work also considering relative safety properties in the context of verification modulo theory.

In [219], Kupferman and Vardi studied the verification of safety properties; they introduce the notion of informative path to formally denote a computation violating a safety property. That work also defines a PSPACE procedure to check whether a property is safety. Later, in [223], the same approach has been applied in a BDD-based algorithm. Another related work for the verification of safety property is [114]. In [114], LTL formulae are translated into circuits. If the property is syntactically safety, it is possible to reduce its verification to reachability checking; otherwise, the



(a) Comparison between rels-la and klive

(b) Comparison between rels-la and l2s

(c) Comparison between rels-la and rels-no-la

(d) Comparison between rels-la and klive

(e) Comparison between rels-la and l2s

(f) Comparison between rels-la and rels-no-la

Figure 4.3: Scatter plots comparing rels-la with the other algorithms. Plots with green crosses represent valid properties while red crosses represent invalid properties.

verification procedure involves also liveness checking. In all these works, the reduction to reachability is limited to safety properties, while relative safety is not considered.

The approach of AGREE [126] proposes a semantics for contracts that can be seen as extension of the reduction for specific pattern of safety assumptions. However, such reductions assume implicitly the absence of deadlocks and livelocks. We are not aware of other approaches that use the assumptions explicitly and exploit the notation of relative safety for generic assumptions.

Other well-known reduction to safety are used to prove full LTL formulae. In [44], the approach duplicates the state variables and add monitoring constraints to look for a fair lasso. It works only for finite-state systems and the resulting reachability condition is related to the structure of the lasso-shaped counterexample. K-liveness [116] is more similar to our reduction as it counts and looks for a bound to the number of times the fairness condition of counterexamples can be visited. If the property is safety,

the fairness cannot be reached or can be reached only once, depending on the actual construction. However, in the case of properties in the form $\alpha \rightarrow \phi$, the fairness conditions related to α are mixed with those of $\neg\phi$ in looking for a counterexample. So, in a sense, our approach gives more priority to look for a counterexample to ϕ, considering the fairness of α in a second step.

Part III

Asynchronously Composing Trace Properties

Chapter 5

Asynchronous Composition of LTL modulo theory

Verifying asynchronous software systems presents substantial challenges due to the nondeterministic interleaving of concurrently executing components and their interactions through shared resources. As discussed in Chapter 1, one of the central obstacles in verifying such systems lies in scalability—especially when using expressive temporal logics like LTL($\mathcal{T}$), introduced in Part II, which allow reasoning over complex data domains and behaviours. While model checking offers a rigorous foundation, it often struggles to scale in the face of concurrency and intricate system architectures.

To address these challenges, this chapter explores how compositional reasoning can alleviate the scalability limitations of monolithic verification. Instead of verifying the entire system as a whole, we decompose the verification task by reasoning about individual components and their local specifications. By doing so, we aim to retain the expressiveness of temporal logic while achieving more scalable verification procedures.

We build on classical results on compositional verification [266], where local properties $\varphi_1, \ldots, \varphi_n$ are verified for components $M_1, \ldots, M_n$, and are then composed to establish that the global system $\gamma_S(M_1, \ldots, M_n)$ satisfies a global property φ. While such reasoning is relatively straightforward in synchronous settings, asynchronous systems present a more intricate landscape: components execute independently, may be scheduled irregularly, and interact via shared memory or I/O ports. These features complicate both the semantics of local properties and their composition.

In this chapter, we propose a general framework for the asynchronous composition of LTL properties, where each local property is interpreted over the local trace of a component, taking into account possible interruptions, scheduling constraints, and

communication via data ports. To capture the possibility of finite executions (e.g., due to unfair scheduling or permanent faults), we adopt the truncated weak semantics from [153], as previously introduced in Section 4.1. This allows local specifications to be interpreted over both finite and infinite traces, providing a flexible foundation for modeling realistic asynchronous behaviours.

Crucially, our composition procedure is compatible with the symbolic and automata-theoretic foundations developed in earlier chapters, ensuring that it remains applicable in the context of expressive LTL($\mathcal{T}$)-based model checking. The approach is based on a syntactic rewriting $\mathcal{R}^*$ of local temporal properties into properties over the composite model, conjoined with fairness-like constraints (ψ_{cond}) to handle inactive components. We also present an optimized variant tailored to the case where all components are assumed to run infinitely often.

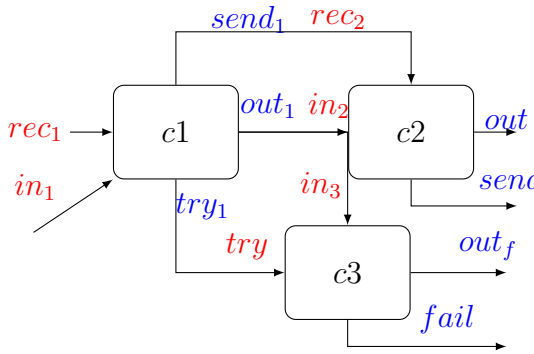
The proposed method has been implemented in the OCRA tool [92], which supports extended LTL syntax and integrates with the nuXmv [87] model checker. We evaluate the framework on both synthetic patterns and industrial models from the automotive domain [91], highlighting the impact of our optimizations and the tradeoffs introduced by different semantic interpretations.

Contributions The main contribution of this chapter is the definition of a rewriting-based technique to verify compositional asynchronous systems generally. Furthermore, we provide an additional optimized rewriting technique to cover the case in which local components are assumed to run infinitely often. The main advantages of our compositional approach are the following:

- It supports asynchronous communication between data ports.
- It supports generic scheduling constraints expressible through LTL formulae.
- It supports both finite and infinite executions of local components making it a suitable approach for safety assessment as well.

The approach have been introduced for LTL with event-freezing function [282] in [59]. Then, a complete version considering the possibly-finite semantics have been included in [62]. The work presented in this chapter is the result of a collaborative effort between the author and Stefano Tonetta.

Motivating examples We propose two examples to motivate our work. The first model is a toy example representing a system that tries to send a value to a network



$$\varphi_{c1} \doteq \mathbf{G}(rec_1 \rightarrow out'_1 = in_1 \wedge \mathbf{X}((try_1 \wedge out'_1 = out_1)\ \mathbf{U}\ send_1))$$
$$\varphi_{c2} \doteq \mathbf{G}(rec_2 \rightarrow out2' = in2 \wedge \mathbf{X}send)$$
$$\varphi_{c3} \doteq \mathbf{G}(try \rightarrow out'_f = in_3 \wedge \mathbf{X}fail)$$
$$\varphi \doteq \mathbf{G}((rec_1 \wedge in_1 = v) \rightarrow \mathbf{F}(send \wedge out = v \vee fail \wedge out_f = v))$$
$$\alpha \doteq \mathbf{G}(in_1 \rightarrow run_1) \wedge \mathbf{G}(send_1 \rightarrow run_2) \wedge \mathbf{G}(\mathbf{H}^{\leq p} try_1 \rightarrow run_3)$$

Figure 5.1: Figure representing the sender model. The colour red represents input variables while the colour blue represents the output variables.

while the second is a real model coming from the automotive domain. Both models can be naturally represented through possibly finite scheduling of local components.

Sender We now model a three-component system that represents a system that receives a message and tries to send it through a network. The component either successfully delivers the message or fails to send it and logs the message. The example is represented in Figure 5.1. This example is composed of three components: Component $c1$ receives a message rec_1 and an input in_1 and tries to send the value to component $c2$. If $c1$ is eventually able to send *send* message through the network, then $c2$ will run and will output the original input. The network guarantees that eventually, $c1$ will be able to send the message to $c2$; however, if $c1$ runs only finitely many times, $c3$ at some point will report a failure. The global property states that if an input message is received, it is either eventually delivered as output or an error occurs. If $c1$ runs infinitely often, the global property is satisfied without the need for $c3$.

Automotive compositional contract Another interesting example comes from the automotive domain. In [91], the EVA framework was proposed for the compositional verification of AUTOSAR components. That work used the rewriting technique we proposed in [59] to verify the correct refinement of contracts defined as a pair of LTL properties.

Due to the complexity of the model, we omit a full description of the system, the specifications, and the properties. We focus on a specific requirement of the system that states that the system shall brake when the Autonomous Emergency Braking module

gets activated. A simplified version of the specification is defined by the following LTL formula:

$$\mathbf{G}(\mathbf{X}(aeb\_breaking.status \neq 0) \rightarrow \mathbf{F}^{\leq 2} Brake\_\_In\_\_BrakeActuator \neq 0)$$

The specification is entailed by a brake actuator component that is composed of the actual actuator and a watchdog. The actuator is scheduled every time a signal is received in input while the watchdog is scheduled periodically. By reasoning over finite executions of components, it is possible to verify whether the global specification is valid even if at some point the Actuator stop working. We omit the detailed structure and specification of the sub-components (actuator and watchdog), for a complete view please refer to [91] or to the experimental evaluation.

Outline The rest of the chapter is organized as follows. In Section 5.1, we formalize the problem; in Section 5.2, we define the rewriting approach, its basic version, its complete version and an optimized variation that is suited for infinite executions only; in Section 5.3, we report on the experimental evaluation; finally, in Section 5.4, we compare our contribution with related works.

5.1 Compositional Reasoning

5.1.1 Formal Problem

Compositional verification proves the properties of a system by proving the local properties on components and by checking that the composition of the local properties satisfies the global one (refer to Chapter 3 for a detailed discussion). This reasoning is expressed formally by inference 5.1, which is parametrized by a function γ_S that combines the component's implementations and a related function γ_P that combines the local properties.

Inference 5.1 (Generic Compositional Inference [266]) *Let $M_1, M_2, \ldots, M_n$ be a set of n components, $\varphi_1, \varphi_2, \ldots, \varphi_n$ be local properties on each component, γ_S is a function that defines the composition of $M_1, M_2, \ldots, M_n$, γ_P combines the properties depending on the composition of γ_S and φ a property such that following inference is true:*

$$\frac{M_1 \models \varphi_1, M_2 \models \varphi_2, \ldots, M_n \models \varphi_n \qquad \gamma_S(M_1, M_2, \ldots, M_n) \models \gamma_P(\varphi_1, \varphi_2, \ldots, \varphi_n) \quad \gamma_P(\varphi_1, \varphi_2, \ldots, \varphi_n) \models \varphi}{\gamma_S(M_1, M_2, \ldots, M_n) \models \varphi}$$

The problem we address in this chapter is to define proper γ_S, γ_P representing the composition of possibly terminating components and temporal properties such that the inference rule holds.

5.1.2 Interface Transition Systems

In this chapter, we represent I/O components as Interface Transition Systems, a symbolic version of interface automata [135] that considers I/O variables instead of I/O actions.

Definition 5.1 (Interface Transition System) *We define the Interface Transition System (ITS) M as the tuple*
$M = \langle V^I, V^O, I, T, \mathcal{SF} \rangle$ where:

- *V^I is the set of input variables while V^O is the set of output variables where $V^I \cap V^O = \emptyset$.*
- *$V \doteq V^I \cup V^O$ denotes the set of the variables of M*
- *I is the initial condition, a formula over V^O,*
- *T is the transition condition, a formula over $V \cup V^{O'}$ where $V^{O'}$ is the primed versions of V^O*
- *$\mathcal{SF}$ is the set of strong fairness constraints, a set of pairs of formulae over V.*

A symbolic transition system with strong fairness $M = \langle V, I, T, \mathcal{SF} \rangle$ is an interface transition system without input variables (i.e., $\langle \emptyset, V, I, T, \mathcal{SF} \rangle$).

The language of ITS is defined as for STS except for strong fairness constraints, formally requiring that every infinite trace satisfy the following constraint: For each $\langle f_A, f_G \rangle \in \mathcal{SF}$: if for all i, there is a $j \geq i$ s.t. $\sigma, j \models f_A$, then for all i there is a $j \geq i$ s.t. $\sigma, j \models f_G$.

Definition 5.2 (ITS Compatibility) *Let $M_1, \ldots, M_n$ be n ITS, they are said compatible iff they share respectively only input with output (i.e. $\forall i \leq n, j \leq n$ s.t. $i \neq j$: $V_i \cap V_j = (V_i^O \cap V_j^I) \cup (V_i^I \cap V_j^O)$).*

5.1.3 Weak Semantic Choice

In this work, we adopt the weak semantics of temporal operators described by Definition 4.3. This choice is motivated by the characteristic of our setting. Specifically, during composition, local traces may be truncated by the scheduler or due to a component failure, and these truncated traces remain relevant and may contribute to the satisfaction of the system-level property. In a sense, we need to pertain the safety part of the property ensured by the finite trace, while disregarding the liveness part of it. The weak semantics proposed by [153] allows us to account for this, ensuring that truncated traces are not disregarded in the logical reasoning process.

Another semantics decision we are going to take is to consider input variables as "next". The intuition is that we are interested in evaluating input over a local trace only during transitions, and when a trace terminates, we don't want to evaluate inputs there. Therefore, when we consider predicates that contain input variables v_i, we will treat them as if they are "next" predicates.

As a motivating example, consider the local property φ_{c1} in Figure 5.1. This property states that whenever component $c1$ receives an input in_1 via the rec_1 signal, it will attempt to propagate the message to component $c2$ until it succeeds. Suppose that $c1$ receives the value 3 at time 0 and begins this propagation. If a failure occurs or the scheduler stops activating $c1$, its execution may be prematurely truncated, and only a finite prefix of its local trace is available. Still, this partial trace captures the component's behaviour up to the end of its execution– specifically, in this case, that $c1$ initiated the process of sending the input to $c2$. Weak semantics ensures that such truncated traces are preserved in system-level reasoning, without requiring that liveness properties be fully realized. If (on the composition) stronger guarantees are required– e.g., to ensure infinite progress or rule out failures—this can be encoded explicitly in the environment assumption, such as $\alpha' := \alpha \wedge \mathbf{GF} run_{c1}$, ensuring fair scheduling and infinite execution.

At the global level, traces may also be finite to support hierarchical composition. In such cases, weak semantics provides a conservative and practical interpretation: a property satisfied on a finite trace indicates that no counterexample has been found yet, though one may still arise on an extension. Consider the system-level property $\varphi := \mathbf{G}((rec_1 \wedge in_1 = v) \rightarrow \mathbf{F}(send \wedge out = v \vee fail \wedge out_f = v))$ from Figure 5.1. In this case, since φ is a liveness property, every finite trace satisfies weakly φ. In general, the only problematic cases would involve finite traces that actually constitute counterexamples, for instance due to reaching a deadlock state that violates liveness.

However, removing deadlocks is computationally costly, especially in a compositional setting. Thus, adopting weak semantics is a conscious compromise: it avoids penalizing valid partial behaviours, and in practice, systems are typically defined to avoid such pathological deadlock cases.

Distinguishing Between Input and Output Recall the notion of truncated semantics of Definition 4.3. We overload the notion of trace providing a distinction between *input* and *output* variables. The idea is that if, a local component reads the input to decide the next state and output. When reasoning over infinite traces, this representation is equivalent to the "standard" one. However, when dealing with finite semantics, given a predicate containing input variables, we need to define its semantics at the end of the trace. From now on, we denote with $Pred^I$ any predicate containing at least an input variable or a primed variable (next). Formally, we define the semantics of input predicates over traces as follows:

$$\sigma, i \models_{t^-} Pred^I(u_1, \ldots, u_n) \text{ iff } \quad i \geq |\sigma| - 1 \text{ or } \langle \sigma^+(i), \mathcal{M} \rangle \models Pred^I(u_1, \ldots, u_n)$$
$$\sigma, i \models_{t^+} Pred^I(u_1, \ldots, u_n) \text{ iff } \quad i < |\sigma| - 1 \text{ and } \langle \sigma^+(i), \mathcal{M} \rangle \models Pred^I(u_1, \ldots, u_n)$$

5.1.4 Asynchronous Composition of Interface Transition Systems

We now provide the notion of asynchronous composition ($\otimes$) that will be used as the composition function (γ_S) of Inference 5.1.

Definition 5.3 (Asynchronous Composition of ITS) *Let $M_1, \ldots, M_n$ be n compatible interface transition systems, let $run_1, \ldots, run_n$ be n Boolean variables not occurring in $M_1, \ldots, M_n$ (i.e. $run_1, \ldots run_n \notin \bigcup_{1 \leq i \leq n}(V_i^I \cup V_i^O)$) and $end_1, \ldots, end_n$ be n Boolean variables not occurring in $M_1, \ldots, M_n$ i.e. (i.e. $end_1, \ldots end_n \notin \bigcup_{1 \leq i \leq n}(V_i^I \cup V_i^O)$). $M_1 \otimes \cdots \otimes M_n = \langle V^I, V^O, \bigwedge_{1 \leq i \leq n} I_i, T, \mathcal{SF} \rangle$ where:*

$$V^I = (\bigcup_{1 \leq i \leq n} V_i^I \cup \{run_i\}) \setminus (\bigcup_{1 \leq i < n} \bigcup_{i < j \leq n} V_i \cap V_j)$$
$$V^O = (\bigcup_{1 \leq i \leq n} V_i^O \cup \{end_i\}) \cup (\bigcup_{1 \leq i < n} \bigcup_{i < j \leq n} V_i \cap V_j)$$
$$T = \bigwedge_{1 \leq i \leq n} ((run_i \rightarrow T_i) \wedge \psi_{cond}^{M_i} \wedge (end_i \leftrightarrow (end_i' \wedge \neg run_i)))$$
$$\mathcal{SF} = \bigcup_{1 \leq i \leq n} (\{\langle \top, run_i \vee end_i \rangle\} \cup \{\langle run_i \wedge \varphi_a, run_i \wedge \varphi_g \rangle | \langle \varphi_a, \varphi_g \rangle \in \mathcal{SF}_i\})$$

where $\psi^{M_i}_{cond} \doteq \neg run_i \rightarrow \bigwedge_{v \in V_i^O} v = v'$.

The operator $\otimes$ provides a notion of asynchronous composition of Interface Transition Systems based on interleaving. Each component can either run (execute a transition) or stutter (freeze output variables). Contrary to our previous work[59], this new definition allows for finite executions of local components. For each i, we introduce the variable run_i representing the execution of a transition of M_i; furthermore, we introduce the prophecy variables end_i that monitor whether a component will execute new transitions in the future i.e. end_i is equivalent to $\mathbf{G}\neg run_i$.

Definition 5.4 (Projection) *Let σ be a trace of $M = M_1 \otimes \cdots \otimes M_n$ with $i \leq n$. We define the projection function $Pr_{M_i} : \Pi(V) \rightarrow \Pi(V_i)$ mapping σ to a trace σ_i as follows:*

$$Pr_{M_i}(\sigma) = \begin{cases} s_{map_0}(V_i), \ldots & \text{If } \sigma \text{ is infinite and } \sigma \models \boldsymbol{GF} run_i \\ s_{map_0}(V_i), \ldots, s_{map_{n-1}}(V_i), s_{map_n}(V_i^O) & \text{Otherwise} \end{cases}$$

where map_k is the sequence mapping each position of σ_i into a position of σ as follows:

$$map_k \doteq \begin{cases} k & \text{If } k < 0 \\ map_{k-1} + 1 & \text{If } k \geq |\sigma| - 1 \\ k' \mid k' > map_{k-1}, \sigma, k' \models run_i \text{ and } \forall_{map_{k-1} < j' < k'} \sigma, j' \not\models run_i & \text{Otherwise} \end{cases}$$

Moreover, we define the inverse operator of Pr, denoted by Pr^{-1}:

$$Pr^{-1}_{M_i}(\sigma) = \{\sigma' | Pr_{M_i}(\sigma') = \sigma\}$$

Definition 5.4 provides a mapping between the composed ITS and the local transition system. The function map_k maps indexes of the local trace to the indexes of the global trace. For each k in the range of the trace excluding the last point (i.e. $[0, |\sigma|-1)$) map_k represent the k-th occurrence of *run* counting from 0. It should be noted that map_0 is not 0 but the first point of the trace in which run_i is true ($k' | map_{-1} < k', \sigma, k' \models run_i$ and $\forall map_{-1} < k'' < k'\ \sigma \not\models run_i$). Finally, the projection of a global trace to a local trace is defined using *map*. A graphical representation of projection is shown in Figure 5.2. We now show that the language of each local ITS M_i contains the language of the composition projected to the local ITS. From this result we can derive a condition for γ_P such that Inference 5.1 holds. We do that by considering a mapping between local trace and global traces that follows the same mapping map_i.

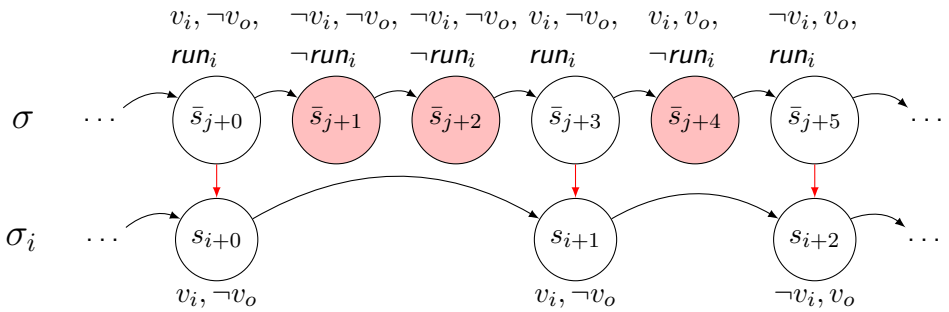


Figure 5.2: Graphical view of trace projection. White states of σ represent the states of the sequence map_i. Pink states represent states in which the local component stutters. Red arrows represent the link between the states of σ and the states of σ_i formally represented by map_k.

Theorem 5.1 (Projection Preserves Local Traces) *Let* $M = M_1 \otimes \cdots \otimes M_n$*:*

$$\textit{For all } 1 \leq i \leq n : \mathcal{L}(M_i) \supseteq \{Pr_{M_i}(\sigma) | \sigma \in \mathcal{L}(M)\}$$

Proof: Given a trace $\sigma \in \mathcal{L}(M), \sigma_i \doteq Pr_{M_i}(\sigma)$. We prove the theorem by induction on the length of σ_i. The inductive hypothesis states that if $|\sigma_i| > k+1$ and $\sigma_i^{0\ldots k} \in \mathcal{L}_{fin}(M_i)$, then $\sigma_i^{0\ldots k+1} \in \mathcal{L}_{fin}(M_i)$.

- Base case:
  By definition $I \doteq \bigwedge_{1 \leq i \leq n} I_i$ and $\sigma_i(0) = \sigma(map_0)(V_i)$. We derive that $\sigma_i, 0 \models I_i \Rightarrow \sigma_i^{0\ldots 0} \in \mathcal{L}_{fin}(M_i)$. It should be noted that this holds also if $|\sigma_i| = 1$ and $\sigma, |\sigma| - 1 \not\models run_i$ because I_i is a proposition on symbols of V_i^O.

- Inductive case:
  By definition $\sigma_i(k) = \sigma(map_k)(V_i)$ where map_k is the k-th position s.t. run_i holds. Therefore, $\sigma(map_k)\sigma(map_k + 1) \models run_i \rightarrow T_i \Rightarrow \sigma(map_k)\sigma(map_k + 1) \models T_i$. By $\psi_{cond}^{M_i}$ we obtain that $\sigma_{map_{k+1}}(V_i^O) = \sigma_{map_k+1}(V_i^O) = \sigma_i(k+1)(V_i^O)$. Since T_i reasons over symbols of $V \cup V^{O'}$, then $\sigma(map_k)\sigma(map_k + 1) \models T_i \Leftrightarrow \sigma_i(k)\sigma_i(k + 1) \models T_i$. Therefore, $\sigma_i, k \models T_i$ which guarantees that $\sigma_i^{0\ldots k+1} \in \mathcal{L}_{fin}(M_i)$.

We proved the theorem for finite traces, we now extend the proof to consider infinite traces. To do so, it is sufficient to prove that each infinite trace σ_i satisfies the strong fairness conditions of $\mathcal{SF}_i$ since it already satisfies I_i and T_i.

Since $\sigma_i = Pr_{M_i}(\sigma)$ and $\sigma \in \mathcal{L}(M)$, $\sigma \models \bigwedge_{\langle f_a, f_g \rangle \in \mathcal{SF}} (\mathbf{GF} f_a \rightarrow \mathbf{GF} f_g)$. In particular, due to the composition, $\sigma \models \bigwedge_{\langle f_a, f_g \rangle \in \mathcal{SF}_i} (\mathbf{GF}(run_i \wedge f_a) \rightarrow \mathbf{GF}(run_i \wedge f_g))$ The projection defines the sequence $map_0, \ldots$ representing the points in which run_i

holds; therefore, for all $\langle f_a, f_g \rangle \in$if for all k there exists $j \geq k$ s.t. $\sigma, map_k \models f_a$ then for all k' there exists $j' \geq k'$ s.t. $\sigma, map'_k \models f_g$. From the projection definition then $\sigma_i \models \mathbf{GF} f_a \rightarrow \mathbf{GF} f_g$ for each $\langle f_a, f_g \rangle \in \mathcal{SF}_i$. □

Definition 5.5 (γ_S) *Let $M_1, \ldots, M_n$ be n ITS, we define γ_S as follows:*

$$\gamma_S(M_1, \ldots, M_n) \doteq M_1 \otimes \cdots \otimes M_n$$

From Theorem 5.1, we obtain a composition γ_S for which each projected global trace is an actual behaviour of a local trace. Therefore, when we consider the global traces, we do not introduce new local traces which cannot be witnessed locally.

To complete our compositional reasoning, we need to define the function γ_P such that Inference 5.1 holds. To do so, Section 5.2 provides a rewriting technique that maps each local trace satisfying φ_i to a global trace satisfying γ_P.

5.2 Rewriting Based Approach for the Composition of LTL modulo Theory

In this section, we introduce a rewriting-based approach for the composition of local properties. In section 5.2.1, we present the rewriting of from LTL($\mathcal{T}$) to LTL-EF($\mathcal{T}$) [282] that maps local properties to global properties with proofs and complexity results. In section 5.2.2, we propose an optimized version of the rewriting. Finally, in section 5.2.3, we introduce a variation of the optimized rewriting tailored for infinite executions of local properties i.e. in which the semantics is the one of standard LTL because we assume fairness of run_i.

To simplify the notation, we assume to be given n interface transition systems $M_1, \ldots, M_n$, a composed ITS $\mathcal{C} = M_1 \otimes \cdots \otimes M_n$, a trace σ of M_i with $i < n$, a local property φ and a local term u. For brevity, we refer to $\mathcal{R}_{M_i}$, $\mathcal{R}^*_{M_i}$, ${\mathcal{R}^\theta}_{M_i}$, $Pr^{-1}_{M_i}$, Pr_{M_i}, end_i and run_i as respectively $\mathcal{R}, \mathcal{R}^*, \mathcal{R}^\theta, \mathcal{R}^{\theta *}, Pr^{-1}, Pr$, *end* and *run*. Moreover, we will refer to $map_0, map_1, \ldots$ as the sequence of definition 5.4. As in Section 5.1.3, we denote input predicates with apex I: $Pred^I$ and the output predicates and terms with apex O: $Pred^O$. Finally, we denote $sgn \in \{-, +\}$.

At next operator [282] In the rest of the Section, we will use the additional operator $@\tilde{\mathbf{F}}$ introduced in [282] for the LTL with Event Freezing Function logic. This operator

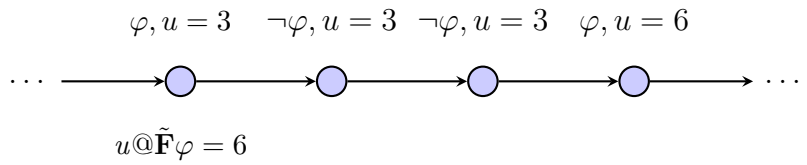


Figure 5.3: Graphical representation of $@\tilde{\mathbf{F}}$.

will be used to rewrite next variables. Intuitively, this operator returns the value of a term u at the next occurrence of a formula φ (excluding the current one); if there is no future occurrence of φ, the variable can take any value. Figure 5.3 gives an intuitive view of the semantics of the operator.

The semantics of the operator is the following:

$$\sigma^{\mathcal{M}}(i)(u@\tilde{\mathbf{F}}(\varphi)) = \sigma^{\mathcal{M}}(k)(u) \text{ if there exists } k > i \text{ such that, for all } l, i < l < k,$$
$$\sigma, \mathcal{M}, l \models_{t^+} \neg\varphi \text{ and } \sigma, \mathcal{M}, k \models_{t^+} \varphi$$

As shown in [282], we can remove the operator using the following rewriting introducing a monitor variable as follows:

$$\mathcal{ATN}(\psi, u@\tilde{\mathbf{F}}\phi) \doteq \psi\lceil p_{u@\tilde{\mathbf{F}}\phi}/u@\tilde{\mathbf{F}}\phi\rfloor \wedge \mathbf{G}(\mathbf{X}\phi \rightarrow p_{u@\tilde{\mathbf{F}}\phi} = u') \wedge \mathbf{G}(p'_{u@\tilde{\mathbf{F}}\phi} \neq p_{u@\tilde{\mathbf{F}}\phi} \rightarrow \mathbf{X}\phi)$$

In the remainder of the Section, we will treat $@\tilde{\mathbf{F}}$ as is it was an operator of our logics.

5.2.1 LTL Compositional Rewriting

In this section, we propose a rewriting to asynchronously compose propositional LTL properties over Interface Transition System symbols.

The idea is based on the notions of projection (introduced in Definition 5.4) and asynchronous composition (introduced in Definition 5.3). We produce a rewriting $\mathcal{R}^*$ for φ such that each trace σ of M_i satisfies the rewritten formula iff the projected trace satisfies the original property.

Definition 5.6 *We define $\mathcal{R}$ as the following rewriting function:*

$$\mathcal{R}^-(Pred^I(u_1,\ldots,u_n)) \doteq \neg run \vee Pred^I(\mathcal{R}^-(u_1),\ldots,\mathcal{R}^-(u_n))$$
$$\mathcal{R}^+(Pred^I(u_1,\ldots,u_n)) \doteq run \wedge Pred^I(\mathcal{R}^+(u_1),\ldots,\mathcal{R}^+(u_n))$$
$$\mathcal{R}^{sgn}(Pred^O(u_1,\ldots,u_n)) \doteq Pred^O(\mathcal{R}^{sgn}(u_1),\ldots,\mathcal{R}^{sgn}(u_n))$$
$$\mathcal{R}^{sgn}(\varphi_1 \vee \varphi_2) \doteq \mathcal{R}^{sgn}(\varphi_1) \vee \mathcal{R}^{sgn}(\varphi_2)$$
$$\mathcal{R}^-(\neg\varphi) \doteq \neg\mathcal{R}^+(\varphi), \mathcal{R}^+(\neg\varphi) \doteq \neg\mathcal{R}^-(\varphi)$$
$$\mathcal{R}^-(\boldsymbol{X}\varphi) \doteq \boldsymbol{X}(state\,\mathbf{R}\,(\neg state \vee \mathcal{R}^-(\varphi)))$$
$$\mathcal{R}^+(\boldsymbol{X}\varphi) \doteq \boldsymbol{X}(\neg state\,\mathbf{U}\,(state \wedge \mathcal{R}^+(\varphi)))$$
$$\mathcal{R}^-(\varphi_1\,\mathbf{U}\,\varphi_2) \doteq (\neg state \vee \mathcal{R}^-(\varphi_1))\,\mathbf{U}\,((state \wedge \mathcal{R}^-(\varphi_2)) \vee \boldsymbol{Y}end)$$
$$\mathcal{R}^+(\varphi_1\,\mathbf{U}\,\varphi_2) \doteq (\neg state \vee \mathcal{R}^+(\varphi_1))\,\mathbf{U}\,(state \wedge \mathcal{R}^+(\varphi_2))$$
$$\mathcal{R}^{sgn}(\boldsymbol{Y}\varphi) \doteq \boldsymbol{Y}(\neg run\,\mathbf{S}\,(run \wedge \mathcal{R}^{sgn}(\varphi)))$$
$$\mathcal{R}^{sgn}(\varphi_1\,\mathbf{S}\,\varphi_2) \doteq (\neg state \vee \mathcal{R}^{sgn}(\varphi_1))\,\mathbf{S}\,(state \wedge \mathcal{R}^{sgn}(\varphi_2))$$
$$\mathcal{R}^{sgn}(func(u_1,...,u_n)) \doteq func(\mathcal{R}^{sgn}(u_1),...,\mathcal{R}^{sgn}(u_n))$$
$$\mathcal{R}^{sgn}(const) \doteq const$$
$$\mathcal{R}^{sgn}(v') \doteq \mathcal{R}^{sgn}(v)@\tilde{\boldsymbol{F}}(state)$$

where $state \doteq run \vee (\boldsymbol{Z}run \wedge end)$ *and* $\mathcal{R}(\varphi) = \mathcal{R}^-(\varphi)$.

$\mathcal{R}$ transforms the formula applying *run* and *state* to φ. The general intuition is that when *run* is true, the local component triggers a transition. Moreover, *state* represents a local state of σ in the global trace. It should be noted that the rewriting has to deal with the last state. Therefore, *state* is represented by a state that either satisfies *run* or is the successor of the last state that satisfies *run*.

For output predicates, the rewriting is transparent because the projection definition guarantees that each state i of σ has the same evaluation of each map_i state of σ^{ST}. Although this holds for input predicates as well, there is a semantic technicality that should be taken into account. Input predicates are evaluated differently on the last state: with *strong* semantics (+) they do not hold while on *weak* semantics (−) they hold. Since (i) the last state of a local trace might not be the last state of the global trace, and (ii) a variable that locally is an input variable might be an output variable of the composed system, we have to take input predicates into account inside the rewriting. Therefore, with *strong* semantics, the rewriting forces the predicate to hold in a transition i.e. when *run* holds ($\sigma, i \models_{t^{sgn}} Pred^I \Leftrightarrow i < |\sigma| - 1$ and

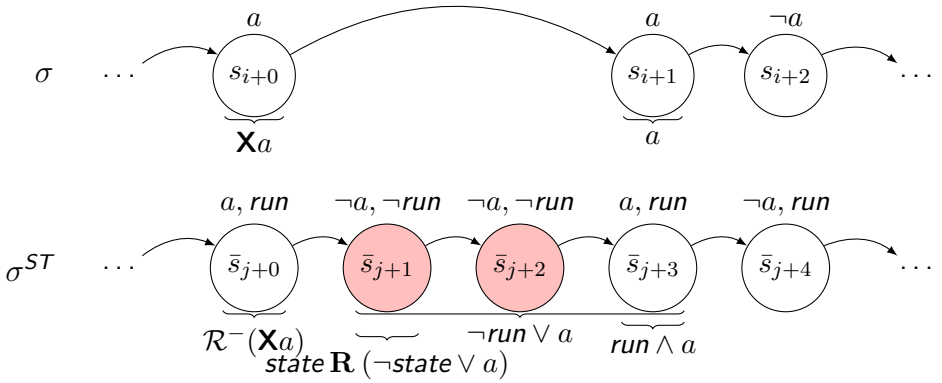


Figure 5.4: Graphical representation of rewriting of $\mathbf{X}a$. σ represents the local trace while σ^{ST} represents the trace of the composition. White states are states of local trace while pink states are states in which the local component is not running. In this example, a is an input variable which happens to be true in state s_i and s_{i+1} of local trace σ and state $\bar{s}_j$. To show the intuition of the rewriting, we show that Release operator permits to skip the pink states (which are not relevant w.r.t. the local trace). Finally, at state $\bar{s}_{j+3}$ a is evaluated since *run* is true.

$Pred^I(\sigma(i)(u_1), \ldots, \sigma(i)(u_n)))$. The *weak* semantics is handled similarly, except that it requires either *run* to hold or the predicate itself to hold.

Regarding $\mathbf{X}$, $\mathcal{R}^-$ needs to pass from point map_i to map_{i+1} and to verify that the sub-formula is verified in that state. The first $\mathbf{X}$ passes from map_i to $map_i + 1$. Then, the rewriting skips all states that are not map_{i+1} and "stops" in position map_{i+1}. Figure 5.4 shows an intuitive representation of the rewriting of $\mathbf{X}$ when the trace is not in its last state.

For Until formulae, the rewriting needs to "skip" all points that either do not belong to the local trace or satisfy the left part of the formula. Then, it must "stop" to a point that satisfies the right part and is a state in the local trace. To do so, it introduces a disjunction with *state* on the left side of the formula and a conjunction with *state* on the right side of the formula. When the rewriting and the formula are interpreted weakly, reaching the end of the local trace makes the formula true; therefore, $\mathbf{Y}end$ is also put in disjunction with the right side of the formula. Figure 5.5 shows an intuitive representation of the rewriting of $\mathbf{U}$ when the trace does not terminate before reaching b.

Lemma 5.1 *For all $\sigma^{ST} \in Pr^{-1}(\sigma)$, for all $i < |\sigma|$:*

$$\sigma, i \models_{t^{sgn}} \varphi \Leftrightarrow \sigma^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{sgn}(\varphi)$$

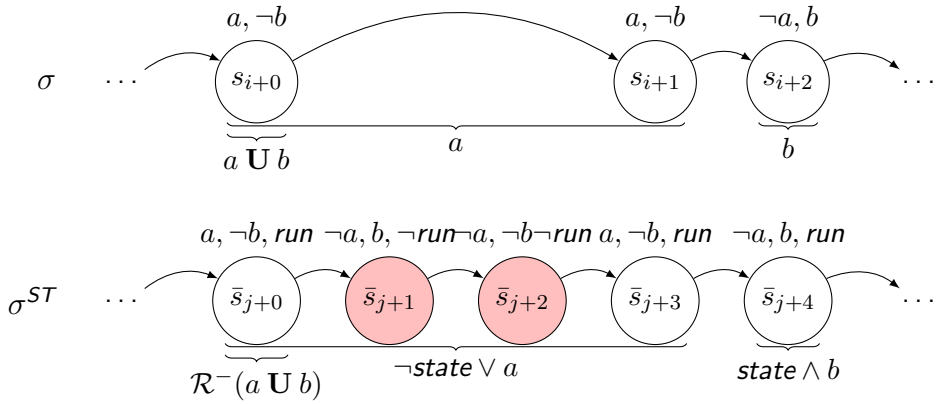


Figure 5.5: Graphical representation of rewriting of **U**. In this example both a and b variables are input variables. Local trace σ satisfies $a \mathbf{U} b$ at position i since a is true at state s_i, s_{i+1} while b is true at state s_{i+2}. The corresponding global trace contains 2 additional states $(\bar{s}_{j+1}, \bar{s}_{j+2})$ which are not considered in the rewriting since $\neg state$ holds these 2 states. Finally, the state $\bar{s}_{j+4}$
of the global trace satisfies both *state* and b.

Proof: We prove Lemma 5.1 by induction on the formula. The high level intuition is that, assuming that we are in a point map_i of the trace, we can "reach" the corresponding map_k using the syntactically rewriting for each temporal operator. Two graphical examples of that are given in Figure 5.4 and Figure 5.5 for respectively **X** and **U**.

Before starting the proof we observe the following facts:

1. For all $i : i < |\sigma| - 1 \Leftrightarrow \sigma^{ST}, map_i \models_{t^+} run$.
2. For all $i : i < |\sigma| \Leftrightarrow \sigma^{ST}, map_i \models_{t^+} state$.
3. $\forall i < |\sigma|, \forall map_{i-1} < j \leq map_i : \sigma^{ST}, map_i \models_{t^-} \psi \Leftrightarrow \sigma^{ST}, j \models_{t^-} state \mathbf{R}(\neg state \vee \psi)$.
4. $\forall i < |\sigma|, \forall map_{i-1} < j \leq map_i : \sigma^{ST}, map_i \models_{t^+} \psi \Leftrightarrow \sigma^{ST}, j \models_{t^+} \neg state \mathbf{U}(state \wedge \psi)$.
5. $\sigma^{ST}, map_{|\sigma|-1} + 1 \models_{t^{sgn}} \mathbf{Y} end$.

Base cases:

- $Pred^I$: $\sigma, i \models_{t^-} Pred^I \Leftrightarrow i \geq |\sigma| - 1 \vee Pred^I \overset{Pr}{\Leftrightarrow} i \geq |\sigma| - 1$ or $\sigma^{ST}, map_i \models_{t^-} Pred^I \overset{1}{\Leftrightarrow} \sigma^{ST}, map_i \models_{t^-} \neg run \vee Pred^I$. The strong semantics case is identical.
- $Pred^O, x, c$: Trivial.

Inductive cases:

- $\neg$: Trivial in both cases. It follows the semantics definition

- $\vee$, $Pred^I$, $Pred^O$, $func$: Trivial.

- $\mathbf{X}\varphi : \sigma^{ST}, map_i \models_{t^-} \mathcal{R}(\mathbf{X}\varphi) \Leftrightarrow \sigma^{ST}, map_i + 1 \models_{t^-} state\, \mathbf{R}\, (\neg state \vee \mathcal{R}(\varphi)) \overset{3}{\Leftrightarrow} \sigma^{ST}, map_{i+1} \models_{t^-} \mathcal{R}(\varphi)$ if $i < |\sigma| - 1$ (Holds by induction). Otherwise, $i = |\sigma| - 1$ and thus $\sigma^{ST}, map_i{+}1 \models_{t^-} \mathbf{G}\neg run$ which implies $\sigma^{ST}, map_i \models \mathcal{R}(\mathbf{X}\varphi)$ as expected by the weak semantics. We skip the strong case because it is similar.

- $\varphi_1\ \mathbf{U}\ \varphi_2 : \sigma^{ST}, map_i \models_{t^-} \mathcal{R}^-(\varphi_1\ \mathbf{U}\ \varphi_2) \Leftrightarrow \sigma^{ST}, map_i \models_{t^-} (state \vee \mathcal{R}^-(\varphi_1))\ \mathbf{U}\ (\neg state \wedge \mathcal{R}^-(\varphi_2) \vee \mathbf{Y} end) \Leftrightarrow \exists k' \geq map_i$ s.t. ($\sigma^{ST}, k' \models_{t^-} \mathcal{R}^-(\varphi_2)$ and $\sigma^{ST}, k' \models_{t^-} state$ or $\sigma^{ST}, k' \models_{t^-} \mathbf{Y} end$) and $\forall map_i \leq j' < k' : \sigma^{ST}, j' \models_{t^-} \mathcal{R}(\varphi_1)$ or $\sigma^{ST}, j' \models_{t^-} state \overset{2,5}{\Leftrightarrow} \exists k \geq i$ s.t. $\sigma^{ST}, map_k \models_{t^-} \mathcal{R}(\varphi_2)$ and $k < |\sigma|$ or $k = |\sigma|$ and $\forall i \leq j < k \sigma^{ST}, map_j \models_{t^-} \mathcal{R}(\varphi_1) \overset{Ind.(k,j<|\sigma|),\forall \phi:\sigma,|\sigma| \models_{t^-} \phi}{\Leftrightarrow} \exists k \geq i$ s.t. $\sigma, k \models_{t^-} \varphi_2$ and $\forall i \leq j < k : \sigma, j \models_{t^-} \varphi_1 \Leftrightarrow \sigma, i \models_{t^-} \varphi_1\ \mathbf{U}\ \varphi_2$. The proof of the strong case is the same.

- $\mathbf{Y}\varphi$: The case is specular to the case of $\mathbf{X}$. In this specific case, we can use run instead of $state$ because we are assuming that $i < |\sigma|$; therefore $i - 1 < |\sigma| - 1 \overset{1}{\Rightarrow} \sigma^{ST}, map_{i-1} \models_{t^-} run$ (if $i = 0$, then property is false).

- $\varphi_1 \mathbf{S} \varphi_2$: The case is specular to $\mathbf{U}$. It is sufficient to observe that 4 can be applied for the past as well.

- (v'): $\sigma^{ST}(map_i)(\mathcal{R}^{sgn}(v') = \sigma^{ST}(map_i)(\mathcal{R}^-(v)@\tilde{\mathbf{F}}(state)) : \sigma^{ST}(map_i)(\mathcal{R}^{sgn}(v')) = \sigma^{ST}(j)(\mathcal{R}^-(u))$ if $\exists j > map_i$ s.t. $\sigma^{ST}, j \models_{t^+} state$; $\sigma^{ST}(map_i)(\dots) = def_{u@\tilde{\mathbf{F}}\varphi}$ otherwise. $\sigma^{ST}(map_i)(\dots) \overset{2}{=} \sigma^{ST}(map_j)(\mathcal{R}^-(u))$ if $\exists j \geq i$ s.t. $\sigma^{ST}, map_j \models_{t^+} \mathcal{R}^+(\varphi)$; $\sigma^{ST}(map_i)(\dots) = def_{\dots}$ otherwise. Finally, by induction hypothesis, we obtain $\sigma(i)(v@\tilde{\mathbf{F}}\top)$ which has the same semantics of v'.

□

Lemma 5.1 states that each point in the sequence $map_0, \dots$ of σ^{ST} satisfies $\mathcal{R}(\varphi)$ iff the local property satisfies in that point φ; therefore, providing a mapping between the two properties.

Definition 5.7 *We define* $\mathcal{R}^*$ *as* $\mathcal{R}^*(\varphi) \doteq state\, \mathbf{R}\, (\neg state \vee \mathcal{R}(\varphi))$.

Lemma 5.2 *For all* $\sigma^{ST} \in Pr^{-1}(\sigma) : \sigma^{ST}, map_0 \models_t \mathcal{R}(\varphi) \Leftrightarrow \sigma^{ST}, 0 \models_t \mathcal{R}^*(\varphi)$

Proof: Proofs follows simply applying fact 3 of the proof of Lemma 5.1. □

Lemma 5.1 shows that $\mathcal{R}$ guarantees that satisfiability is preserved in the active transitions of the global traces. However, map_0 is not always granted to be equal

to 0 (see definition 5.4), and thus, the rewriting must guarantee that satisfiability is preserved in the first transition as well. From lemma 5.1 and lemma 5.2, we infer the rewriting theorem (Theorem 5.2) which shows that $\mathcal{R}^*$ maps the local trace to the global trace as follows:

Theorem 5.2 *For all* $\sigma^{ST} \in Pr^{-1}(\sigma) : \sigma \models_t \varphi \Leftrightarrow \sigma^{ST} \models_t \mathcal{R}^*(\varphi)$

Proof: We prove Theorem 5.2 through Lemma 5.1 and Lemma 5.2:

By Lemma 5.1, $\forall i : \sigma, i \models_{t^{sgn}} \varphi \Leftrightarrow \sigma^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{sgn}(\varphi)$. Therefore, $\sigma \models_{t^-} \varphi \Leftrightarrow \sigma^{ST}, map_0 \models_{t^-} \mathcal{R}^-(\varphi)$. By Lemma 5.2, $\sigma^{ST}, map_0 \models_{t^{sgn}} \mathcal{R}^-(\varphi) \Leftrightarrow \sigma^{ST} \models_{t^{sgn}} \mathcal{R}^*(\varphi)$. Therefore, $\sigma \models_{t^-} \varphi \Leftrightarrow \sigma^{ST} \models_{t^-} \mathcal{R}^*(\varphi)$. □

Theorem 5.3 *Let* φ *be a formula. The size of the rewritten formula with* $\mathcal{R}^*$ *is linear w.r.t.* φ *i.e.* $|\mathcal{R}^*(\varphi)| = O(|\varphi|)$.

Proof: We prove Theorem 5.3 by first showing that there is $c_1, c_2 \in \mathbb{N}$ such that, for each φ: $|\mathcal{R}(\varphi)| \leq c_1|\varphi| + c_2$. We prove the statement fixing $c_1 \doteq 1$ and $c_2 \doteq = 2|s| + 3$ for predicates and temporal formulae and $c_1 \doteq 1$ and $c_2 = 0$ for terms. The base case trivially holds since the $\mathcal{R}^{sgn}(x) = x$ and $\mathcal{R}^{sgn}(const) = const$. We now show the other cases assuming that the theorem holds on the sub-formulae. For brevity, we prove only the weak part of the rewriting; the proof of the strong part is identical since the sizes of the generated formulae are the same.

- (*Pred*) $|\mathcal{R}^-(Pred^I(u_1, \ldots, u_n))| = |\neg run \vee Pred^I(\mathcal{R}^-(u_1), \ldots, \mathcal{R}^-(u_n))| = 3 + \sum_{1 \leq i \leq n} |u_i| \leq |Pred^I(u_1, \ldots, u_n)| + 2|s| + 3$. The proof for output predicate is almost identical.

- ($\vee, \neg$) Trivial.

- ($\mathbf{X}$) $|\mathcal{R}^-(\mathbf{X}\varphi)| = |\mathbf{X}(state\, \mathbf{R}\, (\neg state \vee \mathcal{R}^-(\varphi)))| \leq 1 + 2|state| + 3 + |\mathcal{R}^-(\varphi)| = 3 + 2|state| + |\mathbf{X}\varphi|$.

- ($\mathbf{U}$) $|\mathcal{R}^-(\varphi_1\, \mathbf{U}\, \varphi_2)| = |(\neg state \vee \mathcal{R}^-(\varphi_1))\, \mathbf{U}\, (state \wedge \mathcal{R}^-(\varphi_2))| \leq 3 + 2|state| + 1 + |\varphi_1| + |\varphi_2| = |\varphi_1\, \mathbf{U}\, \varphi_2| + 3 + 2|state|$.

- ($\mathbf{S}, \mathbf{Y}$) The proof is respectively as $\mathbf{U}$ and $\mathbf{X}$.

- (*func*) Trivial.

- (v') $|\mathcal{R}^{sgn}(v')| = |\mathcal{R}^{sgn}(v)@\tilde{\mathbf{F}}(state)| = |u| + 1 + |state| \leq |u'| + 2|state| + 3$.

Finally, $|\mathcal{R}^*(\varphi)| = |state\,\mathbf{R}\,(\neg state \vee \mathcal{R}^-(\varphi))| = |\mathcal{R}^-(\varphi)| + 2|state| + 3$; since $|\mathcal{R}^-(\varphi)|$ is linear w.r.t $|\varphi|$, we deduce that $|\mathcal{R}(\varphi)|$ is linear as well.

□

Example 5.1 *Consider the formula $\varphi_{c2} \doteq \boldsymbol{G}(rec_2 \rightarrow out_2' = in_2 \wedge \boldsymbol{X}send)$ from Figure 5.1. The rewriting $\mathcal{R}^*(\varphi_{c2})$ is defined as the following formula.*

$$\boldsymbol{G}(\neg state \vee (\underbrace{run \wedge rec_2}_{\mathcal{R}^+(rec_2)} \rightarrow \overbrace{(out_2@\tilde{\boldsymbol{F}}state = in_2)}^{\mathcal{R}^-(out_2')} \wedge \underbrace{\boldsymbol{X}(run\,\mathbf{R}\,(\neg run \vee send))}_{\mathcal{R}^-(\boldsymbol{X}send)}))$$

Finally, $\mathcal{R}^*(\varphi_{c2})$ is defined as $state\,\mathbf{R}\,(\neg state \vee \mathcal{R}^-(\varphi_{c2})$.

Recall that $\boldsymbol{G}$ is an abbreviation of Until: $\boldsymbol{G}\psi \doteq \neg(\top \mathbf{U} \neg\psi)$. Therefore, the rewriting of $\boldsymbol{G}$ is equal to $\neg\mathcal{R}^+(\top\,\mathbf{U}\,\neg\psi) \doteq \neg((\neg state \vee \top)\,\mathbf{U}\,(state \wedge \mathcal{R}^+(\neg\psi)))$; we can simplify the left side of $\mathbf{U}$ obtaining $\neg(\top\,\mathbf{U}\,(state \wedge \mathcal{R}^+(\neg\psi)))$; and, with further simplifications we obtain $\neg\boldsymbol{F}(state \wedge \neg\mathcal{R}^-(\psi)) \equiv \boldsymbol{G}(\neg state \vee \mathcal{R}^-(\psi))$. Intuitively, with the "always" modality, we are interested in evaluating the states that are local by evaluating as true any state in which the component stutters.

For what regards rec_2, since it is on the left side of an implication, we rewrite it with strong semantics by asking for run to be true. Then, out_2' is rewritten via at-next operator; the intuition is that at-next provides the "next" value of the variable out_2 if state will be true, otherwise a default value is given. For what regards next, the intuition is given by Figure 5.4.

5.2.2 Optimized LTL compositional rewriting

The main weakness of the rewriting proposed in previous sections is the size of the resulting formula. However, there are several cases in which it is possible to apply a simpler rewriting. For instance, $\mathbf{G}v_o$ is rewritten by $\mathcal{R}^{sgn}$ into $\mathbf{G}(\neg state \vee v_o)$ while by ψ_{cond} (see Definition 5.3) it does not need to be rewritten since output variables do not change when *run* is false. Similarly, $\mathbf{X}v_o$ can be rewritten in the weak semantics to $end \vee \mathbf{X}v_o$.

To do so, we apply the concept of *stutter-tolerance* introduced in [59] tailored for possibly finite traces. Informally, a formula is said *stutter-tolerant* if it keeps the same value when rewritten with $\mathcal{R}^{sgn}$ in all adjacent stuttering transitions.

Definition 5.8 *An LTL formula φ and a term u are respectively said stutter-tolerant w.r.t. $\mathcal{R}^{sgn}$ iff:*
For all σ, for all $\sigma^{ST} \in Pr^{-1}(\sigma)$, for all $0 \leq i < |\sigma|$: for all $map_{i-1} < j < map_i$:

$$\sigma^{ST}, j \models_{t^{sgn}} \mathcal{R}^{sgn}(\varphi) \Leftrightarrow \sigma^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{sgn}(\varphi) \text{ and}$$
$$\sigma^{ST}(j)(\mathcal{R}^{sgn}(u)) = \sigma^{ST}(map_i)(\mathcal{R}^{sgn}(u))$$

Definition 5.9 *An LTL formula φ_{st} is syntactically stutter-tolerant iff it has the following grammar:*

$$\varphi_{st} \doteq \varphi_{st} \vee \varphi_{st} \mid \neg\varphi_{st} \mid Pred^{O}(u, \ldots, u) \mid \varphi \,\mathbf{U}\, \varphi \mid \boldsymbol{Y}\varphi$$

where φ is an LTL formula, u is a term from LTL syntax, s is an output variable and c is a constant.

Lemma 5.3 *Syntactically stutter-tolerant formulae are stutter-tolerant w.r.t. $\mathcal{R}^{sgn}$*

Proof: We prove the Lemma by induction on the size of the formula.

The base case is trivial since output variables remain unchanged during stuttering. The inductive case is proved as follows:

- $\vee, Pred^{O}$ and $\neg$: Trivial.
- **U**: We can prove the correctness by induction on j, with base case $j = map_i - 1$. $\sigma^{ST}, j \models_{t^-} (\neg state \vee \mathcal{R}^-(\varphi_1))\,\mathbf{U}\,(state \wedge \mathcal{R}^-(\varphi_2) \vee end) \Leftrightarrow \sigma^{ST}, j \models_{t^-} (\mathcal{R}^-(\varphi_2) \wedge state \vee end) \vee (\neg state \vee \mathcal{R}^-(\varphi_1)) \wedge \mathbf{X}((\neg state \vee \mathcal{R}^-(\varphi_1)\mathbf{U}(state \wedge \mathcal{R}^-(\varphi_2))) \overset{\sigma^{ST}, j \not\models_t state}{\Leftrightarrow}$ $\sigma^{ST}, j \models_{t^-} \mathbf{X}((\mathcal{R}^-(\varphi_1) \vee \neg state)\,\mathbf{U}\,(state \wedge \mathcal{R}^-(\varphi_2) \vee end) \overset{j < |\sigma^{ST}| - 1}{\Leftrightarrow} \sigma^{ST}, j+1 \models_{t^-} \mathcal{R}^-(\varphi_1\,\mathbf{U}\,\varphi_2)$ which is map_i for the base case. The inductive case follows trivially.
- **Y**: The case of **Y** can be proved in the same way of **U** by expanding **S** instead of **U**.

□

From Lemma 5.3, we derive a syntactical way to determine identify a relevant fragment of stutter tolerant formulae. Since this definition is purely syntactical, it is very simple for an algorithm to determine whether a formula is syntactically stutter tolerant; it is sufficient to traverse the structure of the formula and look at the variables and operators.

From the notion of syntactically stutter tolerant formula, we provide an optimized rewriting that is semantically equivalent to $\mathcal{R}^{sgn}$. If the sub-formulae of φ are syntactically stutter tolerant, we can simplify the rewriting for φ.

Definition 5.10 *We define $\mathcal{R}^{\theta}$ as follows. We omit the cases that are identical to $\mathcal{R}$.*

$\mathcal{R}^{\theta^-}(\boldsymbol{X}\varphi) \doteq end \vee \boldsymbol{X}\mathcal{R}^{\theta^-}(\varphi)$ *If φ is synt. st.tol.*
$\mathcal{R}^{\theta^+}(\boldsymbol{X}\varphi) \doteq \neg end \wedge \boldsymbol{X}\mathcal{R}^{\theta^+}(\varphi)$ *If φ is synt. st.tol.*
$\mathcal{R}^{\theta^-}(\varphi_1 \,\mathbf{U}\, \varphi_2) \doteq \mathcal{R}^{\theta^-}(\varphi_1) \,\mathbf{U}\, (\boldsymbol{Y}end \vee \mathcal{R}^{\theta^-}(\varphi_2))$ *If φ_1 and φ_2 are synt. st.tol.*
$\mathcal{R}^{\theta^+}(\varphi_1 \,\mathbf{U}\, \varphi_2) \doteq \mathcal{R}^{\theta^+}(\varphi_1) \,\mathbf{U}\, (\neg \boldsymbol{Y}end \wedge \mathcal{R}^{\theta^+}(\varphi_2))$ *If φ_1 and φ_2 are synt. st.tol.*
$\mathcal{R}^{\theta^{sgn}}(v') \doteq \mathcal{R}^{\theta^{sgn}}(v)@\tilde{\boldsymbol{F}}(\neg end)$ *If v is syntactically stutter tolerant*

where $\mathcal{R}^{\theta}(\varphi) \doteq \mathcal{R}^{\theta^-}(\varphi)$ and "synt. st.tol." is the abbreviation of syntactically stutter-tolerant.

Lemma 5.4 *For all σ, for all $\sigma^{ST} \in Pr^{-1}(\sigma)$, for all $i < |\sigma|$:*

$$\sigma, i \models_{t^{sgn}} \varphi \Leftrightarrow \sigma^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{\theta}(\varphi) \qquad \sigma(i)(u) = \sigma^{ST}(map_i)(\mathcal{R}^{\theta}(u))$$

Proof: From Lemma 5.1, we deduce that to prove the Lemma it suffices to prove that for each $0 \leq i < |\sigma|$, $\sigma^{ST}, map_i \models_{t^{sgn}} \mathcal{R}(\varphi) \Leftrightarrow \sigma^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{\theta}(\varphi)$ and $\sigma^{ST}(map_i)(\mathcal{R}^{sgn}(u)) = \sigma^{ST}(map_i)(\mathcal{R}^{\theta^{sgn}}(u))$. To prove that, we also prove inductively that if φ is syntactically stutter-tolerant, then for each $0 \leq i < |\sigma|$, for all $map_{i-1} < j < map_i\sigma^{ST}, j \models_{t^{sgn}} \mathcal{R}^{\theta^{sgn}}(\varphi) \Leftrightarrow \sigma^{ST}, map_i\mathcal{R}^{sgn}(\varphi)$.

We need to prove only the cases in which the sub-formulae are stutter tolerant w.r.t $\mathcal{R}^{*}$.

- **X**: If *end* is true, then $i \geq |\sigma| - 1$. Therefore, with weak semantics, the formula shall be true while with strong semantics the formula shall be false.

  If *end* is false, then $\sigma^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{\theta^{sgn}}(\mathbf{X}\varphi) \Leftrightarrow \sigma^{ST}, map_i \models_{t^{sgn}} \mathbf{X}\mathcal{R}^{\theta^{sgn}}(\varphi) \Leftrightarrow \sigma^{ST}, map_i + 1 \models_{t^{sgn}} \mathcal{R}^{\theta^{sgn}}(\varphi)$. By induction hypothesis $\cdots \Leftrightarrow \sigma^{ST}, map_i + 1 \models_{t^{sgn}} \mathcal{R}(\varphi) \Leftrightarrow \sigma^{ST}, map_{i+1} \models_{t^{sgn}} \mathcal{R}(\varphi)$.

- **U**: $\sigma^{ST}, map_i \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_1) \,\mathbf{U}\, (\mathbf{Y}end \vee \mathcal{R}^{\theta^-}(\varphi_2)) \Leftrightarrow \exists k' \geq map_i$ s.t. $\sigma^{ST}, k' \models_{t^-} \mathbf{Y}end$ or $\sigma^{ST}, k' \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_2)$ and for each $map_i \leq j' < k'\sigma^{ST}, j' \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_1)$. By induction hypothesis there exists $map_k \geq map_i$ s.t. $\sigma^{ST}, map_k \models_{t^-} \mathcal{R}(\varphi_2)$ and for all $map_i \leq map_j < map_k\sigma^{ST}, map_j \models_{t^-} \mathcal{R}(\varphi_1)$.

We now prove the additional part of the Lemma. If φ_1 or φ_2 are not syntactically stutter tolerant, then the rewriting is identical and thus, the lemma holds since φ is stutter tolerant.

If φ_1 and φ_2 are syntactic stutter tolerant, $\sigma^{ST}, j \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_1)\ \mathbf{U}\ (\mathbf{Y}end \vee \mathcal{R}^{\theta^-}(\varphi_2)) \Leftrightarrow \exists k' \geq map_i$ s.t. $\sigma^{ST}, k' \models_{t^-} \mathbf{Y}end$ or $\sigma^{ST}, k' \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_2)$ and for each $map_i \leq j' < k' \sigma^{ST}, j' \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_1)$. We observe that $\sigma^{ST}, k' \models_{t^-} \mathbf{Y}end \Leftrightarrow k' > map_{|\sigma|-1}$. Moreover, by induction hypothesis, each j' and each k' can be replaced with respectively map_j and map_k s.t. $i \leq k \leq |\sigma|$ and $i \leq j < k$. Therefore, $\cdots \Leftrightarrow \exists k \geq i$ s.t. $\sigma^{ST}, map_k \models_{t^-} \mathcal{R}(\varphi_2)$ or $k = |\sigma|$ and for all $i \leq j < k : \sigma^{ST}, map_j \models_{t^-} \mathcal{R}(\varphi_1) \Leftrightarrow \sigma, i \models_{t^-} \varphi_1\ \mathbf{U}\ \varphi_2$.

- v': It follows from induction hypothesis and by Lemma 5.1

□

Definition 5.11 *We define $\mathcal{R}^{\theta^*}$ as follows:*

$$\mathcal{R}^{\theta^*}(\varphi) \doteq \begin{cases} \mathcal{R}^{\theta^-}(\varphi) & \textit{If } \varphi \textit{ is syntactically stutter-tolerant} \\ state\ \mathbf{R}\ (\neg state \vee \mathcal{R}^{\theta^-}(\varphi)) & \textit{Otherwise} \end{cases}$$

Lemma 5.5 *For all $\sigma^{ST} \in Pr^{-1}(\sigma) : \sigma^{ST}, map_0 \models_t \mathcal{R}^{\theta}(\varphi) \Leftrightarrow \sigma^{ST}, 0 \models_t \mathcal{R}^{\theta^*}(\varphi)$*

Proof: If φ is not syntactically stutter tolerant, then the proof is the same one of lemma 5.2. If φ is syntactically stutter tolerant, then for each $map_{-1} < j < map_i, \sigma^{ST}, j \models_{t^-} \mathcal{R}^{\theta^-}(\varphi) \Leftrightarrow \sigma^{ST}, map_0 \models_{t^-} \mathcal{R}^{\theta^-}(\varphi)$. Since map_{-1} is -1, then either $map_0 = 0$ or j gets the value 0 in the for all; thus, proving the lemma. □

Theorem 5.4 *For all $\sigma^{ST} \in Pr^{-1}(\sigma) : \sigma \models_t \varphi \Leftrightarrow \sigma^{ST} \models_t \mathcal{R}^{\theta^*}(\varphi)$*

Proof: The proof is identical to the proof of Theorem 5.2 using Lemma 5.4 and Lemma 5.5. □

Theorem 5.2 and Theorem 5.4 show that respectively $\mathcal{R}^*$ and $\mathcal{R}^{\theta^*}$ are able to translate a local LTL property into a global property without changing its semantics in terms of traces. Therefore, we can use the two rewritings to prove Inference 5.1.

Definition 5.12 *Let $M_1, \ldots, M_n$ be n ITS and $\varphi_1, \ldots, \varphi_n$ be LTL formulae on the language of each M_i and let $\psi_{cond}(M_1, \ldots, M_n)$ be as in Definition 5.3. We define γ_P as follows*

$$\gamma_P(\varphi_1, \ldots, \varphi_n) \doteq \mathcal{R}^{\theta*}_{M_1}(\varphi_1) \wedge \cdots \wedge \mathcal{R}^{\theta*}_{M_n}(\varphi_n) \wedge \psi_{cond}(M_1, \ldots, M_n)$$

Corollary 5.1 *Using γ_P from Definition 5.12, γ_S from Section 5.1.1, for all compatible ITS $M_1, \ldots, M_n$, for all local properties $\varphi_1, \ldots, \varphi_n$ over the language of respectively $M_1, \ldots, M_n$, for all global properties φ: Inference 5.1 holds.*

Example 5.2 *Consider the formula $\varphi_{c2} \doteq \boldsymbol{G}(rec_2 \rightarrow out_2' = in_2 \wedge \boldsymbol{X}send)$ from Figure 5.1. The rewriting $\mathcal{R}^{\theta}(\varphi_{c2})$ is defined as the following formula.*

$$\boldsymbol{G}(\neg state \vee (\underbrace{run \wedge rec_2}_{\mathcal{R}^{\theta+}(rec_2)} \rightarrow \underbrace{(end \vee \overbrace{out_2 @ \tilde{\boldsymbol{F}} \neg end}^{\mathcal{R}^{\theta}(out_2 @ \tilde{\boldsymbol{F}} \top)} = in_2)}_{\mathcal{R}^{\theta}(out_2' = in_2)} \wedge \underbrace{end \vee \boldsymbol{X}send}_{\mathcal{R}^{\theta-}(\boldsymbol{X}send)}))$$

Finally, $\mathcal{R}^{\theta*}(\varphi_{c2})$ is defined as $\mathcal{R}^{\theta}(\varphi_{c2})$.

The simplification optimizes the formula in various parts. $\boldsymbol{X}send_2$ is rewritten into a simpler formula, in which we only need to check whether we are at the end of the local trace; the same occurs for primed output variable (next). On the contrary, $\boldsymbol{G}$ must be rewritten as for $\mathcal{R}$ because its sub-formula is not stutter-tolerant. On the other hand, the top-level rewriting is simplified because $\boldsymbol{G}$ is syntactically stutter tolerant.

5.2.3 Rewriting under fairness assumption

This section defines a variation of the previous rewriting that assumes infinite execution of local components. The rewriting is presented as a variation of the optimized rewriting; it is meant to exploit the fairness assumption to be more concise and efficient. The general idea is that since the local components run infinitely often, we can consider the semantics of LTL with event-freezing functions instead of the finite semantics considered up to now.

Definition 5.13 *We define* $\mathcal{R}^{\mathcal{F}}$ *as follows:*

$$\mathcal{R}^{\mathcal{F}}(v) \doteq v \text{ for } v \in V$$
$$\mathcal{R}^{\mathcal{F}}(Pred(u_1, \ldots, u_n)) \doteq Pred(\mathcal{R}^{\mathcal{F}}(u_1), \ldots, \mathcal{R}^{\mathcal{F}}(u_n))$$
$$\mathcal{R}^{\mathcal{F}}(\varphi \vee \psi) \doteq \mathcal{R}^{\mathcal{F}}(\varphi) \vee \mathcal{R}^{\mathcal{F}}(\psi)$$
$$\mathcal{R}^{\mathcal{F}}(\neg\varphi) = \neg\mathcal{R}^{\mathcal{F}}(\varphi)$$
$$\mathcal{R}^{\mathcal{F}}(\boldsymbol{X}\psi) \doteq \begin{cases} \boldsymbol{X}(\mathcal{R}^{\mathcal{F}}(\psi)) & \text{if } \psi \text{ is syntactically stutter-tolerant} \\ \boldsymbol{X}(run\,\mathbf{R}\,(\neg run \vee \mathcal{R}^{\mathcal{F}}(\psi))) & \text{otherwise} \end{cases}$$
$$\mathcal{R}^{\mathcal{F}}(\varphi_1\,\mathbf{U}\,\varphi_2) \doteq \begin{cases} \mathcal{R}^{\mathcal{F}}(\varphi_1)\,\mathbf{U}\,\mathcal{R}^{\mathcal{F}}(\varphi_2) & \text{if } \varphi_1 \text{ and } \varphi_2 \text{ are synt. st. tol.} \\ (\neg run \vee \mathcal{R}^{\mathcal{F}}(\varphi_1))\,\mathbf{U}\,(run \wedge \mathcal{R}^{\mathcal{F}}(\varphi_2)) & \text{otherwise} \end{cases}$$
$$\mathcal{R}^{\mathcal{F}}(\boldsymbol{Y}\varphi) \doteq \boldsymbol{Y}(\neg run\,\mathbf{S}\,(run \wedge \mathcal{R}^{\mathcal{F}}(\varphi)))$$
$$\mathcal{R}^{\mathcal{F}}(\varphi_1\,\mathbf{S}\,\varphi_2) \doteq \begin{cases} \mathcal{R}^{\mathcal{F}}(\varphi_1)\,\mathbf{S}\,\mathcal{R}^{\mathcal{F}}(\varphi_2) & \text{if } \varphi_1 \text{ and } \varphi_2 \text{ are synt. st. tol.} \\ (\neg run \vee \mathcal{R}^{\mathcal{F}}(\varphi_1))\,\mathbf{S}\,(run \wedge \mathcal{R}^{\mathcal{F}}(\varphi_2)) & \text{otherwise} \end{cases}$$
$$\mathcal{R}^{\mathcal{F}}(v') \doteq \begin{cases} v' & \text{if } \psi \text{ is syntactically stutter-tolerant} \\ \mathcal{R}^{\mathcal{F}}(v)@\tilde{\boldsymbol{F}}(run) & \text{otherwise} \end{cases}$$

Moreover, we define $\mathcal{R}^{\mathcal{F}^*}$ *as*

$$\mathcal{R}^{\mathcal{F}^*}(\varphi) \doteq \begin{cases} \mathcal{R}^{\mathcal{F}}(\varphi) & \text{If } \varphi \text{ is syntactically stutter-tolerant} \\ run\,\mathbf{R}\,(\neg run \vee \mathcal{R}^{\mathcal{F}}(\varphi)) & \text{Otherwise} \end{cases}$$

$\mathcal{R}^{\mathcal{F}^*}$ simplifies $\mathcal{R}^{\theta^*}$ in various ways. It does not need to distinguish between input and output variables, it does not distinguish between weak/strong semantics, and it does not distinguish between *state* and *run*. Intuitively, these distinctions make sense only with finite semantics; since the rewriting assumes infinite local executions, these technicalities become superfluous.

Theorem 5.5 *Let* σ *be an infinite trace, for all* $\sigma^{ST} \in Pr^{-1}(\sigma) : \sigma \models \varphi \Leftrightarrow \sigma^{ST} \models \mathcal{R}^{\mathcal{F}^*}(\varphi)$

Proof: (Sketch) Since σ is infinite, *end* is always false, *state* $\Leftrightarrow$ *run*. We can substitute *end* with $\bot$ and *state* with *run* in $\mathcal{R}^{\theta}$ obtaining this rewriting. □

Example 5.3 *Consider the formula* $\varphi_{c2} \doteq \boldsymbol{G}(rec_2 \rightarrow out'_2 = in_2 \wedge \boldsymbol{X} send_2)$ *from Figure 5.1. The rewriting* $\mathcal{R}^{\mathcal{F}}(\varphi_{c2})$ *is defined as the following formula.*

$$\boldsymbol{G}(\neg run \vee (\underbrace{rec_2}_{\mathcal{R}^{\mathcal{F}}(rec_2)} \rightarrow \underbrace{(\overbrace{out'_2}^{\mathcal{R}^{\mathcal{F}}(out'_2)} = in_2)}_{\mathcal{R}^{\mathcal{F}}(out'_2 = in_2)} \wedge \underbrace{\boldsymbol{X} send_2}_{\mathcal{R}^{\mathcal{F}}(\boldsymbol{X} send_2)}))$$

Finally, $\mathcal{R}^{\mathcal{F}^*}(\varphi_{c2})$ *is defined as* $\mathcal{R}^{\mathcal{F}}(\varphi_{c2})$.

By assuming infinite executions of local traces it is possible to drastically simplify the rewriting. Next outputs are left unchanged because they are stutter tolerant and with infinite execution end is not needed. Input variables are also transparent to the rewriting because the infinite semantics do not distinguish between input and output variables. As before, $\boldsymbol{G}$ *must be rewritten because its sub-formula is not stutter-tolerant; however, in this case, the rewriting can use run instead of state because the two expressions are equivalent with infinite semantics. Finally, as before, the top-level rewriting is simplified because* $\boldsymbol{G}$ *is syntactically stutter tolerant.*

5.3 Experimental Evaluation

We implemented the compositional techniques proposed in this chapter inside the contract-based design tool OCRA[92]. Our extension covers the contract refinement check, in which a contract (defined as a couple $\langle A, G \rangle$ of LTL formulae) is considered correct if its sub-component contracts refine it. For simplicity, we consider contracts without assumptions i.e. in which the assumption is $\top$; in this scenario, the refinement of contracts is given by the compositional reasoning described in this chapter. For a detailed description of the assume-guarantee contract proof system employed in the tool refer to [113].

The objective of this experimental evaluation is to do both a quantitative and a qualitative evaluation of our compositional approach differentiating possibly finite and infinite semantics. Qualitatively, we compare the verification results to assess how the finiteness impacts on the result, observing also the required scheduling assumptions for possibly finite systems. Quantitatively, we analyze the overhead of reasoning over a mix of finite and infinite executions to analyse whether our rewriting scales on real models. Due to the absence of equally expressible formalism to define asynchronous composition, our comparison with related work is limited to another rewriting suited only for local infinite systems with event-based asynchronous composition[29].

Model	Algorithm	Result	Time (s)
Response boolean sequence with size=3	TrR	Invalid	0.74
Response boolean sequence with size=3	TrRuFA	Valid	0.79
Response boolean sequence with size=3	TrR+F	Valid	0.48
Simplified Example (no component c_3)	TrR	Invalid	1.15
Simplified Example (no component c_3)	TrRuFA	Valid	0.87
Simplified Example (no component c_3)	TrR+F	Valid	2.82
Example (Figure 5.1)	TrR	Valid	20.13
Example (Figure 5.1)	TrR+F	Valid	5.3
Example (Figure 5.1)	TrRuFA	Valid	1.37
Response event sequence with size=12	TrR	Invalid	14.61
Response event sequence with size=12	TrRuFA	Valid	0.8
Response event sequence with size=12	TrR+F	Valid	3.74
Universality sequence with size=17	TrR	Invalid	6.63
Universality sequence with size=17	TrRuFA	Valid	2.81
Universality sequence with size=17	TrR+F	Valid	11.17
"Brake when AEB status is active" bugged	TrR	Invalid	48.3
"Brake when AEB status is active" bugged	TrRuFA	Valid	24.63
"Brake when AEB status is active" bugged	TrR+F	Valid	74.72
"Brake when AEB status is active" fixed	TrR	Valid	136.34
"Brake when AEB status is active" fixed	TrRuFA	Valid	17.6
"Brake when AEB status is active" fixed	TrR+F	Valid	62.83

Table 5.1: Subset of quantitative and qualitative results of the experimental evaluation.

Although we defined compositional reasoning with truncated semantics, in our experiments we reason about global infinite executions and possibly finite local executions. The experiments[1] were run in parallel on a cluster with nodes with Intel Xeon CPU 6226R running at 2.9GHz with 32CPU, 12GB. The timeout for each run was two hours and the memory cap was set to 2GB.

In this section, we denote the composition based on $\mathcal{R}^{\theta*}$ as TrR(Truncated Rewriting); we denote the composition based on $\mathcal{R}^{\theta*}$ with additional constraints to ensure infinite local executions as TrR+F (Truncated Rewriting + Fairness); we denote the composition based on $\mathcal{R}^{\mathcal{F}*}$ assuming local infinite execution as TrRuFA (Truncated Rewriting under Fairness Assumption).

[1]The results of the experimental evaluation can be found at `https://es-static.fbk.eu/people/bombardelli/papers/nfm22/expeval_journal.tar.gz`

5.3.1 Benchmarks

We have considered benchmarks of various kinds:

1. Simple asynchronous models from OCRA comprehending the example depicted in Figure 5.1.
2. Pattern formula compositions from [59] experimental evaluation
3. Contracts from the experimental evaluation of [91] on AUTOSAR models adapted for truncated semantics.

Simple Asynchronous Models

We considered two variations of the example model depicted in Figure 5.1: one version considers all three components with both the possible outcome of failure and success. The second version is composed only of components $c1$ and $c2$ and the global property states that eventually the message is sent.

The second model we are considering is a simple representation of a Wheel Brake System with two Braking System Control Units (BSCU) connected to two input braking pedals, a Selection Switch and an actual hydraulic component that performs the actual brake. The top-level property states that if one of the two pedals is pressed eventually the hydraulic component will brake.

Pattern Models

We took from [59] some benchmarks based on Dwyer LTL patterns [147]. The considered LTL patterns are the following: *response*, *precedence chain* and *universality* patterns. The models compose the pattern formulae in two ways: as a sequence of n components linked in a bus and as a set of components that tries to write on the output port. These patterns are parametrized on the number of components involved in the composition.

EVA AUTOSAR contracts

We took the experimental evaluation from the tool EVA[91], and we adapted it for our compositional reasoning. The models are composed of various components for which the scheduling is of two types: event-based and cyclic. Event-based scheduling forces the execution of a component when some of its input variables change while cyclic scheduling forces the execution of the component every n time units.

Alg	ALL	TRUE	FALSE	UNK	<1 s	<60 s	<600 s
ALL	624	476	88	60	82	448	535
TrR	208	91	86	31	26	133	167
TrR+F	208	182	1	25	21	126	168
TrRUFA	208	203	1	4	35	189	200

Table 5.2: Summary result for algorithms

In addition to these models, we also considered a variation of a subset of instances (described in Section 5) and we relaxed the scheduling constraints allowing some components to become unresponsive at some point to the original constraints. We forced an event-based scheduled component (Brake Actuator) to be scheduled when its input changes but only until it ends; moreover, we forced a cyclic scheduled component (Brake Watchdog) to run every n times units until it ends. Finally, assuming that one of the two components runs infinitely often (i.e. does not become unresponsive), we check if the composition still holds.

5.3.2 Experimental Evaluation Results

Figure 5.6 provide an overall comparison between TrR, TrR+F and TrRuFA in terms of results and execution time. In these plots, the colour determines the validity results: blue aggregates all the results without distinction between valid and invalid; green considers valid instances of both techniques (if the other algorithm returns valid, timeout is optimistically believed to be true); yellow consider instances in which the two techniques had different results.

Overall, we checked 624 instances (208 for each rewriting); 406 of these instances were proven valid, 88 were proven invalid and 60 instances timed out. The general statistics can be found in Table 5.2

Table 5.1 shows some relevant instances of the experimental evaluation highlighting their execution time and their validity results.

Qualitative Results As we expected, there are differences in validity results between TrR and TrRuFA. The pattern models cannot guarantee the validity of the composition with the weak semantics without additional constraints over the composition. Intuitively, it suffices that a single component is not scheduled to violate bounded response properties.

Similarly, we compared the validity results of the 3 rewritings on a simplified version

of the example of Section 5; this variation of the model contains only components c_1 and c_2. The simplified version was proved valid using TrRuFA and TrR+F while a counterexample was found using TrR. The simplified version is found invalid when c_1 is scheduled only a finite amount of times failing to deliver the message to component c_2. On the other hand, when we correct the model by adding the component c_3, the composition is proved valid in all the 3 cases.

For what regards the EVA benchmarks, TrR and TrRuFA gave the same validity results. That occurred because the scheduling constraints were quite strong; in particular, the cyclic constraints implicitly forced components, such as the watchdog, to run infinitely often. The event-based scheduled component could have a finite execution but only if their input did not change from some point on. Differently, updating the constraints by removing these implicit infinite executions makes the property invalid using TrR. To fix these invalid results, we updated the system assuming that at least one of the two components runs infinitely often and we relaxed the global property increasing the timed bound for the brake to occur from 2 time units to 3. Finally, the corrected model was proved in all the 3 cases.

Quantitative Results We provided a comparison between $\mathcal{R}^\theta$ and $\mathcal{R}^\mathcal{F}$ in terms of impact of the transformation on the verification time. Since $\mathcal{R}^\mathcal{F}$ assumes that each local component is executed infinitely often, we used the rewriting $\mathcal{R}^\theta$ equipped with fairness assumption (TrR+F). Figure 5.6a shows the comparison between TrR+F and TrRuFA. In this case, TrRuFA is more efficient than TrR+F. This is not surprising because the generated formula assuming infinite local execution can be significantly smaller than the general one. In general, we see that, regardless of the result, TrRuFA can prove the property faster. We think that having to check all the possible combinations of finite/infinite local execution provides a significant overhead in the verification.

In Figure 5.7, we compared TrRuFA with another rewriting for the composition of temporal properties from [29]. The comparison has been done over a subset of the *pattern* models. The rewriting of [29] is in principle similar to TrRuFA; it assumes infinite executions of local component but, contrary to our approaches, in [29] the asynchronous composition is supported only considering shared synchronization events[2]. Due to the expressive limitations of the other approach, we applied the evaluation over a smaller set of instances. It should be noted that the technique of [29] already introduces

[2]These events are basically Boolean variables shared between two components. When one of these variable becomes true, both the components run. Although it is not detailed in the thesis, we do support these events natively by adding additional scheduling constraints in α

fairness assumptions of the infinite execution of local components; therefore, we don't need to manipulate/complicate that rewriting in this evaluation. It is clear from the figure that TrRuFA outperforms this other rewriting.

5.4 Related Works

In this section, we compare the contributions proposed in this chapter with related work. For a complete overview of both compositional reasoning and LTL model checking, refer to Chapter 3.

One of the most important works on temporal logic for asynchronous systems is Temporal Logic of Action (TLA) [221] by Leslie Lamport, later extended with additional operators[222]. TLA has been also used in a component-based manner in [269]. Our formalism has similarities with TLA. We use a (quantifier-free) first-order version of LTL [239] with "next" function to specify the succession of actions of a program. TLA natively supports the notion of stuttering for composing asynchronous programs so that the composition is simply obtained by conjoining the specifications. We focus instead on local properties that are specified independently of how the program is composed so that "next" and input/output data refer only to the local execution. Another substantial difference with TLA is that our formalism also supports a finite semantics of LTL, permitting reasoning over finite traces.

As for propositional LTL, the composition of specifications is studied in various papers on assume-guarantee reasoning (see, e.g., [247, 254, 208, 112]) for both synchronous and asynchronous composition. In the case of asynchronous systems, most works focus on fragments of LTL without the next operator, where formulae are always stutter invariant. Other studies investigated how to tackle down state-space explosion for that scenario usually employing techniques such as partial order reduction [29]. However, our work covers a more general setting, where the presence of input variables makes formulae non-stutter-invariant.

Similar to our work, [29] considers a rewriting for LTL with events to map local properties into global ones with stuttering. However, contrary to our work, it does not consider input variables (nor first-order extension) and assume that every variable does not change during stuttering, resulting in a simpler rewriting. In [154], a temporal clock operator is introduced to express properties related to multiple clocks and, in principle, can be used to interpret formulae over the time points in which a component is not stuttering. That rewriting is indeed similar to the basic version defined in this chapter, but is limited to propositional LTL and has not been conceived for

asynchronous composition. The optimization that we introduce to exploit the stutter invariance of subformulae results in simpler formulae easy to be analyzed as shown in our experimental evaluation.

The rewriting of asynchronous LTL is similar to the transformation of asynchronous symbolic transition systems into synchronous ones described in [107]. That work considers connections based on events where data are exchanged only upon synchronization (allowing optimizations as in shallow synchronization [81]). Thus, it does not consider components that read from input variables that may be changed by other components. Moreover, [107] is not able to transform temporal logic local properties in global ones as in this chapter.

In [231], the authors defined a technique to verify asynchronous assume/guarantee systems in which assumptions and guarantees are defined on a safety fragment of LTL with predicates and functions. The main differences between that work and this one are the following. The logic considered by [231] is limited to a safety fragment of LTL with "globally" and past operators; on the other hand, our work covers full LTL with past operators with event freezing functions and finite semantics. Another key difference is that they consider scheduling in which a component is first dispatched and then, after some steps it completes its action; instead, our framework defines scheduling constraints with LTL formulae and actions are instantaneous.

Another related work is the one proposed in [68]. Here, the authors propose a compositional reasoning based on assume-guarantee contracts for model based safety assessment (MBSA). They extend the model with possible faults that are modelled as input Boolean parameter to the system. However, contrary to our work, asynchronous composition is not considered, and a fault represents the non-satisfaction of a local property.

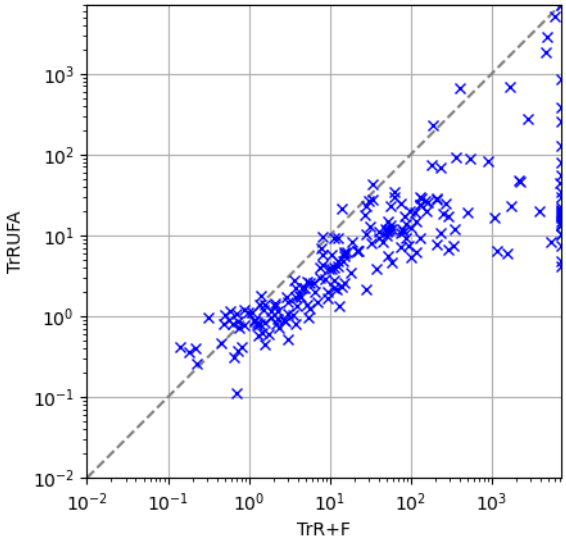
In [26], a related rewriting is proposed in the context of Asynchronous Hyperproperties. That work considers a hyper-logic based on LTL that uses a rewriting to align different traces of the same systems on "interesting" points. Although for certain aspects the work is similar, it considers a very different problem.

A related compositional formalism is BIP [23]. As our formalism it is hierarchical and asynchronous. However, there are conceptual differences between the two. Our formalism is more "liberal" allowing any kind of scheduling of local components (it is possible to explicit the scheduling using the LTL($\mathcal{T}$) specification α). In principle one could compile BIP compositional framework in our (as done for instance in [51] to STS), even though it would require non-trivial effort to do that efficiently. A related tool is D-FINDER [30] which verify safety properties compositionally by computing interaction

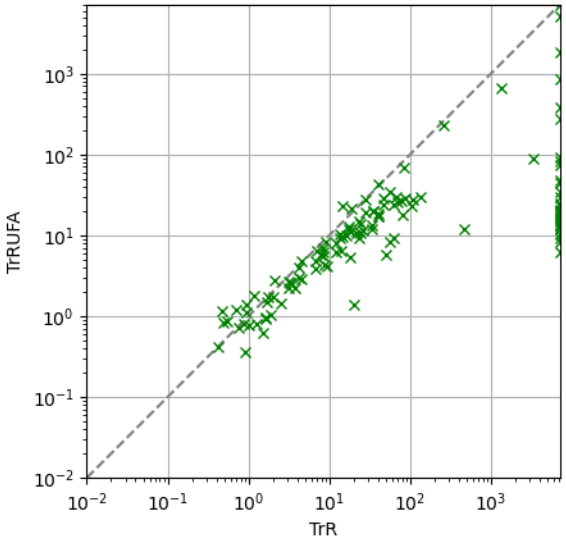
invariants. Their framework is focused on safety properties and deadlock checking. They leverage the interaction between components to generate additional invariants to entail a global property. On the other hand, we use temporal logic formula and reduce everything to LTL($\mathcal{T}$) satisfiability. Although not studied, our framework could benefit from ad-hoc approaches on the component interactions.

Finally, our logic semantics is based on the works of Eisner, Fisman et al. on truncated LTL of [153]. The main differences are the following. Our logic includes past operators, first-order predicates and functions; we consider only weak/strong semantics for our logic while their work instead also deals with neutral semantics.

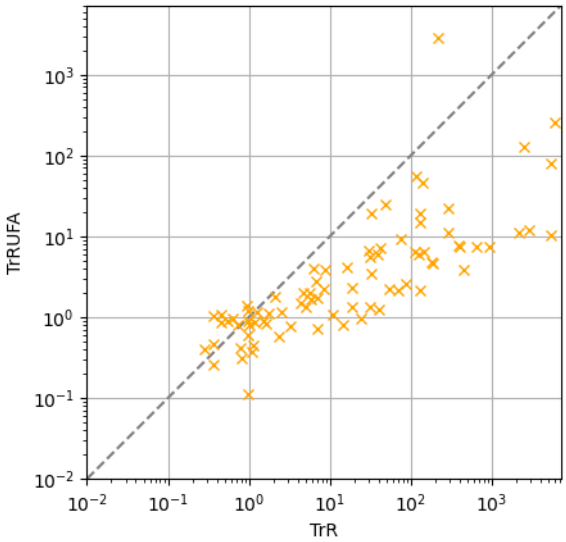
In summary, while existing works address various aspects related to our approach, none provide a unified framework for the composition of asynchronous systems with I/O data ports that explicitly accounts for the termination of local components. To the best of our knowledge, this work is the first to bridge this gap, offering a comprehensive solution that integrates these considerations into a general framework.



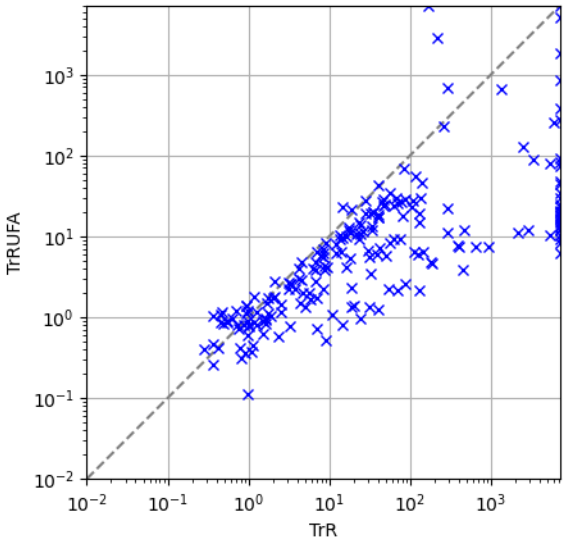
(a) Comparison between TrR+F and TrRuFA over valid instances.



(b) Comparison between TrR and TrRuFA over valid instances.



(c) Comparison between TrR and TrRuFA over "different" results.



(d) Comparison between TrR and TrRuFA over all the instances.

Figure 5.6: Scatter plots comparing TrR,TrR+F and TrRuFA

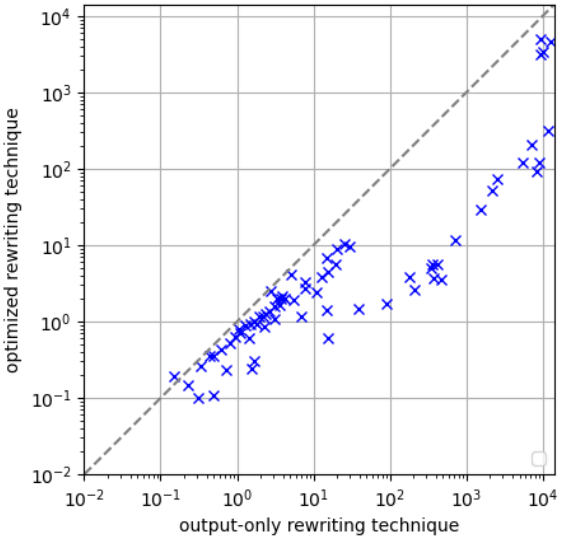


Figure 5.7: Comparison between TrRuFA and event-based rewriting of [29]

Chapter 6

Asynchronous Composition of Timed Systems

As discussed in Chapter 1, one of the central challenges in formal verification lies in scaling verification techniques to realistic systems without compromising expressiveness. In Chapter 5, we addressed this challenge for asynchronous systems by introducing a compositional reasoning framework that enables scalable verification through modular analysis. While effective, that approach was developed for a discrete-time setting, where system evolution is represented as a sequence of actions. Such a model, however, abstracts away the continuous passage of time and cannot directly capture quantitative timing constraints.

At first glance, the introduction of real-time aspects may seem orthogonal to the asynchronous nature of system behaviour because time progresses uniformly for all components regardless of their interleaving. However, this assumption no longer holds in Distributed Real-Time Systems (DRTS), where each component operates according to a local clock that may drift or be skewed relative to others and to global real time. Consequently, reasoning about timing in DRTS requires accounting for both the uncertainty introduced by asynchrony and the discrepancies between local time measurements.

DRTS consist of multiple components that communicate over a network and rely on numerous timing constraints governing data exchange and message delivery. Formal verification of such systems is particularly challenging due to the tight coupling between timing constraints, synchronization mechanisms, and communication protocols. Moreover, in decentralized architectures, clocks are typically skewed, necessitating periodic synchronization–e.g., through algorithms such as the Berkeley clock synchronization protocol–to maintain consistency among local times.

Building on the compositional verification approach of Chapter 5, this chapter extends those ideas into the timed domain, where correctness must be ensured under explicit real-time constraints across distributed, asynchronously communicating components. Our goal is twofold: (i) to enhance scalability by maintaining modular verification through compositional reasoning, and (ii) to preserve expressiveness by supporting rich timing properties characteristic of real-world DRTS.

Following our previous work on compositional reasoning for Linear-time Temporal Logic modulo theory, we extend the framework to enable verification of DRTS, reconciling asynchronous behaviours with real-time guarantees. The core contribution of this chapter lies in adapting compositional reasoning to progressively more realistic timing models: first to systems with skewed but non-resettable clocks, and subsequently to systems featuring resettable, non-monotonic clocks. Each of these models poses distinct challenges for temporal specification and reasoning, which we address through corresponding logical extensions and refined semantic definitions.

We recall the Metric Temporal Logic (MTL) introduced in Section 2.7.3, where each temporal operator is associated with a real-time interval specifying the period within which the formula must hold (e.g., $\mathbf{F}_{\leq 2} v$). In this chapter, we consider a variation of this logic in which the notion of time is determined by a local clock c, informally viewed as a variable that increases as time elapses (e.g., $\mathbf{F}^{c}_{\leq 2} v$). A formal definition of the logic and its semantics will be presented in Section 6.2; for now, it suffices to keep in mind that the temporal operators refer to the progression of time as captured by the value of the local clock c.

Compositional Reasoning Under Drifted Clocks. We first consider systems where local clocks are not perfectly synchronized and may drift relative to each other with bounded skew ϵ. This is common in DRTS where clocks advance at slightly different rates, but no explicit resets are performed. To reason compositionally in such systems, local properties must be specified over local clocks, and conservative safety margins must be introduced to ensure that timing constraints remain valid when moving from local to global reasoning. We show how to adapt the rewriting-based compositional approach of Chapter 5 to this timed setting, where timed transitions are synchronous but observed with possibly skewed clocks.

For instance, a local property like $\mathbf{G}(flt \rightarrow \mathbf{G}^{cl1}_{\leq p} \neg alv)$ may fail globally when considering clock drift. We demonstrate how such formulae must be rewritten to introduce a corrected bound $\tilde{p}$, ensuring their validity despite the bounded drift between components. This reasoning is formalized through a distributed extension of Metric Tempo-

ral Logic (MTL) with local-clock operators like $\mathbf{G}^{c}_{\leq p} b$ that refer to component-specific clocks.

Compositional Reasoning with Resets: MTLSK. We then extend our framework to handle a more complex setting: clocks that may be reset as part of synchronization protocols. Such resets introduce non-monotonic time references and pose significant semantic challenges for classical metric temporal operators. For example:

- A property like $G_{\leq p} b$ can be incorrectly violated across disconnected intervals due to resets. The Boolean variable b might hold continuously for p times unit and afterwards go to false; then, after a long time, there is a reset that significantly reduces the local time.

- A liveness property such as $\mathbf{G}(receive(msg) \rightarrow \mathbf{F}^{sk}_{\leq p} send(ACKmsg))$ might appear valid if the response happens after a clock reset, even though the real-time deadline was violated.

To address these issues, we introduce *MTLSK (Metric Temporal Logic with Resettable Skewed Clocks)*, an extension of MTL that accounts for both bounded drift (ϵ) and bounded resets (λ). MTLSK introduces refined metric operators — such as first-interval semantics — that avoid undesired reset-based satisfaction. Finally, we also lift our compositional reasoning to deal with local component with resettable skewed clock using MTLSK specifications.

Although we theoretically present a complete compositional framework, we don't introduce a complete compositional case study. We leave this type of work for the future.

Contributions To summarize, the contributions of this chapter are the following.

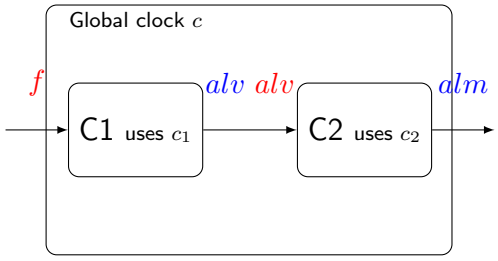
- We provide an extended version of the compositional reasoning introduced in Chapter 5 to deal with local components with possibly skewed clocks;

- We define the novel logic MTLSK to deal with local time semantics with resettable skewed clocks

- We provide an encoding to LTL($\mathcal{T}$) for the satisfiability checking of MTLSK with parametric bounds

The contributions presented in this section are an extended version of the extended abstract work on MTLSK [60] and the conference paper on reasoning of MTLSK [61]. All the technical content related to composition in timed and skewed setting is novel and has not been published yet. All the work presented in this chapter was carried out by the author in collaboration with Stefano Tonetta.

Running Example Consider for example a system of two components. The first component sends an alive signal (variable alv) to the second component unless there is a fault (variable f). The second component monitors the alive signal and, if absent, raises an alarm (variable alm). Globally, we expect that if there is a fault, an alarm is triggered in due time. The components use clocks c_1 and c_2, while the global clock is c. Figure 6.1 shows the two components and the corresponding formulas that characterize their behaviours.



Perfect Clocks ($c_1 = c_2 = c$):
$\varphi_1 = \mathbf{G}(f \rightarrow \mathbf{G}_{\leq p}\neg alv)$
$\varphi_2 = \mathbf{G}(\mathbf{G}_{\leq p}\neg alv \rightarrow \mathbf{F}_{\leq p} alm)$
$\varphi = \mathbf{G}(f \rightarrow \mathbf{F}_{\leq p} alm)$

Skewed Clocks (max drift ϵ):
$\varphi_1 = \mathbf{G}(f \rightarrow \mathbf{G}_{\leq \tilde{p}}\neg alv)$
$\varphi_2 = \mathbf{G}(\mathbf{G}_{\leq p}\neg alv \rightarrow \mathbf{F}_{\leq p} alm)$
$\varphi = \mathbf{G}(f \rightarrow \mathbf{F}_{\leq p} alm)$
with $\tilde{p} = p \cdot \left(1 + \frac{2\epsilon}{1-\epsilon}\right)$

Resettable Skewed Clocks (MTLSK):
$\varphi_1 = \mathbf{G}(f \rightarrow \overline{\mathbf{G}}_{\leq \tilde{p}}\neg alv)$
$\varphi_2 = \mathbf{G}(\overline{\mathbf{G}}^{c}_{\leq p}\neg alv \rightarrow \overline{\mathbf{F}}^{c_2}_{\leq p} alm)$
$\varphi = \mathbf{G}(f \rightarrow \overline{\mathbf{F}}_{\leq p+4\tilde{q}} alm)$
with $\tilde{q} = q(1 + 2\epsilon/(1-\epsilon))$ + clock sync. at least every q time units.

Figure 6.1: Running example: graphical system view with evolving specifications under different clock assumptions.

In this case, the local specifications – that we assumed to be satisfied by local components – would be the following MTL specifications: $\varphi_1 \doteq \mathbf{G}(f \rightarrow \mathbf{G}_{\leq p}\neg alv)$, $\varphi_2 \doteq \mathbf{G}(\mathbf{G}_{\leq p}\neg alv \rightarrow (\mathbf{F}_{\leq p} alm))$. Then, let the global specification be defined as: $\varphi \doteq \mathbf{G}(f \rightarrow \mathbf{F}_{\leq p} alm)$. The compositional reasoning holds if the local component clocks are not skewed.

If instead we consider local components where local time is skewed then the compositional step fails. In this case, our compositional approach γ_P, must consider local

clocks (in this case c_1 and c_2) to represent local time during composition.

Our γ_P will then take φ_1 and φ_2 replacing each $\mathbf{U}_{\mathcal{I}}$ with respectively $\mathbf{U}^{c_1}_{\mathcal{I}}$ and $\mathbf{U}^{c_2}_{\mathcal{I}}$ and then apply a similar rewriting as the one proposed in Section 5.2. The idea is that the new distributed temporal operators will use clock evaluation as time, this is done to represent local skewed time with local clocks. Then, assuming that clock rates are skewed of at most ϵ (meaning that the derivate over time is between $1-\epsilon$ to $1+\epsilon$), we can adjust the local specification φ_1 and the global specification φ considering the larger bound $\tilde{p} = p(1+2\epsilon/(1-\epsilon))$ as: $\varphi_1 \doteq \mathbf{G}(f \rightarrow \mathbf{G}_{\leq \tilde{p}} \neg alv)$ and $\varphi \doteq \mathbf{G}(f \rightarrow \mathbf{F}_{\leq p} alm)$.

Let us now consider resets. Assuming that resets might occur, we need to consider MTL with non-monotonic time even locally. In the following, we use MTLSK (that we will detail in Section 6.2) to describe local specification. To give an intuitive view of the logics, it extends distributed MTL with an additional distributed operator $\overline{\mathbf{U}}^{c}_{\mathcal{I}}$ that, instead of considering an unbounded amount of intervals in which the eventuality part (φ_2 in $\varphi_1 \overline{\mathbf{U}}^{c}_{\mathcal{I}} \varphi_2$), it considers only the first interval. A clear view of such operator is given by Figure **??**.

Then, assuming that the clocks are reset with period q and that q is much smaller than p, we can reduce the safety margins. In particular, considering the following compositional clock-synchronization constraint

$$\alpha^{sync} \doteq \mathbf{G} \left(\begin{array}{cc} (Reset(c_1) \rightarrow c_1' = c) & \wedge \\ (Reset(c_2) \rightarrow c_2' = c) & \wedge \\ \neg reset(c) & \end{array} \right) \wedge \mathbf{GF}^{c}_{\leq q} Reset(c_1) \wedge \mathbf{GF}^{c}_{\leq q} Reset(c_2)$$

We can now, redefine the correct bounds for the composition with the following specification. The new specifications are as follows: $\varphi_1 \doteq \mathbf{G}(f \rightarrow \overline{\mathbf{G}}_{\leq \tilde{p}} \neg alv)$, $\varphi_2 = \mathbf{G}(\overline{\mathbf{G}}^{c}_{\leq p} \neg alv \rightarrow \overline{\mathbf{F}}^{c_2}_{\leq p} alm)$ and $\varphi = \mathbf{G}(f \rightarrow \overline{\mathbf{F}}_{\leq p+4\tilde{q}} alm)$

First, note that we use the "first-interval" variants. In fact, we require that after a fault the alive signal of the first component is down for an interval without discontinuity; similarly, when the second component sends an alarm it cannot rely on a future reset and must send it within the first interval in which the local clock is below the given bound.

Outline The rest of the chapter is organized as follows. In Section 6.1, we present the compositional reasoning for timed systems assuming non-resettable skewed clocks; in Section 6.2, we present a novel logics tailored to reason with local time semantics defined with resettable skewed clocks and its composition; finally, in Section 6.3, we

compare our contributions with respect to other relevant publications.

6.1 Composing MTL with Skewed Clocks

In this section, we show how to extend the compositional approach obtained from discrete systems to timed systems with skewed clocks.

We start by showing how to lift the composition from discrete-time to timed system.

We recall the notion of super-dense timed domain from Section 2.7.1 and the notion of timed trace from Section 2.7.2.

To simplify the construction, we assume that each local component can be scheduled infinitely often. Moreover, to avoid the occurrence of unrealistic traces, we ignore Zeno executions. Therefore, as mandated by the super-dense timed domain definition, our timed traces are all time divergent. We now introduce the notion of *Skewed Clock* i.e. a clock whose rate is not aligned with the real time.

Definition 6.1 (Skewed Clock) *A "skewed clock" is a clock variable $c \in \chi$ such that, for every trace σ and time point $t = \langle i, r \rangle$*

1. *$\sigma(t)(c)$ is differentiable in I_i with $\frac{d\sigma(t)(c)}{dt} = rate(c) \in [1-\epsilon, 1+\epsilon]$.*
2. *For every t, if t is a discrete step $|\sigma(succ(t))(c)) - \nu(t)| \leq \lambda$.*

where $\nu(t)$ represents the value of time at point t, I_i is the i-th interval representing a discrete step is defined as a step in which time does not pass, ϵ and λ are two given positive real parameters.

The rate of a clock c may vary along the trace, depending on the current conditions. We say that c is perfect if its rate remains constant and equal to 1 throughout the trace, and that c is non-resettable if its value does not change during discrete transitions.

From the notion of skewed clock, it is possible to slightly modify the syntax and semantics of MTL to represent skewed time with non-resettable skewed clocks as follows.

$$\varphi \doteq p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \, \mathbf{U}^c_{\mathcal{I}} \, \varphi \mid \mathbf{X}\varphi \mid \tilde{\mathbf{X}}\varphi$$

where $c \in \chi$ is a non-resettable skewed clock.

We then define the semantics of the distributed until operator as follows

$$\begin{aligned} \sigma, t \models \varphi_1 \, \mathbf{U}^c_{\mathcal{I}} \, \varphi_2 \text{ iff} \quad & \text{there exists } t' \geq t \text{ s.t. } \sigma(t')(c) - \sigma(t)(c) \in \mathcal{I}, \sigma, t' \models \varphi_2 \text{ and} \\ & \text{for each } t \leq t'' < t'. \sigma, t'' \models \varphi_1 \end{aligned}$$

6.1.1 Composition with non-resettable skewed clocks

We now define the notion of component in a timed system lifting the notion of ITS for timed systems.

Definition 6.2 (Interface Timed Transition System) *An Interface Timed Transition System is the timed extension of ITS (see Definition 5.1) constructed as the following tuple*

$$M \doteq \langle V^I, V^O, I, T, Inv, \mathcal{SF} \rangle$$

where

- V^I, I, T *and* $\mathcal{SF}$ *are defined as for ITS,*
- V^O *also contains skewed clocks i.e.* $\chi \subseteq V^O$,
- Inv *is a clock invariant over clock variables* χ

Moreover, given a timed trace $\sigma = \langle \mathcal{M}, \tau, \overline{\mu} \rangle$. *We say that* σ *is a trace of* M *iff*

- $\sigma, \boldsymbol{0} \models I$.
- *For each* $t \in T$ *such that* $succ(t) \in T$: $\overline{\mu}(t) \cdot \overline{\mu}(succ(t))' \models T$.
- *Clocks are evaluated in the traces as mandated by Definition 6.1 and the other variables do not change value inside time intervals.*
- *For each* $t \in T : \sigma, t \models Inv$.
- *for all* $\langle f_A, f_G \rangle \in \mathcal{SF}$, *If for all* t *there exists* $t' \geq t$, $\sigma, t' \models f_A$ *then, for all* t *there exists* $t' \geq t : \sigma, t' \models f_G$.

Definition 6.2 represents a I/O automaton with clocks interpreted over timed traces. Given the nature of DRTS systems, we want to model components with skewed clocks, where each clock follows the same rate of some reference clock. In our setting, we want each local component to have skewed clocks with the same rate, meaning that if component M_1 has χ_1 clock variables, they need to grow with the same speed.

We now introduce a function that, given a ITTS M produces a new ITTS with the same behaviours where each skewed clock is bounded by a new master clock c_M.

Definition 6.3 (Skewed Interface Timed Transition System) *Let* M *be an ITTS,* $\mathcal{I}$ *be a positive interval in real. We define* $Skew(M)$ *as*

$$M^{\mathcal{SK}} \doteq \langle V^I, (V^O \setminus \chi) \cup \overline{\chi}, I\lceil \chi/\overline{\chi} \rfloor \wedge c_M = 0, T\lceil \chi/\overline{\chi} \rfloor \wedge {c_M}' = c_M, \mathcal{SF}\lceil \chi/\overline{\chi} \rfloor, Inv\lceil \chi/\overline{\chi} \rfloor \rangle$$

where $\overline{\chi} = \{c_M\} \cup \{\overline{c} | c \in \chi\}$ *such that each* $c, c' \in \overline{\chi}$ *has the same rate i.e.* $rate(c) = rate(c')$.

Theorem 6.1 (Perfect/Skewed Clock Relation of ITTS) *Let* M *be an ITTS with only perfect clocks (i.e. clocks with* $rate(c) = 1$ *for each* $c \in \chi$*) and* φ *be a MTL formula.*

$$M \models \varphi \text{ iff } M^{\mathcal{SK}} \models \varphi^{\mathcal{SK}}$$

where $M^{\mathcal{SK}} \doteq Skew(M)$ *and* $\varphi^{\mathcal{SK}} \doteq \varphi\lceil \mathbf{U}/\mathbf{U}^c \rfloor \lceil \chi/\overline{\chi} \rfloor$

Proof: Given any timed trace σ of M, we can construct a timed trace $\sigma^{\mathcal{SK}}$ over $M^{\mathcal{SK}}$ where the assignments over the variables, both discrete and clocks coincide between the two traces except for the time evaluation, the clock variable c_M has always the same evaluation of the time of trace M. Moreover, we map each point t of σ with the corresponding point $t^{\mathcal{SK}}$ in the trace $\sigma^{\mathcal{SK}}$ as follows:

- For all $v \in V$: $\sigma^{\mathcal{SK}}(t^{\mathcal{SK}})(v) = \sigma(t)(v)$
- $\sigma^{\mathcal{SK}}(t^{\mathcal{SK}})(c_M) = \nu(t)$

It is easy to see that for any "skewed" trace there is a "perfect" trace and vice-versa. We can construct the timed trace σ^{sk} with $t' = \langle i, r' \rangle$ and $\sigma^{sk}(t')(c)$. We prove the following claim inductively over the structure of the formula:

$$\sigma, t \models \varphi \text{ iff } \sigma^{\mathcal{SK}}, t^{\mathcal{SK}} \models \varphi^{\mathcal{SK}}$$

The base case, follows from the equivalence relation assuming for the two traces.
Inductive case:

- $(\mathbf{U}_{\mathcal{I}})$ $\sigma, t \models \varphi_1 \mathbf{U}_{\mathcal{I}} \varphi_2$ iff there is $t' \geq t$ s.t. $\nu(t') - \nu(t) \in \mathcal{I}$, $\sigma, t' \models \varphi_2$ and, for each $t \leq t'' < t'$: $\sigma \models \varphi_1$. Since c_M in $\sigma^{\mathcal{SK}}$ is evaluated as ν in σ, and by induction then ... iff there is $t^{\mathcal{SK}\prime} \geq t^{\mathcal{SK}}$ s.t. $\sigma^{\mathcal{SK}}(t^{\mathcal{SK}\prime})(c_M) - \sigma^{\mathcal{SK}}(t^{\mathcal{SK}})(c_M) \in \mathcal{I}$, $\sigma^{\mathcal{SK}} \models \varphi_2^{\mathcal{SK}}$ and, for each $t^{\mathcal{SK}} \leq t^{\mathcal{SK}\prime\prime} < t^{\mathcal{SK}\prime}$: $\sigma^{\mathcal{SK}}, t^{\mathcal{SK}\prime} \models \varphi_1$, which from semantics is equivalent to $\sigma^{\mathcal{SK}}, t^{\mathcal{SK}} \models \varphi_1^{\mathcal{SK}} \mathbf{U}_{\mathcal{I}}^c \varphi_2^{\mathcal{SK}}$
- $(\mathbf{X})$ $\sigma, t \models \mathbf{X}\varphi$ iff there is $succ(t) \in T$ and $\sigma, succ(t) \models \varphi$. Recall that the only difference between the two traces is that the time evaluation; therefore, we can deduce that $succ(t)$ exists iff $succ(t^{\mathcal{SK}})$ exists as well. Then, the case trivially follows from induction.

- ($\tilde{\mathbf{X}}$) Same as $\mathbf{X}$.

□

We now define the adaptation of Definition 5.1 for the timed domain. In this setting, the composition must distinguish between discrete and timed transitions. Discrete transitions may interleave across components, reflecting independent actions. In contrast, timed transitions must be synchronized: all components must simultaneously perform a timed transition of the same duration. This synchronization ensures that time progresses consistently across the composed system. As a result, any point in the composed trace immediately preceding a timed transition must correspond to local points in each component trace. This guarantees that the timing structure of the global trace aligns with that of each individual system.

Definition 6.4 (Asynchronous Timed Automata Composition) *Let $M_1, \ldots, M_n$ be n compatible interface timed transition systems, let $run_1, \ldots, run_n$ be n Boolean variables not occurring in $M_1, \ldots, M_n$ (i.e. $run_1, \ldots run_n \notin \bigcup_{1\leq i\leq n}(V_i^I \cup V_i^O)$). $M_1 \otimes \cdots \otimes M_n = \langle V^I, V^O, \bigwedge_{1\leq i\leq n} I_i, T, \mathcal{SF}, Inv\rangle$ where:*

$$
\begin{aligned}
V^I &\doteq (\bigcup_{1\leq i\leq n} V_i^I \cup \{run_i\}) \setminus (\bigcup_{1\leq i<n} \bigcup_{i<j\leq n} V_i \cap V_j)\\
V^O &\doteq (\bigcup_{1\leq i\leq n} V_i^O) \cup (\bigcup_{1\leq i<n} \bigcup_{i<j\leq n} V_i \cap V_j)\\
T &\doteq \bigwedge_{1\leq i\leq n} ((run_i \rightarrow T_i) \wedge \psi_{cond}^{M_i})\\
\mathcal{SF} &\doteq \bigcup_{1\leq i\leq n} (\{\langle \top, run_i\rangle\} \cup \{\langle run_i \wedge \varphi_a, run_i \wedge \varphi_g\rangle | \langle \varphi_a, \varphi_g\rangle \in \mathcal{SF}_i\})\\
Inv &\doteq \bigwedge_{1\leq i\leq n} Inv_i
\end{aligned}
$$

where $\psi_{cond}^{M_i} \doteq \neg run_i \rightarrow \bigwedge_{v\in V_i^O} v = v'$.

In the following, we define the notion of projection lifting Definition 5.4 to timed traces. In this setting, instead of providing a direct mapping – which was defined over countable positions of the traces – we consider a mapping between intervals. Therefore, first we introduce a mapping between intervals indexes, and then we give a complete definition of the projected timed trace. We recall that, in our setting, each interval is either dense i.e. $\subset \mathbb{R}_0^+$ or a singleton i.e. t with $t \in T$ (see Section 2.7.1 for details). As mentioned before, we want to synchronize over timed transitions. To do so, in our

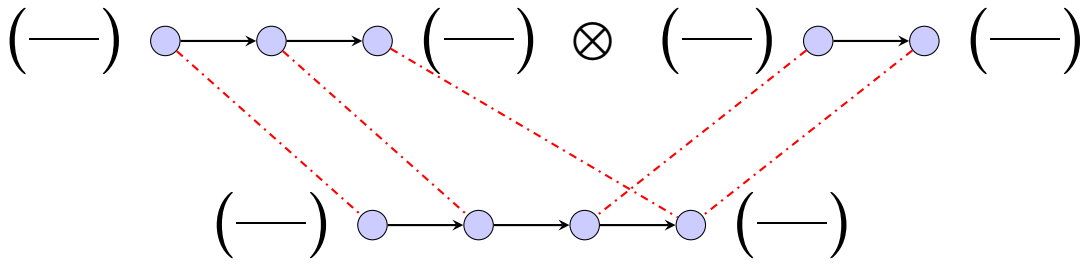


Figure 6.2: Figure showing the mapping between local timed traces to their composition. The timed interval are considered synchronously while discrete steps might interleave. Red lines map the local discrete point the mapped one in the composite trace. As for Figure 2.3, black lines between parenthesis represent timed transitions while violet circle represent discrete points. Since traces are synchronized over timed transitions, the point preceding the second timed transition is mapped to both local traces.

mapping, we consider the timed intervals, the discrete points where *run* is true and the points immediately preceding a timed transition (i.e. discrete points such that $\tilde{\mathbf{X}}\top$ is satisfied). Figure 6.2 provides a high-level idea of the composed trace and its relation with the local trace.

Definition 6.5 (Interval Mapping) *Given an infinite set of intervals $\mathcal{I}_0, \ldots$ defined as mandated by Section 2.7.1 from a trace σ of the composition of $M_1 \otimes \cdots \otimes M_n$, we define a mapping function from $\mathbb{N}$ to $\mathbb{N}$ for each M_h with $1 \leq h \leq n$ as follows. For each $k \in \mathbb{N}$: $map^{\tau}_k(M_h) = k'$ where $k' > map^{\tau}_{k-1}(M_h)$ such that either $\mathcal{I}_{k'}$ is a dense set or $\{t\} = \mathcal{I}_k$ and $\sigma, t \models run \vee \tilde{\boldsymbol{X}}\top$, and for each $map^{\tau}_{k-1}(M_h) < k'' < k'$: $\mathcal{I}_{k''} = \{t''\}$ is a singleton interval and $\sigma, t'' \not\models run_h \vee \tilde{\boldsymbol{X}}\top$.*

Moreover, given a point $t = \langle i, r\rangle$ of a trace of M_h, we denote $map^{\tau}_t(M_h) \doteq \langle map^{\tau}_i, r\rangle \in T$. [1]

Definition 6.6 (Timed Projection) *Let $\sigma = \langle \mathcal{M}, \tau, \overline{\mu}\rangle$ be a timed trace of $M_1 \otimes \cdots \otimes M_n$ with $\tau = \langle T, <, \boldsymbol{0}, \nu\rangle$ as its time model structure.*

We define the projected timed trace $\sigma_h = \langle \mathcal{M}, \tau_h, \overline{\mu}_h\rangle$ of M_i denoted as $Pr^{\tau}_{M_i}(\sigma)$ with the following characteristic:

1. *$\tau_i = \langle T_h, <, \boldsymbol{0}, \nu\rangle$ with $T_i = \{\langle k, r\rangle | for\ k \in \mathbb{N}, r \in \mathcal{I}_{map^{\tau}_k(M_h)}\}$*
2. *For all $t \in T_h$: $\sigma_h(t) = \sigma(t')(V_h)$ with $t' = map^{\tau}_t(M_h)$ $\sigma_i(t)$*

Moreover, we define the inverse operator of Pr^{τ}, denoted by $Pr^{\tau^{-1}}$:

$$Pr^{\tau^{-1}}_{M_i}(\sigma) = \{\sigma' | Pr^{\tau}_{M_i}(\sigma') = \sigma\}$$

[1] The inclusion is guaranteed by the fact that it is a point of the composition of all local traces.

6.1.2 Defining γ_P

We provided a notion of composition for the components with skewed clocks. Now we want to also provide a compositional rewriting for the temporal formulae. To do so, we apply the same process that we followed in Section 5.2. We start by considering a notion of timed projections and mapping – which will be used to map local traces to global traces. Then, we show a rewriting tailored for distributed MTL. Finally, we show that the local/global trace relation is preserved by the rewriting, entailing that we can apply the compositional Inference (Inference 5.1) in this setting as well.

To simplify the notation, we assume to be given n ITTS $M_1, \ldots, M_n$, a composed ITTS $M \doteq M_1 \otimes \cdots \otimes M_n$, a timed trace σ_i of M_i, with $1 \leq i \leq n$ and a local distributed MTL formula φ. For brevity, we refer to $\mathcal{R}^{\tau}_{M_i}, \mathcal{R}^{\tau *}_{M_i}, Pr^{\tau}_{M_i}, Pr^{\tau -1}_{M_i}$ and run_i as respectively $\mathcal{R}^{\tau}, \mathcal{R}^{\tau *}, Pr^{\tau}, Pr^{\tau -1}$ and run.

For each $\langle map^{\tau}_i, r\rangle = t' \in T'$, $\sigma_i(t') = \sigma(t)(V_i)$ where $t = \langle i, r\rangle \in T$.

We can now define a rewriting considering local time with non-resettable skewed clocks as follows

Definition 6.7 (Rewriting of Distributed MTL) *We define the rewriting $\mathcal{R}^{\tau}$ as follows:*

$$\mathcal{R}^{\tau}(\varphi_1 \,\mathbf{U}^{c}_{\mathcal{I}}\, \varphi_2) \doteq ((\neg(run \vee \tilde{\boldsymbol{X}}\top) \vee \mathcal{R}^{\tau}(\varphi_1))\, \mathbf{U}^{c}_{\mathcal{I}}\, ((run \vee \tilde{\boldsymbol{X}}\top) \wedge \mathcal{R}^{\tau}(\varphi_2))$$
$$\mathcal{R}^{\tau}(\boldsymbol{X}\varphi) \doteq \boldsymbol{X}((\neg run \wedge \boldsymbol{X}\top)\, \mathbf{U}\, ((run \vee \tilde{\boldsymbol{X}}\top) \wedge \mathcal{R}^{\tau}(\varphi))$$
$$\mathcal{R}^{\tau}(\tilde{\boldsymbol{X}}\varphi) \doteq \tilde{\boldsymbol{X}}\mathcal{R}^{\tau}(\varphi)$$

finally, we define $\mathcal{R}^{\tau *}(\varphi) \doteq \mathcal{R}^{\tau}(\varphi)\, \mathbf{R}\, \neg(run \vee \tilde{\boldsymbol{X}}\top)$

Lemma 6.1 ($\mathcal{R}^{\tau}$ Mapping) *For all $\sigma^{ST} \in Pr^{-1}(\sigma)$:*

$$\sigma, t \models \varphi \Leftrightarrow \sigma^{ST}, map^{\tau}_t \models \mathcal{R}^{\tau}(\varphi)$$

Proof: The proof is a straightforward adaptation of the proof of Lemma 5.1 for timed systems with infinite traces. The base case trivially holds because the assignments over the variables in position t are identical to the one of the variables (V_h) in position map^{τ}_t.

The inductive case is a straightforward adaptation of the discrete case considering as synchronization points also timed positions.

- $\mathbf{U}^{c}_{\mathcal{I}}$: Trivial adaptation of $\mathbf{U}$ case of Lemma 5.1.
- $\mathbf{X}$: Trivial adaptation of $\mathbf{X}$ case of Lemma 5.1.

- $\tilde{\mathbf{X}}$: It follows because timed points and points such that $\tilde{\mathbf{X}}\top$ are true are in the same mapping intervals. Then, we can deduce it by induction.

□

Lemma 6.2 ($\mathcal{R}^{\tau *}$ Mapping) *For all* $\sigma^{ST} \in Pr^{\tau -1}(\sigma) : \sigma^{ST}, map_0^\tau \models \mathcal{R}^\tau(\varphi)$ *iff* $\sigma^{ST}, \boldsymbol{0} \models \mathcal{R}^{\tau *}(\varphi)$

Proof: Same as for Lemma 5.2. □

Theorem 6.2 (Soundness of Rewriting) *Let* σ *be a timed trace of* M_i*, for all* $\sigma^{ST} \in Pr^{\tau -1}(\sigma_i)$*:* $\sigma \models \varphi$ *iff* $\sigma^{ST} \models \mathcal{R}^{\tau *}(\varphi)$

Proof: Follows from Lemma 6.1 and Lemma 6.2. □

Finally, the formula composition γ_P for timed systems is defined as

$$\gamma_P(\varphi_1, \ldots, \varphi_n) \doteq \bigwedge_{1 \leq i \leq n} \psi_{cond}(M_1, \ldots, M_n) \wedge \mathcal{R}^{\tau *}(\varphi_1^{\mathcal{SK}}) \wedge \cdots \wedge \mathcal{R}^{\tau *}(\varphi_n^{\mathcal{SK}})$$

6.2 Reasoning over Distributed Real-Time Systems with MTL

In the previous section, we discussed how to apply compositional reasoning over timed component systems with local skewed clocks assuming that the clock were not resettable.

In this section, we consider components of DRTS that use local clocks that are occasionally reset for synchronization and, so, that may be not monotonic. This may happen in practice, for example, when a component uses a local clock to send a message periodically and the clock is sometimes updated for synchronization with other components by means of a distributed algorithm for approximating real-value variables (cfr., e.g., [235]). Standard metric operators (i.e. MTL operators), which are typically defined over global timestamps rather than local clocks, are not directly applicable in this setting. Since components use local clocks that may be occasionally reset for synchronization, expressing properties relative to these clocks requires operators that can account for non-monotonic time. For example, suppose to specify that a certain condition b holds for p time units with respect to a local clock c. If c is monotonic, this

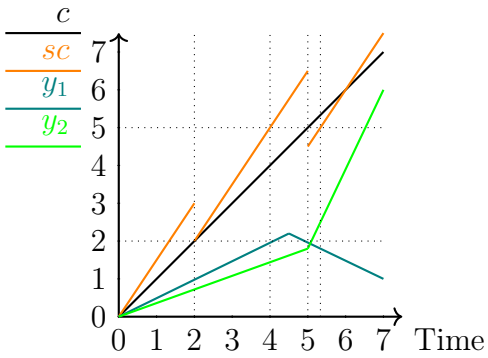


Figure 6.3: Examples of real functions, including a perfect and a skewed resettable clock

property can be formalized in a distributed version of MTL with the formula $\mathbf{G}^c_{\leq p} b$. If c is not monotonic, the same formula requires b to hold in all (possibly disconnected) points that are less than p, which is not what intended.

As a concrete example, consider the plot in Figure 6.3. Let us first refer to the perfect clock c (black line). The formula $\mathbf{F}^c_{\leq 5}(y_1 \geq 2)$ is true in 0 because there is a point between time 4 and 5 where $y_1 \geq 2$. For the same reason, the formula $\mathbf{G}^c_{\leq 5}(y_1 \leq 2)$ (which is equivalent to the negation of the first one) is false. If we consider instead y_2, the formula $\mathbf{G}^c_{\leq 5}(y_2 \leq 2)$ is true.

Let us now consider the skewed clock sc (orange line), which runs too fast and is reset at points 2 and 5 to approximately the correct value (black line). If we consider again $\mathbf{F}^{sc}_{\leq 5}(y_1 \geq 2)$, the formula is now false because of the drift as $y_1 \geq 2$ should hold before. Note however that thanks to the reset the points where $sc(t) \leq 5$ lie in disconnected intervals, namely $[0, 4]$ and $[5, 16/3]$. Thus, while $G^c_{\leq 5}(y_1 \leq 2)$ is true, $G^c_{\leq 5}(y_2 \leq 2)$ is false.

Thus, in this section, we define alternative metric operators $\overline{U}$ and its derivates $(\overline{G}, \overline{F})$ with a semantics that is more suitable to specify properties of components that use skewed resettable clocks. For example, $\overline{G}^c_{\leq p} b$ requires a component to keep b true for the first p time units according to its local clock c, without relying on a clock reset, since this is not under its control.

Outline In Section 6.2.1, we define the novel logics MTLSK which extends distributed MTL with an alternative "first" version of until to better deal with components of DRTS that uses local resettable-skewed clocks; in Section 6.2.2, we show a decision procedure for the satisfiability of a parametrized variant of MTLSK by reducing the verification using an intermediate logic and a subsequent discretization; in Section 6.2.3, we present

a proof-of-concept tool to check validity of MTLSK formulae with a small experimental evaluation; finally, in Section 6.2.4, we adapt the compositional reasoning approach to deal with MTLSK.

6.2.1 MTLSK

Definition 6.8 *Given a signature Σ, a set of variables V, and a set of clock variables $\chi \subseteq V$, MTLSK formulae are built with the following grammar:*

$$\phi := Pred \mid \phi \wedge \phi \mid \neg\phi \mid \phi \overline{\mathbf{U}}^{c}_{\mathcal{I}} \phi \mid \phi \mathbf{U}^{c}_{\mathcal{I}} \phi$$

where Pred is a predicate over V, $c \in \chi$, and $\mathcal{I} \subseteq \mathbb{R}$.

Abbreviations are defined similarly to the standard case.

The semantics of the new operator is defined as follows:

$$\begin{aligned} \sigma, t \models \varphi \overline{\mathbf{U}}^{c}_{\mathcal{I}} \psi \Leftrightarrow \text{exists } t' > t, \text{ s. t.} \\ \sigma(t')(c) - \sigma(t)(c) \in \mathcal{I}, \sigma, t' \models \psi, \text{ and} \\ \text{for all } t \leq t'' < t : \\ \sigma, t'' \models \varphi \text{ and } \sigma(t'')(c) - \sigma(t)(c) \in \mathcal{I}^{-} \end{aligned}$$

where $\mathcal{I}^{-} \doteq \mathcal{I} \cup (-\infty, inf(\mathcal{I})]$ and $inf(\mathcal{I})$ is the infimum of $\mathcal{I}$ i.e. the greatest lower bound of the interval $\mathcal{I}$.

$\overline{U}$ extends U to guarantee that in each point t'' between t and t', the difference between c at step t'' and c at step t is below the upper threshold of $\mathcal{I}$. Thus, surpassing the upper threshold of $\mathcal{I}$ without an occurrence of ψ falsifies $\varphi \overline{\mathbf{U}}^{c}_{\mathcal{I}} \psi$ while it does not falsify $\varphi \mathbf{U}^{c}_{\mathcal{I}} \psi$ since ψ might hold in a future point after a reset.

It should be noted that with resettable clocks, the interval domain is $\mathbb{R}$ instead of the positive counterpart used in MTL. It happens because clock resets may decrease clocks. Indeed, $\mathcal{I}$ is defined as a subset of $\mathbb{R}$ instead of $\mathbb{R}^{+}$. We say that a clock is weakly-monotonic if it is never decreased by a reset, so that its value over time either remains constant or increases.

Theorem 6.3 *For all σ, φ: $\sigma \models \varphi \overline{\mathbf{U}}^{c}_{\mathcal{I}} \psi \Rightarrow \sigma \models \varphi \, \mathbf{U}^{c}_{\mathcal{I}} \, \psi$ and*
If $\sup(\mathcal{I}) = +\infty$ i.e. the interval is open on the right, $\sigma \models \varphi \mathbf{U}^{c}_{\mathcal{I}} \psi \Leftrightarrow \sigma \models \varphi \overline{\mathbf{U}}^{c}_{\mathcal{I}} \psi$.
If there is no reset (weakly monotonic case)

$\sigma \models \varphi \overline{\mathbf{U}}^c_{\mathcal{I}} \psi \Leftrightarrow \sigma \models \varphi \, \mathbf{U}^c_{\mathcal{I}} \, \psi$
Moreover, if there is no drift and no reset, i.e., $\epsilon = 0 \wedge \lambda = 0$ (perfect clocks), then

$$\sigma \models \varphi \overline{\mathbf{U}}^c_{\mathcal{I}} \psi \Leftrightarrow \sigma \models \varphi \, \mathbf{U}_{\mathcal{I}} \, \psi \Leftrightarrow \sigma \models \varphi \mathbf{U}^c_{\mathcal{I}} \psi$$

Proof: The parts of the theorem are proved in the following list:

- Case $\sup(I) = \infty$: $\sigma, t \models \varphi \mathbf{U}^c_I \psi \Leftrightarrow \sigma, t \models \varphi \overline{\mathbf{U}}^c_I$: $\sigma, t \models \varphi \overline{\mathbf{U}}^c_I \psi \Leftrightarrow \exists t' \geq t : \sigma, t' \models \psi$ and $\sigma(t')(c) - \sigma(t)(c) \in I$ and $\forall t \leq t'' < t : \sigma, t'' \models \varphi$ and $\sigma(t'')(c) - \sigma(t)(c) \in I^-$ Since $\sup(I) = +\infty$, then $I^- = \mathbb{R}$ and thus $\sigma(t'')(c) - \sigma(t)(c) \in I^-$ is always true. Thus, $\sigma, t \models \varphi \mathbf{U}^c_I \psi \Leftrightarrow \sigma, t \models \varphi \overline{\mathbf{U}}^c_I \psi$ if I is either $(p, +\infty)$ or $[p, +\infty)$.

- $\sigma, t \models \varphi \overline{\mathbf{U}}^c_I \psi \Rightarrow \sigma, t \models \varphi \mathbf{U}^c_I \psi$. This theorem follows trivially by the semantics of the two operators. $\overline{\mathbf{U}}^c_I$ extends $\mathbf{U}^c_I$ forcing all t'' points to have a clock valuation in I^-.

- Case no reset: $\sigma, t \models \varphi \overline{\mathbf{U}}^c_I \psi \Leftrightarrow \sigma, t \models \varphi \mathbf{U}^c_I \psi$. Since no reset occurs, then c is weakly monotonic. Thus, it follows that $\sigma, t' \models \sigma(t')(c) - \sigma(t)(c) \in I \Rightarrow \forall t \leq t'' < t' : \sigma(t'')(c) - \sigma(t)(c) \in I^-$. Thus, if no reset occurs $\sigma, t \models \varphi \overline{\mathbf{U}}^c_I \psi \Leftrightarrow \sigma, t \models \varphi \mathbf{U}^c_I \psi$.

- Case no drift and no reset: $\sigma, t \models \varphi \mathbf{U}^c_I \psi \Leftrightarrow \sigma, t \models \varphi \overline{\mathbf{U}}^c_I \psi \Leftrightarrow \sigma, t \models \varphi \, \mathbf{U}_I \, \psi$ Since there are no resets $\sigma, t \models \varphi \overline{\mathbf{U}}^c_I \psi \Leftrightarrow \sigma, t \models \varphi \mathbf{U}^c_I \psi$. Since clocks are perfect:$\forall t : \sigma(t)(c) = \nu(t)$. Thus, $\sigma, t \models \varphi \mathbf{U}^c_I \psi \Leftrightarrow \exists t' \geq t : \sigma, t' \models \psi$ and $\sigma(t')(c) - \sigma(t)(c) \in I$ and $\forall t \leq t'' < t' : \sigma, t'' \models \varphi \Leftrightarrow \exists t' \geq t : \sigma, t' \models \psi$ and $\nu(t') - \nu(t) \in I$ and $\forall t \leq t'' < t' : \sigma, t'' \models \varphi \Leftrightarrow \sigma, t \models \varphi \, \mathbf{U}_I \, \psi$.

□

6.2.2 Encoding and Verification of Parametric MTLSK

This section introduces an encoding from a parametric variant of MTLSK to LTL($\mathcal{T}$). The encoding is divided into 3 parts: the first part considers a novel intermediate logic LTL-min-max that can express MTLSK properties through a straightforward rewriting Υ; the second part defines a discretization process of LTL-min-max based on the discretization proposed in [282]; the last part defines an encoding from LTL-min-max to First-order LTL.

In the following, we define our parametric variation of MTLSK. To simplify the encoding, we consider intervals of the form $[0, p)$,$[0, p]$, $(p, +\infty)$ and $[p, +\infty)$; moreover,

we allow clock predicates to occur guarded with discrete transitions. We add these limitations to simplify the definition of the operators of our intermediate logic LTL-min-max and to simplify the discretization. We do not introduce $\overline{\mathbf{U}}^c$ for intervals going from p to infinity because, for such fragment, the semantics of the two operators is equivalent.

Definition 6.9 *(Parametric MTLSK) Given a signature Σ, a set of variables V, and a set of clock variables $\chi \subseteq V$, MTLSK formulae are built with the following grammar:*

$$\phi := Pred(u, \ldots, u) \mid \phi \wedge \phi \mid \neg\phi \mid \boldsymbol{X}\phi \mid \tilde{\boldsymbol{X}}\phi \mid \phi \mathbf{U}^c_{\lhd u}\phi \mid \phi \mathbf{U}^c_{\rhd u}\phi \mid \phi \overline{\mathbf{U}}^c_{\lhd u}\phi$$

where $c \in \chi$, $\lhd \in \{<, \leq\}$,$\rhd \in \{>, \geq\}$, and u is defined as in LTL but clock variables can occur only guarded by discrete transition i.e. if a predicate Pred contains $c \in \chi$, it must occur in the form $(\boldsymbol{X}\top \rightarrow Pred)$.

We just define the semantics of metric operators, while the other cases are defined as for LTL. We assume here that the background first-order theory contains the theory of reals and that the clock variables and each parametric bound has real type.

- $\sigma, t \models \varphi \, \mathbf{U}^c_{\lhd p} \, \psi$ iff there exists $t' \geq t$ such that $\sigma(t')(c) - \sigma(t)(c) \lhd \sigma(p)$ and $\sigma, t' \models \psi$ and for all $t'', t \leq t'' < t', \sigma, t'' \models \varphi$

- $\sigma, t \models \varphi \, \mathbf{U}^c_{\rhd p} \, \psi$ iff there exists $t' \geq t$ such that $\sigma(t')(c) - \sigma(t)(c) \rhd \sigma(p)$ and $\sigma, t' \models \psi$ and for all $t'', t \leq t'' < t', \sigma, t'' \models \varphi$

- $\sigma, t \models \varphi \overline{\mathbf{U}}^c_{\lhd p} \psi$ iff there exists $t' \geq t$ such that $\sigma(t')(c) - \sigma(t)(c) \lhd \sigma(p)$, $\sigma, t' \models \psi$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi$ and $\sigma(t'')(c) - \sigma(t)(c) \lhd \sigma(p)$

It is easy to see that Theorem 6.3 holds for this parametric fragment as well.

LTL-min-max Definition and Rewriting We construct a new intermediate logic LTL-min-max extending First-order LTL with minimum and maximum operators. The intuition behind this logic follows from the following remark: while the satisfaction of bounded operators in $\mathrm{MTL}_{0,+\infty}$ can be achieved by looking at the value of time in the next occurrence of the formula, in MTLSK due to non-monotonicity of time we have to analyze the minimum and maximum time in which the property holds.

Definition 6.10 (LTL-min-max Syntax) *Given a signature Σ, a set of variables V, and a set of clock variables $\chi \subseteq V$, LTL-min-max formulae are built with the following*

grammar:

$$\begin{aligned}
\phi :=& Pred(u,\dots,u) \mid tu \bowtie cu \mid \phi \wedge \phi \mid \neg\phi \mid \phi \mathbf{U} \phi \mid \boldsymbol{X}\phi \mid \tilde{\boldsymbol{X}}\phi \\
u :=& cu \mid v \mid func(u,\dots,u) \mid ite(\phi,u,u) \mid v' \\
tu :=& time \mid c \mid min\Delta^c_{\phi \mathbf{U} \phi} \mid f(tu,\dots,tu) \mid c' \mid max\Delta^c_{\phi \mathbf{U} \phi} \mid maxbef\,\Delta^c_\phi \\
cu :=& p \mid f(cu,\dots,cu)
\end{aligned}$$

where $\bowtie \in \{\leq, \geq, <, >\}$, *$p$ is a real-value parameter,* $c \in \chi$, $v \in V \setminus \chi$ *and clock variables can occur only in event predicates and in min-max operator superscripts i.e.* $min\Delta^c_{\phi \mathbf{U} \psi}$, $max\Delta^c_{\varphi \mathbf{U} \psi}$ *and* $maxbef\,\Delta^c_\varphi$.

Definition 6.11 *The semantics of LTL-min-max operators is defined as follows:*

If $\sigma, t \models \varphi \,\mathbf{U}\, \psi$ *then* $\sigma(t)(min\Delta^c_{\varphi \mathbf{U} \psi}) = \min(\sigma(t)(U^c_{\varphi \mathbf{U} \psi})) - \sigma(t)(c)$

If $\sigma, t \models (\varphi \,\mathbf{U}\, \psi) \wedge \Phi_{finU}$ *then* $\sigma(t)(max\Delta^c_{\varphi \mathbf{U} \psi}) = \max(\sigma(t)(U^c_{\varphi \mathbf{U} \psi})) - \sigma(t)(c)$

If $\sigma, t \models \boldsymbol{F}\varphi$ *then* $\sigma(t)(maxbef\,\Delta^c_\varphi) = \max(Bef^c_\sigma(t,\varphi)) - \sigma(t)(c)$

If $\sigma, t \models \varphi$ *then* $\sigma(t)(ite(\varphi,u_1,u_2)) = \sigma(t)(u_1)$,

otherwise $\sigma(t)(ite(\varphi,u_1,u_2)) = \sigma(t)(u_2)$

where: $\Phi_{finU} \doteq \boldsymbol{F}(\neg\varphi \vee \boldsymbol{G}\neg\psi)$,

$\sigma(t)(U^c_{\varphi \mathbf{U} \psi}) \doteq \{\sigma(t')(c) | t' \geq t : \sigma, t' \models \psi$ *and for all* $t \leq t'' < t' : \sigma, t'' \models \varphi\}$,

$\sigma(t)(Bef^c_\varphi) \doteq \{\sigma(t')(c) | t' \geq t :$ *for all* $t < t'' < t' : \sigma, t'' \not\models \varphi\}$.

The set $\sigma(t)(U^c_{\varphi \mathbf{U} \psi})$ represents the value of each $\sigma(t')(c)$ such that ψ holds at point t' and each point between t and t' (excluded) satisfies φ. This set takes all the witnesses of σ satisfying $\varphi \,\mathbf{U}\, \psi$ from point t; then, it extracts the value of c at point t' i.e. when ψ holds. $min\Delta^c_{\varphi \mathbf{U} \psi}$ and $max\Delta^c_{\varphi \mathbf{U} \psi}$ represent respectively the minimum and the maximum of that set. The existence of the minimum of $\sigma(t)(U^c_{\varphi \mathbf{U} \psi})$ is guaranteed if $\varphi \,\mathbf{U}\, \psi$ holds. $\sigma, t \models \varphi \,\mathbf{U}\, \psi$ is a sufficient condition because clocks diverge (see Definition 6.1) and ψ contains clocks only inside predicates that are evaluated during discrete transitions. Similarly, the assumptions on $max\Delta$ guarantee that the maximum exists if the property holds and holds only finitely many times. If Φ_{finU} does not hold, then the maximum does not exist because the clock diverges.

$\sigma(t)(Bef^c_\psi)$ is the set containing all clock values from point t to the first point such that ψ holds after t. $maxbef\,\Delta^c_\psi$ represents the maximum in that set. $\mathbf{F}\varphi$ guarantees the existence of the maximum of that set.

Definition 6.12 *We define the rewriting* Υ *from MTLSK to LTL-min-max as follows:*

$$\Upsilon(\varphi \mathbf{U}^{c}_{\lhd p}\psi) \doteq \Upsilon(\varphi \,\mathbf{U}\, \psi) \wedge min\Delta_{\Upsilon(\varphi \mathbf{U}\psi)} \lhd p$$
$$\Upsilon(\varphi \mathbf{U}^{c}_{\rhd p}\psi) \doteq \Upsilon(\boldsymbol{G}(\varphi \wedge \boldsymbol{F}\psi)) \vee \Upsilon(\varphi \,\mathbf{U}\, \psi) \wedge max\Delta_{\Upsilon(\varphi \mathbf{U}\psi)} \rhd p$$
$$\Upsilon(\varphi \overline{\mathbf{U}}^{c}_{\lhd p}\psi) \doteq \Upsilon(\varphi \,\mathbf{U}\, \psi) \wedge maxbef\Delta^{c}_{\Upsilon(\psi)} \lhd p$$

Theorem 6.4 φ *and* $\Upsilon(\varphi)$ *are equisatisfiable*

We first provide a sketch proof.

Proof: [Sketch] $\varphi \mathbf{U}^{c}_{\bowtie p}\psi$ is true at point t only if there exist a point t' such that ψ hold, $\sigma(t')(c) - \sigma(t)(c) \bowtie \sigma(p)$ and all points from t to t' satisfy φ. Since $min\Delta^{c}_{\varphi \mathbf{U}\psi}$ and $max\Delta^{c}_{\varphi \mathbf{U}\psi}$ are respectively the minimum/maximum of the set containing the evaluation of c in each t' point minus $\sigma(t)(c)$, then the translation holds. The same applies to $\varphi \overline{\mathbf{U}}^{c}_{\lhd p}\psi$. In that case, the property holds iff $\varphi \,\mathbf{U}\, \psi$ holds and each value of $\sigma(t'')(c)$ with $t \leq t'' \leq t'$ is not greater than the upper bound $\sigma(t)(c) + \sigma(p)$. Thus, it is sufficient to verify that the $max(\sigma(t)(Bef)) - \sigma(t)(c) \lhd \sigma(p)$. □

The complete proof is as follows:

Proof: We prove the theorem inductively on the length of the formula.

Base case: Base case is trivial since the formulae are equal, and thus, they are also equisatisfiable.

Inductive case: We ignore homomorphic transformations since they trivially hold.

- $\varphi \mathbf{U}^{c}_{\lhd p}\psi$:

  $\sigma, t \models \varphi \mathbf{U}^{c}_{\lhd p}\psi \Leftrightarrow \exists t' \geq t . \sigma, t' \models \psi$ and $\sigma(t')(c) - \sigma(t)(c) \lhd p$ and $\forall t < t'' < t' : \sigma, t'' \models \varphi$. By U set definition $\sigma(t)(U^{c}_{\varphi \mathbf{U}\psi}) \neq \emptyset \Leftrightarrow \sigma, t \models \varphi \,\mathbf{U}\, \psi$.

  If $\sigma, t \not\models \varphi \,\mathbf{U}\, \psi$ then both the formulae are not satisfied. If $\sigma, t \models \varphi \,\mathbf{U}\, \psi$ then $\sigma, t \models \varphi \,\mathbf{U}^{c}_{\lhd p}\, \psi \Leftrightarrow \exists c_v \in U^{c}_{\sigma}(t, \varphi, \psi)$ s.t. $c_v - \sigma(t)(c) \lhd p$. Since c diverges (i.e. $\forall t, \exists t' \geq t$ s.t. $\sigma(t')(c) > \sigma(t)(c)$) and by assumption λ bounds the reset lower value that could be reached, $min(U^{c}_{\sigma}(t, \varphi, \psi)$ is defined. Since min is the smallest value of the set, then $\exists c_v \in U^{c}_{\sigma}(t, \varphi, \psi)$ s.t. $c_v - c \lhd p \Leftrightarrow \min(U^{c}_{\sigma}(t, \varphi, \psi)) - c \lhd p \Leftrightarrow \sigma, t \models min\Delta^{c}_{\varphi \mathbf{U}\psi} \lhd p$. By induction hypothesis φ and $\Upsilon(\varphi)$ are equisatisfiable, ψ and $\Upsilon(\psi)$ are equisatisfiable; thus, $\cdots \Leftrightarrow \sigma, t \models min\Delta^{c}_{\Upsilon(\varphi \mathbf{U}\psi)} \lhd p$.

- $\varphi \mathbf{U}^c_{\triangleright p} \psi$: If $\sigma, t \models \mathbf{G}(\varphi \wedge \mathbf{F}\psi)$ then there are infinitely many $t' > t$ s.t. $\sigma, t' \models \psi$ and $\sigma(t')(c) - \sigma(t)(c) \triangleright p$ and $\forall t \leq t'' < t : \sigma, t'' \models \varphi$ (because clock diverges). Thus, if $\sigma, t \models \mathbf{G}(\varphi \wedge \mathbf{F}\psi)$ both properties holds. As for $\triangleleft$ case, if $\varphi \,\mathbf{U}\, \psi$ does not hold, then both properties are falsified. The remaining of this case is equal to the $\triangleleft$ case (just replace *min* with max).

- $\varphi \overline{\mathbf{U}}^c_{\triangleleft p} \psi$: If $\sigma, t \not\models \varphi \,\mathbf{U}\, \psi$ then neither $\varphi \overline{\mathbf{U}}^c_{\triangleleft p} \psi$ nor $\varphi \,\mathbf{U}\, \psi \wedge \mathit{maxbef}\, \Delta^c_{v(\varphi)} \triangleleft p$ holds. If $\sigma, t \models \varphi \,\mathbf{U}\, \psi$, then $\sigma(t)(\mathit{maxbef}\, \Delta^c_\psi) = \max(\sigma(t)(\mathit{Bef}^c_\psi)) - \sigma(t)(c)$. By $\overline{\mathbf{U}}^c_{\triangleleft p}$ definition, $\sigma, t \models \varphi \overline{\mathbf{U}}^c_{\triangleleft p} \psi \Leftrightarrow \sigma, t \models \varphi \,\mathbf{U}\, \psi$ and $\forall c_v \in \sigma(t)(\mathit{Bef}^c_\psi) c_v - c \triangleleft p \Leftrightarrow \max(\mathit{Bef}^c_\psi) - c \triangleleft p \Leftrightarrow \sigma, t \models \mathit{maxbef}\, \Delta^c_\psi \triangleleft p$. (Similarly to the previous cases, all the points are below the threshold iff their maximum is below that threshold).

□

LTL-min-max Discretization We apply our discretization process based on the one defined in [282]. The discretization process defines time evolution as a sequence of open or singular intervals. The discretization process assumes that the satisfiability of each subformula of ϕ does not vary inside intervals. That assumption is ensured because MTLSK allows clocks only as part of event predicates.

Singular intervals are encoded by the fresh variable ι while the time elapsed in each interval is encoded by δ. The real variable ζ encodes an arbitrary accumulation of sums of δ. ζ is used to guarantee the divergence of the global time. For each skewed clock c, we introduce two fresh variables: δ_c and diff_c. δ_c encodes the clock time elapse in each transition while diff_c represents the difference between the global time and the current clock value c. We encode c using diff_c to guarantee the divergence of c relying on the divergence of global time.

The rewritten formula ϕ_D is composed of four parts: the constraints defining time and forcing discrete variables to stutter when time elapses (ψ_{time}); the constraints defining each clock c according to Definition 6.1 (ψ^c_{clock}); the constraint to define when an interval is open or when it is singular (ψ_ι) and; the discretized formula ($\mathcal{D}(\phi)$). The

whole transformation is as follows:

$$\begin{aligned}
\phi_D &\doteq \psi_{time} \wedge \bigwedge_{c\in\chi} \psi^c_{clock} \wedge \psi_\iota \wedge \mathcal{D}(\phi)\\
\psi_{time} &\doteq time = 0 \wedge \mathbf{G}(time' - time = \delta) \wedge \mathbf{G}(\delta > 0 \rightarrow \bigwedge_{v\in V\setminus\chi} (v' = v))\\
\psi^c_{clock} &\doteq diff_c = 0 \wedge \mathbf{G}(diff'_c - diff_c = \delta_c - \delta)\wedge\\
&\quad \mathbf{G}((\delta > 0 \rightarrow \delta_c \in [\delta(1-\epsilon), \delta(1+\epsilon)]) \wedge (\delta = 0 \rightarrow |diff_c| \leq \lambda))\\
\psi_\iota &\doteq \iota \wedge \mathbf{G}((\iota \wedge \delta = 0 \wedge \mathbf{X}\iota) \vee (\iota \wedge \delta > 0 \wedge \mathbf{X}\neg\iota) \vee (\neg\iota \wedge \delta > 0 \wedge \mathbf{X}\iota))\wedge\\
&\quad \mathbf{G}((\zeta' - \zeta = \delta) \vee (\zeta \geq 1 \wedge \zeta = 0)) \wedge \mathbf{GF}(\zeta \geq 1 \wedge \zeta' = 0)
\end{aligned}$$

Definition 6.13 (Discretization Rewriting) *The discretization rewriting $\mathcal{D}$ is defined as follows:*

$$\begin{aligned}
&\mathcal{D}(\boldsymbol{X}\varphi) \doteq \iota \wedge \boldsymbol{X}(\iota \wedge \mathcal{D}(\varphi)) \quad \mathcal{D}(\tilde{\boldsymbol{X}}\varphi) \doteq (\neg\iota \wedge \mathcal{D}(\phi)) \vee \boldsymbol{X}(\neg\iota \wedge \mathcal{D}(\varphi))\\
&\mathcal{D}(\varphi\ \mathbf{U}\ \psi) \doteq \mathcal{D}(\psi) \vee (\mathcal{D}(\varphi)\ \mathbf{U}\ \tilde{\psi}) \quad \mathcal{D}(maxbef\,\Delta^c_\varphi) \doteq maxbef\,\Delta^c_{\mathcal{D}(\varphi)}\\
&\mathcal{D}(min\Delta^c_{\varphi\mathbf{U}\psi}) \doteq ite(\mathcal{D}(\psi) \wedge 0 \leq min\Delta^c_{\mathcal{D}(\varphi)\mathbf{U}\tilde{\psi}}, 0, min\Delta^c_{\mathcal{D}(\varphi)\mathbf{U}\tilde{\psi}})\\
&\mathcal{D}(max\Delta^c_{\varphi\mathbf{U}\psi}) \doteq ite(\mathcal{D}(\psi) \wedge 0 \geq max\Delta^c_{\mathcal{D}(\varphi)\mathbf{U}\tilde{\psi}}, 0, max\Delta^c_{\mathcal{D}(\varphi)\mathbf{U}\tilde{\psi}})\\
&\textit{where } \tilde{\psi} = \mathcal{D}(\psi) \wedge (\iota \vee \mathcal{D}(\varphi)).
\end{aligned}$$

$\mathbf{X}\varphi$ is discretized forcing a discrete transition i.e. $\iota \wedge \mathbf{X}\iota$. $\tilde{\mathbf{X}}\varphi$ requires that either φ holds now in an open interval or it holds in the next state in an open interval. Until forces either ψ to hold now or φ has to hold until ψ holds; moreover, if ψ holds in an open interval also φ must hold in that point too. The discretization of $min\Delta^c_{\varphi\mathbf{U}\psi}$ and $max\Delta^c_{\varphi\mathbf{U}\psi}$ is similar to the one of until. It considers the current point in the minimum/maximum computation as a candidate min/max and then applies the discrete operator on the discretized until. It should be noted that, even if $\mathcal{D}(\psi)$ holds at point t_i, if $min\Delta^c_{\mathcal{D}(\varphi)\mathbf{U}\tilde{\psi}}$ is lower than 0, we still need to pick $min\Delta^c_{\mathcal{D}(\varphi)\mathbf{U}\tilde{\psi}}$; this is necessary because the value of the clock might decrease in the future.

Theorem 6.5 *ϕ and ϕ_D are equisatisfiable*

Proof: We use the proofs described in [98] and [282] with the difference that since in our case clocks can occur only guarded with discrete transitions, we do not need continuity constraints. This guarantees that σ is fine w.r.t. the formula i.e. the evaluation of a predicate over an interval is constant.

Therefore, we need to extend the inductive proof for $\mathcal{D}$ to $min\Delta$, $max\Delta$ and $maxbef\,\Delta$.

Subsequently, we consider the notation of [98] where σ represents the timed trace, σ_D the discrete trace, t_i represents the point in the timed trace represented by the interval t_i and i as its discrete point counterpart.

- $min\Delta$. Suppose that $\sigma, t_i \models \varphi \mathbf{U} \psi$. $\sigma(t_i)(min\Delta^c_{\varphi \mathbf{U} \psi}) = min(\sigma(t)(U^c_{\varphi \mathbf{U} \psi}))$. $\sigma(t_i)(U^c_{\varphi \mathbf{U} \psi}) = \{\sigma(t')(c) | t' \geq t \text{ and } \sigma, t' \models \psi \text{ and } \forall t \leq t'' < t' : \sigma, t'' \models \varphi\})$.

  Which is equal to $\{\sigma(t_i)(c) \text{if } \sigma, t_i \models \psi\} \cup \{\sigma(t)(c) | t' > t_i \sigma, t' \models \psi \text{ and } \forall t_i \leq t'' < t' \sigma, t'' \models \varphi\}$. By induction, $t' \in I_j$ for some j. Since σ is fine, $\sigma, t_j \models \psi$ and if t_j is open then $\sigma, t_j \models \varphi$. Thus, $\cdots = \{\sigma_D(i)(c)\} \cup \sigma_D(i)(U^c_{\mathcal{D}(\varphi) \mathbf{U} (\mathcal{D}(\psi) \wedge (\iota \vee \mathcal{D}(\varphi)))})$ if $\sigma_D, i \models \mathcal{D}(\psi)$. Otherwise, $\cdots = U^c_{\mathcal{D}(\varphi) \mathbf{U} (\mathcal{D}(\psi) \wedge (\iota \vee \mathcal{D}(\varphi)))}$. Thus, the discretization is correct.

- $max\Delta$ As $min\Delta$.

- $maxbef\Delta$. The discretization trivially holds because σ is fine w.r.t. φ, and thus, φ does not change value during the interval.

□

LTL-min-max Discrete Encoding The discrete setting enables recursive definitions of minimum and maximum. Consider for instance the set $S^v_\top$ which stores the values of variable v at each time point. $\sigma(t)(S^v_\top)$ can be defined by the following recursive definition: $\sigma(t)(S^v_\top) = \{\sigma(t)(v)\} \cup \sigma(t+1)(S^v_\top)$. Our discrete encoding of LTL-min-max exploits this inductive structure of $U^c_{\varphi \mathbf{U} \psi}$ and Bef^c_φ to provide a sound translation to first-order LTL. Our encoding introduces new monitor variables representing $min\Delta^c_{\varphi \mathbf{U} \psi}$, $max\Delta^c_{\varphi \mathbf{U} \psi}$ and $maxbef\Delta^c_\varphi$. For each operator, a monitor replaces it in the formula, and the constraints of these monitors imply the original formula.

In the remainder of this section, we denote $\delta_c \doteq c' - c$ and $\tilde{\mathbf{U}}$ as the "strict" version of $\mathbf{U}$ operator (also expressible as $\varphi \tilde{\mathbf{U}} \psi \doteq \varphi \wedge \mathbf{X}(\varphi \mathbf{U} \psi)$ in the discrete setting).

$$\begin{aligned} \mathcal{R}epl(\Psi, min\Delta^c_{\varphi \mathbf{U} \psi}) \doteq \mathbf{G}(\varphi \mathbf{U} \psi \rightarrow \rho_{min\Delta^c_{\varphi \mathbf{U} \psi}} = \\ ite(\psi \wedge (\neg(\varphi \tilde{\mathbf{U}} \psi) \vee 0 \leq \rho'_{min\Delta^c_{\varphi \mathbf{U} \psi}} + \delta_c), 0, \rho'_{min\Delta^c_{\varphi \mathbf{U} \psi}} + \delta_c) \wedge \\ (\mathbf{F}(\psi \wedge \rho_{min\Delta^c_{\varphi \mathbf{U} \psi}} = 0))) \rightarrow \Psi \lceil min\Delta^c_{\varphi \mathbf{U} \psi} / \rho_{min\Delta^c_{\varphi \mathbf{U} \psi}} \rfloor \end{aligned}$$

The value of $\rho_{min\Delta^c_{\varphi \mathbf{U} \psi}}$ is evaluated as the minimum only when $\varphi \mathbf{U} \psi$ holds; otherwise, $min(U^c_{\varphi \mathbf{U} \psi})$ would be undefined. The *ite* expression evaluates the minimum between 0 and $\rho'_{min\Delta^c_{\varphi \mathbf{U} \psi}} + \delta_c$ $(min(\sigma(succ(t))(U^c_{\varphi \mathbf{U} \psi}))$. Finally, $F(\psi \wedge \rho_{min\Delta^c_{\varphi \mathbf{U} \psi}} = 0)$ guarantees that a minimum exists.

Theorem 6.6 *Ψ and $\mathcal{R}epl(\Psi, min\Delta^c_{\varphi \mathbf{U} \psi})$ are equisatisfiable*

Proof: We need to prove that for all σ, for all t: $\sigma(t)(min\Delta^c_{\varphi \mathbf{U} \psi}) = \sigma(t)(\rho_{min\Delta^c_{\varphi \mathbf{U} \psi}})$. We need to prove that $\sigma(t)(\rho_{\min^c_{\varphi \mathbf{U} \psi}}) = min(\sigma(t)(U^c_{\varphi \mathbf{U} \psi})) - \sigma(t)(c)$ (min is granted to exist because the set is not empty and definition 6.1 prevents the clocks from resetting below time -λ).

Let $m_t = min(\sigma(t)(U^c_{\varphi \mathbf{U} \psi}))$ and $m\delta_t = m_t - \sigma(t)(c)$. Either $m_t = \sigma(t)(c)$ or $m_t = m_{t+1}$.

If $m_t = \sigma(t)(c)$ then ψ holds and either $\sigma, t \not\models \varphi \tilde{\mathbf{U}} \psi$ or $m_t \leq m_{t+1}$.

If $m_t = m_{t+1}$ then either $\sigma, t \not\models \psi$ or $\sigma(t)(c) > m_{t+1}$. Thus, $\sigma(t)(m_t) = \sigma(t)(ite(\psi \wedge (\neg(\varphi \tilde{\mathbf{U}} \psi) \vee \sigma(t)(c) \leq m_{t+1}), c, m_{t+1})$.

We derive the formula in terms of $m\delta_i$: $m\delta_i = ite(\psi \wedge (\neg \varphi \tilde{\mathbf{U}} \psi \vee c - c \leq m\delta_{i+1} + c' - c, c - c, m\delta_{i+1} + c' - c) = ite(\psi \wedge (\varphi \tilde{\mathbf{U}} \psi \vee 0 \leq m\delta_{i+1} + \delta_c, 0, m\delta_{i+1} + \delta_c))$.

By replacing $m\delta_i$ with $\rho_{min\Delta^c_{\varphi \mathbf{U} \psi}}$ we obtain: $\rho_{min\Delta^c_{\varphi \mathbf{U} \psi}} = ite(\psi \wedge (\neg(\varphi \tilde{\mathbf{U}} \psi) \vee 0 \leq \rho'_{min\Delta^c_{\varphi \mathbf{U} \psi}} + \delta_c, 0, \rho_{min\Delta^c_{\varphi \mathbf{U} \psi}} + \delta_c))$. Finally, if $\sigma, t \models \mathbf{F}(\psi \wedge \rho_{min\Delta^c_{\varphi \mathbf{U} \psi}} = 0)$ then $\sigma(t)(min\Delta^c_{\varphi \mathbf{U} \psi}) = \sigma(t)(\rho_{min\Delta^c_{\varphi \mathbf{U} \psi}})$ □

$$\begin{aligned}
&\mathcal{R}epl(\Psi, max\Delta^c_{\varphi \mathbf{U} \psi}) \doteq \mathbf{G}((\varphi \,\mathbf{U}\, \psi) \wedge \Phi_{finU} \rightarrow \rho_{max\Delta^c_{\varphi \mathbf{U} \psi}} = \\
&ite(\psi \wedge (\neg(\varphi \tilde{\mathbf{U}} \psi) \vee 0 \geq \rho'_{max\Delta^c_{\varphi \mathbf{U} \psi}} + \delta_c), 0, \rho'_{max\Delta^c_{\varphi \mathbf{U} \psi}} + \delta_c) \wedge \\
&\qquad (\mathbf{F}(\psi \wedge \rho_{max\Delta^c_{\varphi \mathbf{U} \psi}} = 0))) \rightarrow \Psi\lceil max\Delta^c_{\varphi \mathbf{U} \psi} / \rho_{max\Delta^c_{\varphi \mathbf{U} \psi}} \rfloor
\end{aligned}$$

The encoding of $max\Delta$ is the same as the one of $min\Delta$ only flipping the sign and introducing the constraint Φ_{finU} from definition 6.11 to evaluate the monitor only if the maximum exists i.e. the formula holds finitely often.

Theorem 6.7 *Ψ and $\mathcal{R}epl(\Psi, max\Delta^c_{\varphi \mathbf{U} \psi})$ are equisatisfiable*

Proof: Identical to the proof of Theorem 6.6. □

$$\begin{aligned}
&\mathcal{R}epl(\Psi, maxbef\,\Delta^c_{\varphi}) \doteq \mathbf{G}(\mathbf{F}\varphi \rightarrow \rho_{maxbef\,\Delta^c_{\varphi}} = \\
&ite(\varphi \vee 0 \geq \rho'_{maxbef\,\Delta^c_{\varphi}}, 0, \rho'_{maxbef\,\Delta^c_{\varphi}} + \delta_c)) \rightarrow \Psi\lceil maxbef\,\Delta^c_{\varphi} / \rho_{maxbef\,\Delta^c_{\varphi}} \rfloor
\end{aligned}$$

The encoding of $maxbef\,\Delta$ evaluates the maximum of c before and including the first point in which φ holds.

Theorem 6.8 *Ψ and $\mathcal{R}epl(\Psi, maxbef\,\Delta^c_{\psi})$ are equisatisfiable*

Formula	Time in sec.	λ	ϵ	alg	valid
$G(\overline{F}^c_{\leq p}a \rightarrow F^c_{\leq p}a)$	2.81	any	any	klivenes	True
$F(c = p) \rightarrow F(((\overline{G}^c_{\leq p}a) \wedge (\overline{G}^c_{\geq p}\neg a)) \rightarrow \bot)$	3.62	any	0.4	kzeno	True
$(q \geq p) \rightarrow G((\overline{G}^c_{\leq q}a) \rightarrow (\overline{G}^c_{\leq p}a))$	0.38	any	any	klivenes	True
$G^c_{\leq p}a \rightarrow G(a \vee c > p)$	9.03	any	any	kzeno	True
$G^c_{\leq p}a \rightarrow G(a \vee c > p)$	1.09	any	any	klivenes	True
$(q \geq p) \rightarrow G((G^c_{\geq p}a) \rightarrow (G^c_{\geq q}a))$	2.22	any	any	klivenes	True
$\Phi_{exp} := q = p(2+\epsilon) + 2\lambda \wedge (G(fault \rightarrow G\neg alive) \wedge G(\overline{G}^{cl}_{\leq p}\neg alive \rightarrow (\overline{F}^{cl}_{\leq p}alarm))) \rightarrow G(fault \rightarrow F_{[0,q]}alarm)$	94.26	14.0	0.1	klivenes	True
$(G((Reset(cl1) \rightarrow cl1' = cl) \wedge (\neg Reset(cl))) \wedge GF^{cl}_{\leq q}(cl' = cl1)) \rightarrow \mathbf{G}(cl - cl1 \leq q*(1+2\epsilon/(1-\epsilon)))$	7.05	any	any	kzeno	True
$G(f \rightarrow \overline{G}^{cl1}_{\leq p}\neg alv) \wedge G(\overline{G}^{cl2}_{\leq p-4r}\neg alv \rightarrow (\overline{F}^{cl2}_{\leq p}alm))) \rightarrow G(f \rightarrow \overline{F}^{cl}_{\leq p+2r}alm)$	19.86	any	any	klivenes	True
$\mathbf{G}(\mathbf{F}^c_{\leq p}a \rightarrow \overline{\mathbf{F}}^c_{\leq p}a)$	0.27	any	any	bmc	False
$\mathbf{G}((a \vee \mathbf{X}a) \rightarrow (\mathbf{F}^c_{<0}a \wedge \mathbf{F}^c_{>0}a))$	0.18	any	any	bmc	False
Bounded Response invalid with 11 clocks	1.36	any	any	bmc	False

Table 6.1: Some MTLSK properties and their verification results.

Proof: We need to prove that for all σ, if $\sigma, t \models \mathbf{F}\varphi$, then $\sigma(t)(maxbef\Delta^c_\varphi) = \sigma(t)(\rho_{maxbef\Delta^c_\varphi})$. Since $\mathbf{F}\varphi$ is satisfied then $Bef^c_\sigma(t,\varphi)$ is a finite (σ is a discrete trace) and not empty. We prove it by induction. $\sigma(t)(\rho_{maxbef\Delta^c_\varphi} = max(Bef^c_\sigma(t+1,\varphi) - c$ Base: $\sigma, t \models \varphi$. $Bef^c_\sigma(t,\varphi) = \{\sigma(t)(c)\}$; thus, $max(Bef^c_\sigma(t,\varphi) = \sigma(t)(c)$. Thus, $\sigma(t)(maxbef\Delta^c_\varphi) = 0$. Since $\sigma, \models \varphi$, then $\sigma(t)(\rho_{\max\_bef^c_\varphi}) = 0$. (if then else).
Inductive case: Since $\sigma, t \not\models \varphi$ and $\sigma, t \models \mathbf{F}\varphi$, then $Bef^c_\sigma(t,\varphi) = \{\sigma(t)(c)\} \cup \sigma(t+1)(Bef^c_\varphi)\}$. Thus, $max(Bef^c_\sigma(t,\varphi)) = max(\sigma(t)(c), max(\sigma(t+1)(Bef^c_\varphi)))$. By induction hypothesis: $\sigma(t+1)(\rho_{maxbef\Delta^c_\varphi}) = max(\sigma(t+1)(Bef^c_\varphi))$. Thus, if $0 \geq \rho'_{maxbef\Delta^c_\varphi} + \delta_c$ then $\sigma(t)(c)$ is the max. Otherwise, $\sigma(t+1)(c)$ is the maximum (as shown by ite expr.). □

6.2.3 Implementation and Experimental Evaluation

Proof of Concept Implementation In this section, we present the implementation and experimental analysis of the procedure we described in the previous section to prove the validity and satisfiability of MTLSK formulae. We implemented MTLSK verification as an extension of timed nuXmv[87]. We used the following model-checking algorithms to verify the validity and satisfiability of the formulae: kzeno [103] in lockstep with bmc, k-liveness with ic3-ia [102] and BMC with diverging variables [98]. The algorithms used inside nuXmv are constructed on top of MathSAT5 [105] SMT-solver, which supports combinations of theories such as $\mathcal{LIA}$, $\mathcal{LRA}$ and $\mathcal{EUF}$. Unlike the logic definition of Section 6.2.1, our implementation permits the usage of formulae containing

clocks in state predicates (e.g., $\mathbf{G}(c - c1 \leq p)$ is allowed). Indeed, we introduced continuity constraints of [282] inside the discretization process to relax this syntactic restriction. Our implementation considers λ and ϵ constants from Definition 6.1 to instantiate bounds for resettable clocks. These parameters are defined inside the SMV model either as scalar values (DEFINE) or as a parameter (FROZENVAR).

Experimental Evaluation We are not aware of other tools supporting MTLSK or other variations of MTL over resettable skewed clock; therefore, our experimental evaluation is performed only on our implementation. The experiments[2] were run in parallel on a cluster with nodes with Intel Xeon CPU running at 2.40GHz with 12CPU, 64GB. The timeout for each run was one hour and the memory cap was set to 1GB. The experimental evaluation considers the scalability of the tool concerning λ, ϵ, formulae size and until bound.

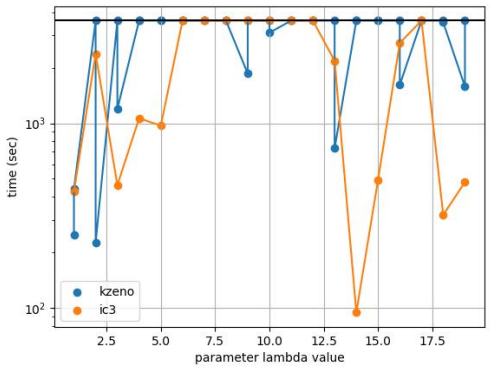
Benchmarks. Our experiment is divided into the following groups:

The first group of formulae is composed of a chain of *bounded response* (BR). Each local component i sends to its successor message a_i in at most p time unit where p is a parameter and time is interpreted as the local clock c_i. Finally, given a parametrized maximum bound r between local clocks and the global clock, the bounded response chain guarantees that $q = 2n(p+r)$ is the right bound for the global bounded response. $\mathbf{G}(\bigwedge_{0 \leq i < n} ((a_i \rightarrow \overline{\mathbf{F}}^{c_i}_{\leq p} a_{i+1}) \wedge (c - c_i \leq r \wedge c_i - c \leq r))) \rightarrow \mathbf{G}(a_0 \rightarrow \overline{\mathbf{F}}^{c}_{\leq q} a_n)$

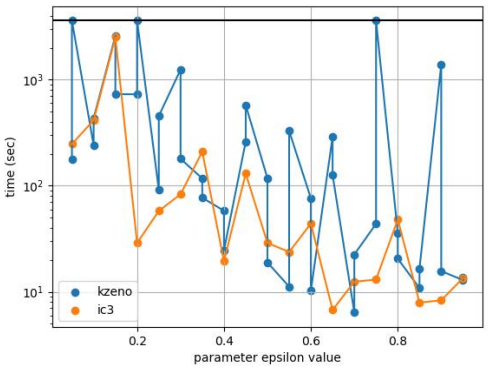
The second group of formulae is a Fischer algorithm benchmark taken from [98] MTL experimental evaluation and adapted for MTLSK. The pool of formulae is parametrized over the number of components. Each component has its skewed clock and a local formula representing its behaviour, the conjunction of all the formulae is used to imply a property of the first component.

The third group of formulae is a variation of the running example proposed at the beginning of this Chapter that instantiates the parameters p, λ and ϵ to study their impact on the validity checking performance. We also considered a timed and simplified version of a Wheel Brake System model of [132]. Given a redundant signal of the brake pedal, a redundant braking component, the property states that the hydraulic system should brake before a specified time threshold if the redundant components do not fail at the same time. This property has been translated to MTLSK and each component has been augmented with a local clock.

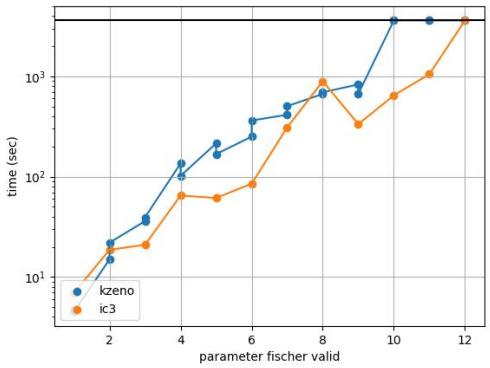
[2]The results of the experimental evaluation can be found at `https://es-static.fbk.eu/people/bombardelli/papers/nfm23/nfm23.tar.gz`



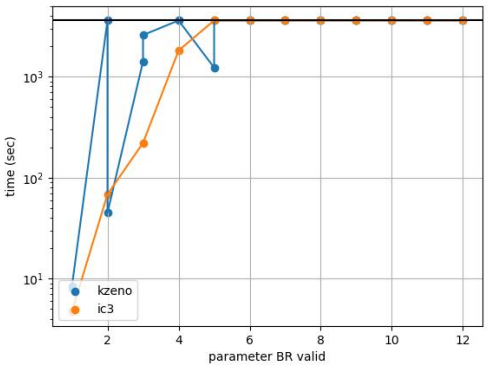
(a) λ evaluation



(b) ϵ evaluation



(c) Fischer experimental evaluation



(d) BR experimental evaluation

Figure 6.4: Parametric experimental evaluations

The last group is a collection of roughly 100 MTLSK specifications, 60 valid and 40 invalid, defined to validate the semantics and the implementation.

Experimental Results. Table 6.1 shows the results on a subset of formulae. The solver proves most of the tautologies of group 4 in less than 10 seconds and 38 properties were proved in less than 2 seconds. All the invalid properties of this group were disproved in less than 2 seconds with the BMC algorithm. Moreover, we performed an experimental evaluation using the running example of this chapter. We were able to prove the example by splitting the property into two parts. First, we proved that q is an upper bound for the maximum distances between c and $c1$ if $c1$ is synchronized to c every r time unit. Second, we proved that assuming a maximum distance between c and the other clocks the property holds.

The experiments on λ and ϵ pointed out the instability brought by instantiating these parameters to an arbitrary value. In particular, Figure 6.4a and Figure 6.4b

show the impact of the two constants increasing their values. The plots show a strong instability with jumps in execution time with both λ and ϵ.

Figure 6.4c shows the experimental evaluation of the Fischer algorithm formula. The k-liveness algorithm proves the correctness of the formula with 11 components in less than one hour. Figure 6.4d shows the Bounded Response experimental evaluation. No algorithm can prove the formula with 6 components. Indeed, this model is challenging because to prove that the global component sees a_n before q time units we need to ensure that each component skewed does not delay too much the response. Moreover, this model is parametrized over λ, ϵ, p and r; thus, the solver needs to explore a larger state space.

Overall, the results show that MTLSK formulas can be verified within a reasonable amount of time. However, scalability becomes an issue as the number of components increases or when the parameters λ, ϵ, and the timing intervals are left symbolic. This behaviour is expected, since each component introduces its own clock variable and corresponding timing constraints, which together increase the complexity of the verification process. Nevertheless, the experiments demonstrate that the proposed approach remains practical for moderately sized systems and provides a solid foundation for further optimization and integration into larger verification workflows.

6.2.4 Applying Compositional Reasoning to DRTS using MTLSK

In Section 6.1, we defined a compositional reasoning approach for skewed clocks system. The idea in that case is that, one can locally check a MTL property with standard semantics and, afterwards, use satisfiability to check the compositional proof obligation. However, with resettable clocks, this is no longer possible. Although we might use clocks with $\epsilon = 0$, we still require the clock to be resettable. Therefore, locally our system need to be verified with a non-monotonic time semantics which requires using MTLSK as a target logic.

Model Checking MTLSK to Validity To apply model checking locally, we can simply reduce the model checking problem of MTLSK to the validity problem following a similar approach to the one presented in Chapter 4 for LTL($\mathcal{T}$) [3].

Intuitively, transitions can be encoded by the following formula:

$$\phi_T \doteq \mathbf{G}(\mathbf{X}\top \rightarrow T)$$

[3]Note that the fragment presented in Section 6.2.2 does not support the encoding of *Inv* and $\mathcal{SF}$ in case it contains unguarded clocks.

Clock invariants constraints, which must be enforced in timed transitions as well, are included in a generic globally constraint:

$$\phi_{Inv} \doteq \mathbf{G}\mathit{Inv}$$

Finally, fairness constraints are parsed to the following liveness constraint:

$$\phi_{\mathcal{SF}} \doteq \bigwedge_{\langle a_i, g_i \rangle \in \mathcal{SF}} (\mathbf{GF}a_i \rightarrow \mathbf{GF}g_i)$$

Computing γ_P for MTLSK We can straightforwardly construct the formula composition function by extending the one presented for distributed MTL as follows:

$$\mathcal{R}^{\tau}(\varphi_1 \overline{\mathbf{U}}^{c}_{\mathcal{I}} \varphi_2) \doteq ((\neg \mathit{run} \wedge \mathbf{X}\top) \vee \mathcal{R}^{\tau}(\varphi_1)) \overline{\mathbf{U}}^{c}_{\mathcal{I}} ((\mathit{run} \vee \tilde{\mathbf{X}}\top) \wedge \mathcal{R}^{\tau}(\varphi_2))$$

Finally, putting all together, we can construct γ_P with a complete compositional approach as expressed in Inference 5.1.

6.3 Related Works

This section compares our contributions with related works. A complete overview of the state-of-the-art can be found in Chapter 3.

This work is built on top of [59, 62], also presented in Chapter 5. Other compositional frameworks can deal with timed systems e.g. BIP [23] and OCRA [92] (in timed mode). Other work related to our compositional proof systems are discussed in Chapter 5, in particular in Section 5.4. As far as we know, there are no other techniques for the compositional verification of temporal specification with resettable skewed clocks.

Various works customized the modal operators of temporal logics to better suit the specification of DRTS. TPTL was extended in [291] by using explicitly multiple local clocks and supporting inequalities to express constraints on the precedence between local clock readings. In [253], a distributed variant of ECTL is proposed. Similarly, [252] defines a distributed modal logic where the time varies independently in each component of the system, represented by a network of timed automata. In all these works, local times are assumed strictly increasing, and thus do not address the semantic issues that arise when modeling time using non-monotonic local clocks, as we do in this chapter.

The problem of modelling DRTS with drifting and synchronized clocks was considered in [265], where specific patterns of timed automata were proposed and verified. Similarly, [79] considers the problem of parametrized verification of 8N1 and Biphase Mark protocols with skewed clocks using the SAL model checker[140]. These works focus on the modelling of clock drifts and synchronizations, but do not consider the specification of timed properties that refer to skewed synchronized clocks.

The problem of validating the correctness of drifted clocks synchronization algorithms have been studied in other works using theorem provers, e.g., [268, 19].

The work closer to the problem addressed in this chapter is focused on the satisfiability of MTL and TPTL over non-monotonic time, which has been studied in [86] in the more general context of data words. Timed words are considered a special case, although [86] considers only discrete time. A decision procedure is given for the fragment without negation and only temporal operators $\mathbf{X}$ and $\mathbf{F}$. Instead, we address an undecidable fragment of MTLSK with SMT-based model checking.

Last, we mention [172], which focuses on runtime verification of MTL formulae in a distributed system. Here, the authors address the problem of monitoring global properties on all traces that are compatible with a given sequence of local observations with timestamps taking into account the possible drift of local clocks. Thus, the metric operators are not, as in our case, used in local properties and related to local clocks.

Part IV

Hyperproperties for Comparing Traces

Chapter 7

HyperLTL Model Checking with HyperK-Induction

In the previous chapters, we focused on verifying properties over single system executions, such as invariants and LTL specifications over rich state spaces. These properties are fundamental for ensuring the correctness of individual behaviors and capture essential notions such as safety and liveness. However, many correctness and security requirements of modern systems cannot be verified by examining single executions in isolation. They depend on how different executions relate to one another–for instance, whether a program produces indistinguishable outputs when given inputs that differ only in secret data, or whether concurrent processes remain consistent under varying schedules. They depend on how different executions relate to one another–for instance, whether a program produces indistinguishable outputs when given inputs that differ only in secret data, or whether concurrent processes remain consistent under varying schedules. Capturing and verifying such relational behaviours requires specification languages that can reason about multiple executions at once. Hyperproperties are a class of system specifications that describe system properties that require relating multiple execution traces [122]. Hyperproperties are common in information-flow security [186, 298], robustness and sensitivity of cyber-physical systems [293], concurrency [63] and symmetry [162]. HyperLTL [120] is a temporal logic for hyperproperties that equips LTL with quantification over traces. HyperLTL includes relational capabilities to compare different traces and temporal modalities where the temporal operators move all traces synchronously. Several algorithms and verification methods for HyperLTL have been proposed [162, 125, 199], particularly for $\forall^*.\varphi$ and $\exists^*.\varphi$ hyperproperties (that is, hyperproperties without quantifier alternations).

A particularly expressive and practically useful subclass of hyperproperties are those of the form $\forall\exists.\varphi$, where for every trace, there exists another trace such that a property φ holds relating corresponding states in both traces. When φ is restricted to *invariants* or *SafetyLTL* formulae, this class can express many interesting properties of interest, including noninterference, symmetry, and robust path planning. These $\forall\exists$-hyperproperties are widely applicable, yet remain challenging to verify in practice. Current verification techniques include automata-based approaches [162] and bounded model checking for HyperLTL [199].

As discussed in Chapter 1, scalability gets significantly exacerbated when we consider expressive formalisms. More expressive logics allow richer specifications, but often at the cost of tractability and automation. For HyperLTL, this trade-off is especially evident–existing verification procedures, such as automata-based and symbolic approaches, face severe scalability challenges when applied to properties with alternating quantifiers. These limitations motivate the search for complementary reasoning techniques that can handle complex relational properties without exhaustively exploring all trace combinations.

In this chapter, we propose a new approach to verify $\forall\exists$-hyperproperties where φ is either an invariant or a SafetyLTL formula. When φ is a SafetyLTL formula, we first adapt the automata-based construction presented in Chapter 4 to reduce it to an invariant, providing a practical foundation for the verification process. Building on this, the central contribution of our method is the adaptation of induction and k-induction techniques [272], widely used in trace-based verification, to the hyperproperty setting. This requires reasoning over sets of traces and lifting classical inductive principles to operate on these sets, allowing us to tackle relational properties while maintaining a symbolic approach that can scale to larger systems. We have implemented this framework and evaluated it on representative benchmarks, demonstrating its practical potential and complementarity with existing HyperLTL model checkers.

Contributions Our contributions are as follows:

- We define inductive and k-inductive proof rules for $\forall\exists$-hyperproperties, extending base and step reasoning to trace sets. As a byproduct, we introduce hyper-(k)-inductive invariants, as certificates to witness satisfaction of $\forall\exists$-hyperproperties over transition systems.
- We define a refinement that strengthens the given $\forall\exists$ specification to make it inductive, so that the property can be proved using our generalized rules.

- We introduce incremental strategies to increase the bound of k-induction learning from previous checks.
- We implement our approach and evaluate it on a range of benchmarks from security, distributed systems, and motion planning. Our results show that our technique is complementary to existing model checkers for HyperLTL, and often scales significantly better than explicit-state techniques when refinement is able to find "hyper-k-inductive invariant" in a reasonable amount of iterations.

The contribution presented in this Chapter has not been published yet. This chapter is the result of a collaborative work with César Sánchez and Stefano Tonetta.

Running Example In the remainder of the chapter, we will introduce the proof obligations of our k-inductive approach using variations of this simple program representing non-interference settings. More general variations of such example will be also considered in the experimental evaluation (see Figure 7.1 and Figure 7.2). The kind of property we will consider is a non-interference property introduced in the following example.

Example 7.1 *Let* $M_{bp} := \langle \{ch, o, h, p\}, \neg ch, (ch' \leftrightarrow \neg ch) \wedge (ch \rightarrow (o' \leftrightarrow (h \leftrightarrow p)))\rangle$ *be the STS representing a Boolean program.* M_{bp} *represents a system with 2 "modalities", one in which it chooses a value for* o *(*$ch = \top$*) depending on* h *and* p*, and the other where* o *can take any possible value. The forall-exists Hyperproperty that we are interested in proving is* $\phi_{bp} \doteq \forall x \exists y. \boldsymbol{G}((o[x] \leftrightarrow o[y]) \wedge (h[x] \leftrightarrow \neg h[y]))$*. The property – representing non-interference – states that for each execution of the system providing and a value for the output* o*, there is another execution that outputs the same value of* o *but with a different value for the secret variable* h*.*

Outline This chapter is organized as follows. In Section 7.1, we present the techniques of induction and k-induction for hyperproperties; in Section 7.2, we present an experimental evaluation between our implementation and state-of-the-art tools; finally, in Section 7.3, we provide a comparison of the approach presented in this chapter with respect to other relevant publications.

7.1 Induction and K-Induction for Hyperproperties

Induction and k-induction are two well-known techniques that are used to prove symbolically invariant (trace) properties. In this section we present two novel techniques

constructed adapting induction, K-induction and the notion of generalization of counterexamples to induction for hyperproperties. We showcase our techniques with examples, and we demonstrate their soundness and completeness.

In the rest of the chapter, we assume M is a deadlock-free STS, and the property under analysis to be $\varphi := \forall x.\exists y.\mathbf{G}\psi$ where ψ is a state predicate over $V[x]$ and $V[y]$. Although the theorems presented in this section are proved over $\forall x.\exists y.\mathbf{G}\psi$ properties, the results can be easily extended to $\forall^*\exists^*.\mathbf{G}\psi$ hyperpropeties. Similarly, all results can be easily extended for ψ being a SafetyLTL formula.

Outline This Section is organized as follows. Section 7.1.1 presents Induction adapted for Hyperproperties showing soundness of the algorithm; Section 7.1.2 presents k-induction adapted for Hyperproperties showing soundness and completeness; Section 7.1.3 presents refinement techniques to generalize counterexample to induction for our algorithms; finally, Section 7.1.4 presents an incremental encoding of the k-induction algorithm to be used with incremental solvers.

7.1.1 Induction

The induction principle for proving an invariant trace property ψ consists of finding a property P such that the following three checks hold: (i) the property P holds at all initial states; (ii) anytime that P holds, each successor state also satisfies P; (iii) P implies the original invariant property ψ. The adaptation of induction for hyperproperties must take into account quantifier alternation and the self-product of the system.

The base case $BASE^{\forall\exists}$ is defined as follows:

$$BASE^{\forall\exists} := \forall V[x].\exists V[y].I(V[x]) \rightarrow I(V[y]) \wedge P(V[x], V[y])$$

$BASE^{\forall\exists}$ checks whether for each initial state of M (instantiated with x trace variable), there is an initial state of M (instantiated with y trace variable) such that $P(V[x], V[y])$ holds.

Lemma 7.1 *Let $P := \psi$, if $\not\models BASE^{\forall\exists}$ then $M \not\models \forall x\exists y.\boldsymbol{G}\psi$*

Proof: By assumption $\not\models BASE^{\forall\exists}$, which means that there is an initial state s of M such that for any initial state s' of M s.t. together they violate P. Since M has no deadlock there exists an infinite trace σ that extends s ($\sigma(0) = s$). Suppose by contradiction that $M \models \forall x\exists y.\mathbf{G}\psi$. Then there exists infinite trace σ' such that the trace assignment

Input: A STS $M = \langle V, I, T \rangle$ and a formula $\varphi := \forall x.\exists y.\mathbf{G}\psi$
1 $\varphi := \psi$;
2 **if** $BASE^{\forall\exists}$ *does not hold* **then return** INVALID
3 **else if** $IND^{\forall\exists}$ *holds* **then return** VALID
4 **else return** *UNKNOWN*

Algorithm 2: Induction algorithm

Π that assigns σ to x and σ' to y, is such that $\Pi \models \mathbf{G}\psi$, and consequently the initial states $\sigma(0)$ and $\sigma'(0)$ satisfy P, which is a contradiction. Thus, $M \not\models \forall x\exists y.\mathbf{G}\psi$. □

We define the inductive case $IND^{\forall\exists}$ as follows:

$$IND^{\forall\exists} := \forall V[x]V[x]'V[y].\exists V[y]'.\ \ T(V[x], V[x]') \wedge P(V[x], V[y]) \rightarrow T(V[y], V[y]') \wedge P(V[x]', V[y]')$$

$IND^{\forall\exists}$ universally quantifies over states of the self-product of M that satisfy P, and checks that for each successor of x there is a successor for y such that the property still holds. Interestingly, this property is much stronger that the target $\forall x.\exists y.\mathbf{G}\psi$ property because we are requiring each state of the self-product to satisfy this property.

Lemma 7.2 *Let $P := \psi$, if $\models BASE^{\forall\exists}$ and $\models IND^{\forall\exists}$ then $M \models \forall x\exists y.\boldsymbol{G}\psi$*

Proof: We prove the lemma constructively by providing a trace σ' for each trace σ, defined as follows:

- $(i = 0)$: Choose $\sigma'(0)$ to be any initial state such that the pair $\sigma(0), \sigma'(0)$ satisfies ψ. The existence of such a state is guaranteed by the assumption $\models BASE^{\forall\exists}$ and the fact that the system is deadlock-free.
- Inductive case $(i > 0)$: Suppose we have already constructed $\sigma'(i-1)$ such that $\sigma(i-1), \sigma'(i-1)$ satisfies P. Since $\sigma(i)$ is a successor of $\sigma(i-1)$ (which exists because M is deadlock-free), and $IND^{\forall\exists}$ holds, there exists a successor $\sigma'(i)$ of $\sigma'(i-1)$ such that the pair $\sigma(i), \sigma'(i)$ satisfies P, and hence also satisfies ψ.

□

These proof obligations are combined to obtain Algorithm 2 to verify $\forall x\exists y.\mathbf{G}\psi$ hyperproperties. The following theorem follows directly from Lemma 7.1 and Lemma 7.2.

Theorem 7.1 *Algorithm 2 is sound.*

Example 7.2 *Consider a variation of Ex. 7.1, where we highlight the new parts:*

$$\varphi_{ex1} := \forall x \exists y. \boldsymbol{G}((o[x] \leftrightarrow o[y]) \wedge (h[x] \leftrightarrow \neg h[y]) \wedge (\mathbf{p[x]} \leftrightarrow \neg \mathbf{p[y]}) \wedge (\mathbf{ch[x]} \leftrightarrow \mathbf{ch[y]}))$$

This example property is satisfied by M_{bp}. The variable ch is true only at odd positions (counting from 0) of the trace and, choosing $p[y]$ as the negation of $p[x]$ ensures that $o[y]$ is set to the same value of $o[x]$ when ch is true. The proof obligations instantiated for the properties are as follows:

$$BASE^{\forall\exists} : \forall Vbp[x] \exists Vbp[y]. \neg ch[x] \rightarrow \neg ch[y] \wedge \begin{pmatrix} (o[x] \leftrightarrow o[y]) \wedge (h[x] \leftrightarrow \neg h[y]) \wedge \\ (p[x] \leftrightarrow \neg p[y]) \wedge (ch[x] \leftrightarrow ch[y]) \end{pmatrix}$$

$$IND^{\forall\exists} : \forall Vbp[x] Vbp[x]' Vbp[y] \exists Vbp[y]'. \quad \begin{pmatrix} (o[x] \leftrightarrow o[y]) \wedge (h[x] \leftrightarrow \neg h[y]) \wedge \\ (p[x] \leftrightarrow \neg p[y]) \wedge (ch[x] \leftrightarrow ch[y]) \wedge \\ T(Vbp[x], V_{ex}[x]') \end{pmatrix} \rightarrow \begin{pmatrix} ((o[x]' \leftrightarrow o[y']) \wedge (h[x]' \leftrightarrow \neg h[y'])) \wedge \\ ((p[x]' \leftrightarrow \neg p[y']) \wedge (ch[x]' \leftrightarrow ch[y'])) \wedge \\ T(Vbp[y], V_{ex}[y'])) \end{pmatrix}$$

$BASE^{\forall\exists}$ is valid because it is sufficient for y to choose $\neg ch$ and then simply assign the remaining variables satisfying φ_{ex1}. $IND^{\forall\exists}$ is valid as well. If $ch[x]$ is false, then it is possible to select $Vbp[y]'$ satisfying φ_{ex1}. Otherwise, since $p[y] = \neg p[x]$ and $h[y] = h[x]$ then $h[y] = p[y]$ iff $h[x] = p[x]$ which means that $o[x]' = o[y]'$.

It should be noted that if we remove $p[x] \leftrightarrow \neg p[y]$ from φ_{ex1}, the property (denoted as φ'_{ex1}) is no longer inductive and $IND^{\forall\exists}$ is not satisfied. A counterexample to induction for that weakened property would any assignment over $Vbp[x]$,$Vbp[x]'$ and $Vbp[y]$ such that $ch[x] = ch[y]\top$ and $p[x] = p[y]$.

7.1.2 K-Induction for Hyperproperties

K-induction is a well-known technique for the verification of invariant trace properties. This technique extends induction by reasoning over paths of length k. The base case of the k-induction algorithm corresponds to bounded model checking (BMC), while the inductive case checks whether unrolling the transition system for k steps, together with the property over these steps, entails the property at step $k + 1$.

We adapt the proof obligations for k-induction to hyperproperties. The proof obligation concerning BMC is a special case of the more generic encoding introduced in [199].

$$BMC_k^{\forall\exists} := \forall V[x]_0 \ldots V[x]_k \exists V[y]_0 \ldots V[y]_k . I(V[x]_0) \wedge \mathit{Unroll}_k(\overline{V}[x]) \rightarrow I(V[x]_0) \wedge \mathit{Unroll}_k(\overline{V}[y]) \wedge \bigwedge_{0 \leq i \leq k} P(V[x]_i, V[y]_i)$$

Lemma 7.3 *Let $P := \psi$. There is a $k \geq 0$ such that $BMC_k^{\forall\exists}$ does not hold iff $M \not\models \forall x \exists y. \boldsymbol{G}\psi$*

Proof: ($\Rightarrow$) The proof is the same as for Lemma 7.1 but with a longer path. ($\Leftarrow$) Suppose that $M \not\models \forall x \exists y \mathbf{G}\psi$. Then there is a trace σ and $j \in \mathbb{N}$ such for any trace σ' of M $\Pi \not\models \psi$ where Π is the trace assignment assigning x to the pointed trace (σ, j) and y to (σ', j). By assumption $BMC_j^{\forall\exists}$ holds, which means that for each path of M of length j there is a path of length j such that the product of the two paths always satisfies ψ. Then, if we consider the finite path build from σ stopping at point j there must exist a finite path such that the composition of the two paths satisfies ψ. However, since by assumption M is deadlock-free, then the "existential" path is extendible to an infinite trace, which contradicts $M \not\models \forall x \exists y. \mathbf{G}\psi$. □

The proof obligation $KIND_k^{\forall\exists}$ for the k-inductive case extends $IND^{\forall\exists}$ with the universally quantified unrolling of the self-product of M in conjunction of the property at each position $i < k$:

$$KIND_k^{\forall\exists} := \forall V[x]_0 \ldots \forall V[x]_k \forall V[y]_0 \ldots \forall V[y]_{k-1} \exists V[y]_k . \mathit{Unroll}_k(\overline{V}[x]) \wedge \mathit{Unroll}_k(\overline{V}[y]) \wedge \bigwedge_{0 \leq i < k} P(V[x]_i, V[y]_i) \rightarrow T(V[y]_{k-1}, V[y]_k) \wedge P(V[x]_k, V[y]_k)$$

Example 7.3 *Consider the following variation of Example 7.1, which models a finite program with a bounded counter c ranging from 0 to some $j \in \mathbb{N}$. After exactly j steps, the output o is set to $h \leftrightarrow p$, and the system stutters.*

$$M'_{bp} \doteq \langle \{o, h, p, c\}, c = 0, (c < j \rightarrow c' = c + 1) \wedge (c = j \rightarrow c' = j \wedge o' \leftrightarrow (h \leftrightarrow p)) \rangle$$

and the formula $\phi'_{bp} \doteq \forall x \exists y. \boldsymbol{G}((o[x] \leftrightarrow o[y]) \wedge (h[x] \leftrightarrow \neg h[y]) \wedge (\mathbf{p}[\mathbf{x}] \leftrightarrow \neg \mathbf{p}[\mathbf{y}]))$ obtained removing the $ch[x] \leftrightarrow ch[y]$ from ϕ_{bp} (see Example 7.2).

This example is not inductive because the property is not strong enough to prove the system. The counters of the universally quantified self-product can have different values, and if $c[x] < j$ and $c[y] = j$, the inductive proof obligation fails, as demonstrated by the following counterexample to induction (CTI), where j is instantiated to 6: $(c[x] = 3, o[x], h[x], \neg p[x])$, $(c[x]' = 4, o[x]', h[x]', \neg p[x]')$, $(c[y] = 6, o[x], \neg h[y], p[y])$. Intuitively, there is no successor for y that sets o to true, because $h \neq p$. On the other hand, it is easy to see that this example is j-inductive, since any path over the self-product of length j will reach a configuration in which both $c[x]$ and $c[y]$ equal j.

$$\widehat{KIND}_k^{\forall\exists} := \forall V[x]_0 \ldots \forall V[x]_k \forall V[y]_0 \ldots \forall V[y]_{k-1} \exists V[y]_k . (\mathit{Unroll}_k(\overline{V}[x]) \wedge \mathit{Unroll}_{k-1}(\overline{V}[y]) \wedge \bigwedge_{0 \leq i < k} P(V[x]_i, V[y]_i) \wedge \bigwedge_{\mathbf{0 \leq i < j < k}} (\mathit{Diff}(\mathbf{V[x]} \cup \mathbf{V[y]}, \mathbf{i}, \mathbf{j}))) \rightarrow T(V[y]_{k-1}, V[y]_k) \wedge P(V[x]_k, V[y]_k)$$

where $\mathit{Diff}(S, i, j) := \neg \bigwedge_{s \in S}(s_i = s_j)$.

$\widehat{KIND}_k^{\forall\exists}$ extends $KIND_k^{\forall\exists}$ by adding a constraint that prevents state repetition across the self-composition (over $V[x]$ and $V[y]$). This constraint, commonly referred to as the simple path condition in classical k-induction, ensures that the paths considered are loop-free. Enforcing this condition is essential for guaranteeing the completeness of the k-inductive proof. Consider for instance Example 7.1. For any possible $k \in \mathbb{N}$ the proof obligation $KIND_k^{\forall\exists}$ would fail because any path starting with $ch[x] \neq ch[y]$ would lead to a CTI; this occurs because y is forced to pick a specific value for $o[x]_k$ if ch is $\top$ while $ch[x]_k$ can be both true or false. In this specific case, the simple path constraint will eventually block all the undesirable configurations.

Lemma 7.4 *Let M be an STS, ψ be $P(x, y)$ and let $\alpha := \forall x.\exists y.\boldsymbol{G}\psi$. If there is a $k > 0$ such that $BMC_{k-1}^{\forall\exists}$ and $\widehat{KIND}_k^{\forall\exists}$ hold, then $M \models \alpha$*

Proof: Since $BMC_{k-1}^{\forall\exists}$ holds, then for each path $V[x]_0, \ldots, V[x]_{k-1}$ there is a path $V[y]_0, \ldots, V[y]_{k-1}$ such that $P(V[x]_i, V[y]_i)$ holds for each i. Consider all paths over $V[x], V[y]$ such that $P(V[x]_i, V[y]_i)$ holds for each i. We can partition the set of paths into G_1, which contains cycles, and G_2 which does not contain cycles. Since these paths are induced by $BMC_k^{\forall\exists}$ the $V[x]$ part of $G_1 \cup G_2$ is exactly the set of all paths of length k. Since $\widehat{KIND}_k^{\forall\exists}$ holds, each path of G_2 extended with any $V[x]_k$ has a $V[y]_k$ such that $P(V[x]_k, V[y]_k)$ still holds.

Let $V[x]_0, V[x]_{k-1}$, $V[y]_0, \ldots, V[y]_{k-1} \in G_1$. Since there is at least a cycle, we can compress the path obtaining: $\widehat{V[x]}_0, \ldots, \widehat{V[x]}_l$, $\widehat{V[y]}_0, \ldots, \widehat{V[y]}_l$ with $l < k-1$. Then if we pick any $V[x]_k = \widehat{V[x]}_{l+1}$ s.t. $T(\widehat{V[x]}_l, \widehat{V[x]}_{l+1})$ by $\models BMC_k^{\forall\exists}$ there is an assignment $V[y]_0, \ldots, V[y]_{l+1}$ for the compressed path satisfying always P from which we deduce that the response exists for the original path as well. □

Lemma 7.5 *Let $P := \psi$. If M is a finite state STS and $M \models \forall x \exists y. \boldsymbol{G}\psi$, then there exists a $k > 0$ such that both $BMC_{k-1}^{\forall\exists}$ and $\widehat{KIND}_k^{\forall\exists}$ hold.*

Proof: We proceed by contradiction. Suppose that for all $k > 0$ either $\not\models BMC_{k-1}^{\forall\exists}$ or $\not\models \widehat{KIND}_k^{\forall\exists}$. Since M satisfies the property, then for all k' : $BMC_{k'}^{\forall\exists}$. Therefore, $\widehat{KIND}_k^{\forall\exists}$ must be violated at each k. Let d be the diameter of $M \times M$ (i.e. the amount of states of $M \times M$). It follows that there are not loop-free paths longer than d in $M \times M$. Then there are no loop free paths over $V[x], V[y]$ of length $d+1$ contradicting that $\widehat{KIND}_{d+1}^{\forall\exists}$ does not hold. □

The following theorem follows directly from Lemma 7.3, Lemma 7.4 and Lemma 7.5.

Theorem 7.2 *Algorithm 3 is sound and complete.*

We can generalize the result for $\forall x \exists y. \mathbf{G}\psi$ to any $\forall\exists$ temporal-safety formula exploiting the automata construction presented in Chapter 4 for LTL($\mathcal{T}$) [1]. As for the construction in Chapter 4, we consider an abstract SafetyLTL formula $\hat{\psi}$ constructed replacing all predicates in ψ with new Boolean fresh variables. Subsequently, we can construct the composed STS $M \times M^l_{\hat{\psi}}$ over the variables $V \cup \{v_p | p \in Pred(\psi)\}$. Then, to relate the

[1] Note that we are using the liveness construction of the SafetyLTL formula and not its negation – as instead done in classical automata-based approach

Input: A STS $M = \langle V, I, T \rangle$ and a formula $\varphi := \forall x. \exists y. \mathbf{G}\psi$

```
φ := ψ; i := 0;
while true do
    if BMC_i^{∀∃} does not hold then return INVALID
    i := i + 1;
    if KIND_i^{∀∃} holds then return VALID
end
```

Algorithm 3: K-induction algorithm

fresh variables to the predicates of the quantifier-alternation relational formula we introduce a globally temporal operator to relate the fresh variables (quantified existentially) with the predicates.

Theorem 7.3 *Let $\forall x \exists y.\psi$ be a temporal-safety HyperLTL formula, let M be a deadlock-free STS. Then, $M \models \forall x.\exists y.\psi$ if and only if*

$$M \times M_{\widehat{\psi}} \models \forall x.\exists y.\boldsymbol{G}\left(\bigwedge_{p \in Pred(\psi)} \left(p[y] \leftrightarrow p(x,y)\right)\right)$$

where $\widehat{\psi}$ is the LTL formula over the symbols $Pred(\psi)$ obtained replacing each top-level predicate p over x y with a fresh variable v_p; $M_{\widehat{\psi}}$ is the STS obtained through the symbolic automata construction for LTL of Section 2.5.4.

Proof: (Sketch) Intuitively the conjunction over predicates of ψ to the existential variables of $M_{\widehat{\psi}}$. Since $\widehat{\psi}$ is in safetyLTL, the automata construction does not involve fairness constraints because **U** occurs negatively in the *ltl2sts* construction. Therefore, bounding the predicates of ψ with the variables of $\widehat{\psi}$ is sufficient to have an equivalent model checking problem. □

7.1.3 Refinement Counterexamples To Induction

In classical invariant verification a *counterexample to induction* (CTI) is a pair of assignments $(X^{cex}, X^{cex\prime})$ over V, V' such that $P(X^{cex})$ and $T(X^{cex}, X^{cex\prime})$ holds but $P(X^{cex\prime})$ is false.

In our case, we define a CTI as a triple of assignments $(X^{cex}, X^{cex\prime}, Y^{cex})$ over $V[x], V[x]'$ and $V[y]$ such that $P(X^{cex}, Y^{cex})$ and $T(X^{cex}, X^{cex\prime})$ hold, and no successor $Y^{cex\prime}$ of Y^{cex} exists such that $P(X^{cex\prime}, Y^{cex\prime})$ holds. Ex. 7.2 shows an example of a counterexample to induction for property $\varphi_{ex1'}$. In this section, our aim is to find a strengthening of the formula derived by counterexamples to induction to either prove or disprove the property.

Definition 7.1 *Let ψ, ψ' be two state predicates over $V[x], V[y]$. We say that ψ' is a strengthening of ψ (over M) whenever 1) $\psi' \rightarrow \psi$, 2) $\psi \not\rightarrow \psi'$, and 3) $M \models \forall x \exists y.\boldsymbol{G}\psi$ if and only if $M \models \forall x \exists y.\boldsymbol{G}\psi'$.*

Algorithm 4 generalizes Alg.2 using function *genCTI* that generates CTIs.

Input: A STS $M = \langle V, I, T \rangle$ and a formula $\varphi := \forall x.\exists y.\mathbf{G}\psi$
1 $P := \psi$;
2 **while** *true* **do**
3 | **if** $\not\models BASE^{\forall\exists}$ **then return** INVALID
4 | **else if** $\models IND^{\forall\exists}$ **then return** VALID
5 | **else** $P := genCTI(\varphi, (X^{cex}, Y^{cex}, X^{cex\prime}))$
6 **end**

Algorithm 4: Induction algorithm with generalization function *genCTI*.

Theorem 7.4 *If genCTI returns a strengthening of P over M, then Algorithm 4 is sound and complete.*

Proof: The soundness follows from Theorem 7.1 and by the fact that *genCTI* returns the strengthening of P (and indirectly of ψ).

Completeness follows from the fact that M is finite state. Suppose that the algorithm does not terminate, then at some point P will block all the states of the self-product of M. Therefore, $BASE^{\forall\exists}$ and $IND^{\forall\exists}$ will become vacuously true contradicting the assumption that the algorithm does not terminate. □

A simple way to generalize a counterexample to induction to get a strengthening of P is to block the assignments X^{cex} and Y^{cex} of $V[x]$ and $V[y]$, as shown in Algorithm 5. This naive generalization blocks a state over the self-composition of M. More efficient techniques aim to block larger portions of the state space of the self-product. Algorithm 6 shows a generalization technique based on Craig interpolation and quantifier elimination.

Theorem 7.5 *Both Algorithm 5 and Algorithm 6 return a strengthening of P over M.*

Proof: [Sketch]

- (naive) The idea is that the tuple (X^{cex}, Y^{cex}) represents a pair of states such that there exists a successor of x for which no corresponding successor of y satisfies ψ. If M satisfies the original property, then either X^{cex} is unreachable, or for every

Input: A CTI $X^{cex}, X^{cex\prime}, Y^{cex}$, a STS M and $\varphi(V[x], V[y])$
Output: A strengthening of φ w.r.t. M
1 **return** $\varphi \wedge \neg(X^{cex} \wedge Y^{cex})$

Algorithm 5: A naive algorithm to strengthen φ using CTI.

successor of X^{cex}, there exists an alternative assignment $\overline{Y^{cex}}$ such that the successor of y satisfies ψ. This condition ensures that blocking the pair (X^{cex}, Y^{cex}) does not erroneously eliminate a behaviour that is required for the property to hold.

- (itp) The interpolation-based approach follows a similar intuition. It starts from the triple $(X^{cex}, X^{cex\prime}, Y^{cex})$ and computes, via interpolation, a formula $Q(V[x], V[x]', V[y])$ that captures the conflict between the two paths: there is no $V[y]'$ such that $\psi(V[x]', V[y]')$ holds. To eliminate $V[x]'$, we compute the preimage of Q using quantifier elimination. As in the naive case, the result is used to strengthen P in a way that excludes problematic behaviours without violating the original property.

□

The complete proof is as follows:

Proof:

naive: The idea is that the tuple (X^{cex}, Y^{cex}) represents a pair of states such that there exists a successor of x for which no corresponding successor of y satisfies ψ. If M satisfies the original property, then either X^{cex} is unreachable, or for every successor of X^{cex}, there exists an alternative assignment $\overline{Y^{cex}}$ such that the successor of y satisfies ψ. It is easy to see by contradiction: There is a trace σ, $i \in \mathbb{N}$ such that for each trace σ' $\sigma(i)$ assigns the same values as X^{cex} and $\sigma'(i)$ assigns the same value as Y^{cex}. The trace assignment Π_i (obtained by mapping x to (σ, i) and y to (σ', i)) satisfies ψ and, since the original property is satisfied at index $i+1$; it contradicts the fact that there is no successor for Y^{cex} satisfying with all successor of X^{cex} ψ. Since either X^{cex} is unreachable or there is an alternative Y^{cex} for each

Input: A CTI $X^{cex}, X^{cex\prime}, Y^{cex}$, a STS M and $\varphi(V[x], V[y])$
Output: A strengthening of φ w.r.t. M
1 $Q(V[x], V[x]', V[y]) := Itp(X^{cex} \wedge X^{cex\prime} \wedge Y^{cex}, T(V[x], V[x]') \wedge T(V[y], V[y]') \wedge \varphi(V[x]', V[y]') \wedge \varphi(V[x], V[y]))$
2 **if** *getVars(Q)* $\cap V[x]'$ **then**
3 $\quad Qpre(V[x], V[y]) := QElim(\exists V[x]'.[T(V[x], V[x]') \wedge Q(V[x], V[x]', V[y])])$
4 **end**
5 **else** $Qpre := Q$
6 **return** $\varphi \wedge \neg Qpre$

Algorithm 6: Generalization using quant. elim. and Craig interpx.

X^{cex}, then any counterexample for $\psi \wedge \neg(X^{cex} \wedge Y^{cex})$ is also a counterexample for ψ.

itp: For this case we need to show that *Qpre* has the same characteristic of $X^{cex} \wedge Y^{cex}$ i.e. there is a x successor of *Qpre* such that no successor of y can satisfy ψ.

First of all, let $A = X^{cex} \wedge X^{cex\prime} \wedge Y^{cex}$ and $B = T(V[y], V[y]') \wedge P(V[x]', V[y]')$: $A \wedge B$ is UNSAT (note that in the original algorithm we add more predicates to heuristically shrink the interpolant). This occurs because $(X^{cex}, X^{cex\prime}, Y^{cex})$ is a CTI and thus there is no $V[y]'$ such that $T(V[y], V[y]')$ and $P(V[x]', V[y]')$.

Then, following the algorithm, we obtain $Q(V[x], V[x]', V[y])$ which generalizes the CTI and for which we are guaranteed that there is no $V[y]'$ satisfying $T(V[y], V[y]') \wedge P(V[x]', V[y]')$.

If Q does not contain $V[x]'$ variables, then *Qpre* is Q and satisfies the desired constraint (there is a successor of x such that any successor of y violates x). If not, then the pre-image obtained via quantifier elimination has that characteristic.

□

7.1.4 Incrementality

One of the advantages of classical k-induction lies in the incrementality of the verification (see e.g. [149]). In this section, we present our incremental algorithm to verify induction and k-induction tailored for SMT solvers that support incremental verification of quantified formulae. To enable incremental reasoning, the proof obligations are first rewritten by removing the outermost universal quantifiers. The remaining existential quantifiers are then pushed inward, introduced only over the sub-formulae that contain their corresponding variables, thus preserving the formula's closure. Finally, negation is applied to transform the validity of the implication into a satisfiability problem over a conjunction. The conjunctive proof obligation for K-Induction (without simple path) is then:

$$\begin{gathered} \mathit{Unroll}_k(\overline{V}[x]) \wedge \mathit{Unroll}_{k-1}(\overline{V}[y]) \wedge \textstyle\bigwedge_{0 \le i < k} P(V[x]_i, V[y]_i) \wedge \\ \neg \exists V[y]_k . [T(V[y]_{k-1}, V[y]_k) \wedge P(V[x]_k, V[y]_k)] \end{gathered}$$

Similarly, the conjunctive proof obligation for BMC is:

$$\begin{gathered} I(V[x]) \wedge \mathit{Unroll}_k(\overline{V}[x]) \wedge \neg \exists V[y]_0, \ldots V[y]_k . [\textstyle\bigwedge_{0 \le i \le k} P(V[x]_i, V[y]_i) \wedge \\ I(V[y]) \wedge \mathit{Unroll}_k(\overline{V}[y])] \end{gathered}$$

Input: A STS M, a formula $\forall x \exists y. \mathbf{G}\psi$
Output: $M \models \forall x \exists y. \mathbf{G}\psi$?

1 $solverKind = IncrSolver()$, $solverBMC = IncrSolver()$
2 $\varphi := \psi$
3 res:= *UNKNOWN* **for** i=0; res = *UNKNOWN*; i++ **do**
4 **if** $(i == 0)$ **then** $solverBMC.addAssertion\ (I(V[x]_0))$
5 **else** $solverBMC.addAssertion\ (T(V[x]_{i-1}, V[x]_i))$
6 $ex_\varphi := \exists V[y]_0, \ldots, V[y]_i. Unroll_i(\overline{V[y]}[M]) \wedge \bigwedge_{0 \leq j \leq i} \varphi(V[x]_i, V[y]_i)$
7 **if** $solverBMC.solve\ (\neg ex_\varphi)$ == SAT **then** res = INVALID
8 **else**
9 **if** $(i$==0$)$ **then** $ex_\varphi := \exists V[y]_i. T(V[y]_{i+1}, V[y]_i) \wedge \varphi(V[x]_i, Y_i)$
 $solverKind.addAssertion\ (\neg ex_\varphi)$
10 **else**
11 $solverKind.addAssertion\ (T(V[y]_{i+1}, V[y]_i))$
12 **end**
13 $solverKind.addAssertion\ (T(V[x]_{i+1}, V[x]_i))$
14 $solverKind.addAssertion\ (\varphi(V[x]_{i+1}, V[y]_{i+1}))$
15 **for** $(j = 1; j < i; j++)$ **do**
16 $solverKind.addAssertion\ (Diff(V[x] \cup V[y], i, j)))$
17 **end**
18 **if** $\neg solverKind.solve()$ **then** res = VALID
19 **end**
20 **end**
21 **return** res

Algorithm 7: Incremental k-induction for hyperproperties

Regarding BMC, the potential for improvement through incrementality is limited to the unrolling up to $Unroll_k(\overline{V[x]}[M])$. The structure of the problem with the increasing internal quantification makes it harder to reason incrementally.

On the other hand, the k-induction proof obligation has only one inner quantification. By fixing the index in the quantified conjunction and progressively adding constraints to the universally quantified part of the formula, the proof can be incrementally constructed. The approach is detailed in Algorithm 7, which enables solving k-induction (without refinement).

7.2 Implementation and Experimental Evaluation

7.2.1 Implementation

We implemented the algorithms presented in this chapter in a prototype tool, implemented in Python3 and leveraging the `PySMT` [175] library as the main interface to various SMT solvers. Our tool uses `Z3` [139] for the verification of quantified SMT formulae and for performing quantifier elimination. For interpolation, we use the default solver backend for interpolation of `PySMT` (MathSAT5[104]). For QBF solving we use `QuAbs` [193]. Our tool implements several techniques:

- **Induction with Refinement:** It implements the method described in Section 7.1.1 with the refinement of Section 7.1.3.
- **k-Induction:** It implements the method of Section 7.1.2 with and without refinement. When refinement is enabled, the tool refines the CTI after each check of k-induction.
- **Incremental Techniques:** It implements k-induction without refinement in the incremental fashion described in Section 7.1.4 (only for Z3).

The input to our tool are system models specified in the SMV input language, extended with HyperLTL formulae. This syntax is closely aligned with the one used by the tools `HyperQB` [199] and `AutoHyper` [38]. Our tool supports both finite and infinite state systems, as long as they can be expressed using theories supported by Z3 (e.g., linear integer and linear real arithmetic). However, since the tools we compare against only support finite-state systems, all benchmarks used in our evaluation are restricted to finite-state SMV models.

7.2.2 Experimental evaluation

Experimental setup.

We evaluate our prototype on a diverse benchmark suite, collected from AutoHyper [38], HyperQB [199], HyPro [36] and HyperATL* [35, 39]. The benchmarks[2] cover a variety of property types and systems including:

[2]The results of the experimental evaluation can be found at `https://drive.proton.me/urls/B55NGVZE0C#nXsOW3pC5MAX`.

- **Path planning and robotic robustness**: Given a grid with a set of obstacles, a starting position and a goal position, the shortest path property is $\exists x \forall y.(\neg goal[y])$ **U** $goal[x]$. The second property considers a set of starting positions and a set of goal positions, and states whether there is a single plan to reaching a goal from all the starting positions. Both specifications are $\forall\exists$ safety properties checked over grids of various sizes. Both benchmarks are taken from the experimental evaluation of HyperQB [199].

- **Symmetry in mutual exclusion protocols**: variants of *Lamport's bakery algorithm* enriched with symmetric treatment of processes. The original Bakery implementation violates symmetry because if the process with lower pid is favoured. A modified model uses a flip variable and, if the flip variable is true, we compare the PID in increasing order. We consider models with 2, 3 and 4 processes with a property that also encodes strategy in the formula (i.e. $flip[x] \leftrightarrow \neg flip[y]$) and a version without it. These symmetry models are variations of the bakery models used in the experimental evaluation of HyperQB [199]. While the original models were false, meaning the protocol was not symmetric, this version introduces a flip mechanism that ensures symmetry is satisfiable.

- **Non-interference in Boolean programs**: *Generalized non-interference (GNI)* over non-terminating Boolean programs with varying bit-widths (e.g., 2, 4, and 6 bits). The models are expressed as Boolean programs (only supported by AutoHyper) and using the SMV format. The pseudo-code of the Boolean programs are depicted in Figure 7.1 and Figure 7.2. These models are taken from the experimental evaluations of AutoHyper [38], HyPro [36], as well as from [35] and [39]. We extended them by introducing variations and increasing the size of the models increase the amount of bits of each program.

Since our tool supports only $\forall^*\exists^*.\varphi_S$, the benchmarks are restricted to this fragment. The evaluation includes both a comparison with `AutoHyper` and `HyPro` and an analysis of the prototype's configurations. We extended the benchmarks increasing the size of the programs and increasing (and reducing) the state space to various configurations to better analyze the performance of each approach.

Experiments for our tool, `AutoHyper` and `HyPro` were conducted on a SLURM-managed high-performance computing cluster. Each job was allocated a single node with homogeneous hardware: Intel Xeon Gold 6226R CPUs running at 2.9GHz, equipped with 32 logical cores and 16 GB of memory, with a strict runtime limit of 2 hours per

job, a limit to single core usage and a memory cap of 8GB. Due to cluster policy and dependency isolation requirements, both tools were executed within `Singularity` containers.

In contrast, experiments involving `HyperQB` [199] were conducted on a dedicated machine outside the cluster environment. This decision was made due to technical issues in setting up `HyperQB` on the cluster. As `HyperQB` supports only falsification of HyperLTL formulae, we restricted this comparison to a subset of benchmarks containing known counterexamples.

Beyond tool-to-tool comparison, we conducted an in-depth evaluation of the various verification strategies supported by our prototype, including plain induction, k-induction, and their respective combinations with refinement and incremental solving, as outlined in Section 7.1.3. This analysis aims to assess the impact of refinement techniques and incremental solving on performance and scalability across a pool of benchmarks.

Results.

The overall performance results considering `AutoHyper` and `HyPro` are summarized in the cactus plot in Figure 7.4, with Table 7.1 and Table 7.2 showing the results of each virtual-best for each category. The maximum number of solved instances are obtained by k-induction with refinement and induction with refinement. However, the various configurations and tools exhibit complementary strengths, as evidenced by the virtual best solver significantly outperforming any individual approach.

Detailed comparison with AutoHyper and HyPro Both k-induction and induction consistently outperformed AutoHyper on a substantial subset of benchmarks. We attribute this to the scalability limitations of explicit-state techniques, which tends to struggle with larger systems. In contrast, our symbolic approach, especially when combined with refinement, is able to generalize properties more effectively and provide invariants on larger benchmarks. For example, while AutoHyper performs well on non-interference benchmarks involving Boolean programs with few bits, its performance degrades significantly as the bit-width increases (e.g., 4–6 bits), often resulting in timeouts or out-of-memory failures. On the other hand, AutoHyper outperforms our tool on some benchmarks, particularly those involving robust path planning and shortest path properties. These examples require deep unrolling to uncover counterexamples—an area where our BMC-based method is less efficient. Conversely, when properties

```
1  o ← true^k
2  while true do
3  |  h ← nondet^k()
4  |  if h[0] then
5  |  |  o ← !o
6  |  end
7  |  else
8  |  |  o ← (!o) & (h | !h)
9  |  end
10 end
```

(a) concur p1

```
1  while true do
2  |  h ← nondet^k()
3  |  b ← l
4  |  l ← nondet^k()
5  |  o ← l & b
6  end
```

(b) concur p2

```
1  o ← true^k
2  while true do
3  |  h ← nondet^k()
4  |  if nondet() then
5  |  |  o ← true^k
6  |  end
7  |  else
8  |  |  o ← false^k
9  |  end
10 end
```

(c) concur p3

```
1  while true do
2  |  h ← nondet^k()
3  |  if nondet() then
4  |  |  if h[0] then
5  |  |  |  o ← true^k
6  |  |  end
7  |  |  else
8  |  |  |  o ← false^k
9  |  |  end
10 |  end
11 |  else
12 |  |  if h[0] then
13 |  |  |  o ← false^k
14 |  |  end
15 |  |  else
16 |  |  |  o ← true^k
17 |  |  end
18 |  end
19 end
```

(d) concur p4

Figure 7.1: Four algorithms over Boolean vectors of size k introduced in [35]. Common variables: `h[1..k]`, `l[1..k]`, `o[1..k]`, `b[1..k]`. Here, $\texttt{nondet}^k\texttt{()}$ denotes a nondeterministic assignment to all k elements.

```
1 o ← true^k
2 while true do
3 |  o ← !o
4 end
```

(a) lmcs p1

```
1 o ← true^k
2 while true do
3 |  l ← nondet^k()
4 |  o ← l
5 end
```

(b) lmcs p2

```
1 o ← true^k
2 while true do
3 |  l ← nondet^k()
4 |  if nondet() then
5 |  |  o ← l
6 |  end
7 |  else
8 |  |  o ← !l
9 |  end
10 end
```

(c) lmcs p3

```
1 while true do
2 |  if nondet() then
3 |  |  h ← nondet^k()
4 |  |  o ← h
5 |  end
6 |  else
7 |  |  h ← nondet^k()
8 |  |  o ← !h
9 |  end
10 end
```

(d) lmcs p4

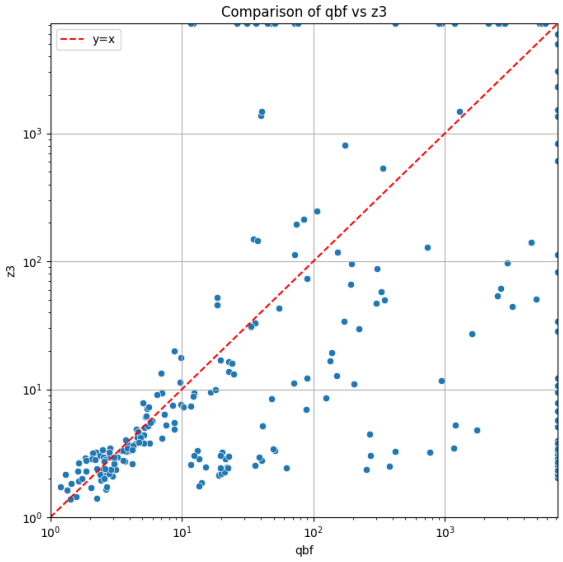
Figure 7.2: Four examples operating on Boolean vectors of size k. Variables: `h[1..k]`, `l[1..k]`, `o[1..k]` coming from [39]. Here, $\texttt{nondet}^k\texttt{()}$ assigns arbitrary values to all k bits; `nondet()` is a nondeterministic Boolean choice.

are false and the counterexamples are short, our approach is significantly faster than AutoHyper.

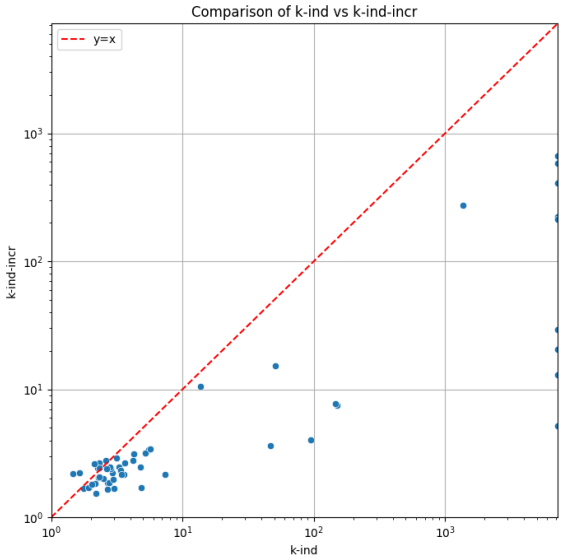
Our experimental evaluation also highlights notable differences in performance between our approach and HyPro. While HyPro shares some scalability limitations observed in AutoHyper—solving fewer benchmarks overall—HyPro stands out in some specific cases. Notably, HyPro was the only tool that handles the Bakery benchmarks with three processes without requiring a user-provided strategy, which suggests that prophecy-based strategy synthesis (implemented in HyPro) can serve as a powerful refinement technique in certain scenarios. On the other hand, HyPro struggled significantly on larger or more complex instances and was unable to handle many false benchmarks.

Detailed comparison with HyperQB. Table 7.3 reports the verification times obtained on a Linux workstation equipped with an Intel(R) Core(TM) i5-5300U CPU running at 2.30 GHz and 12 GB of RAM. Unlike comparisons that report only total runtimes, we analyze the time spent at each bound i and accumulate it up to the step where the property is disproved. This provides a finer-grained view of how each solver configuration progresses across successive bounds until falsification. Overall, the results indicate that model parsing and formula encoding in HyperQB introduce a noticeable overhead during the translation phase. This overhead likely arises because the tool cannot incrementally construct the QBF encoding and must therefore repeat the entire translation process for each bound. Nevertheless, once the formulas are generated, the per-step solving times of HyperQB and HyperKind-QBF remain comparable, since both rely on the same underlying QBF solver (QuAbS), the resulting formulas are structurally similar, and for these benchmarks the k-inductive proof obligations are falsified quickly. When comparing the HyperKind variants, the SMT-based configuration (k-ind-z3) fails to solve the more demanding robust path-planning instances when run non-incrementally. Despite this limitation, the incremental configurations of HyperKind, particularly k-ind-incr, consistently outperform both the QBF-based HyperKind variant and the HyperQB baseline, underscoring the importance of incremental verification for achieving scalability in this setting.

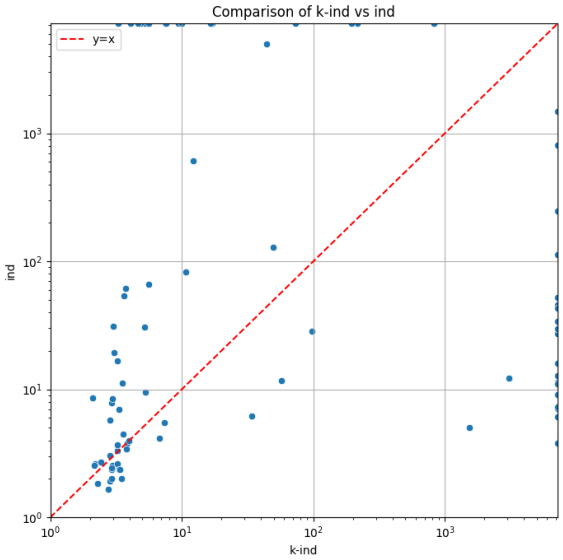
Detailed comparison of various configurations. From Figure 7.3(c), we observe that k-induction with refinement and induction with refinement produce complementary results. Specifically, k-induction performs significantly better on invalid benchmarks and when unrolling helps eliminate large portions of counterexamples to induction.



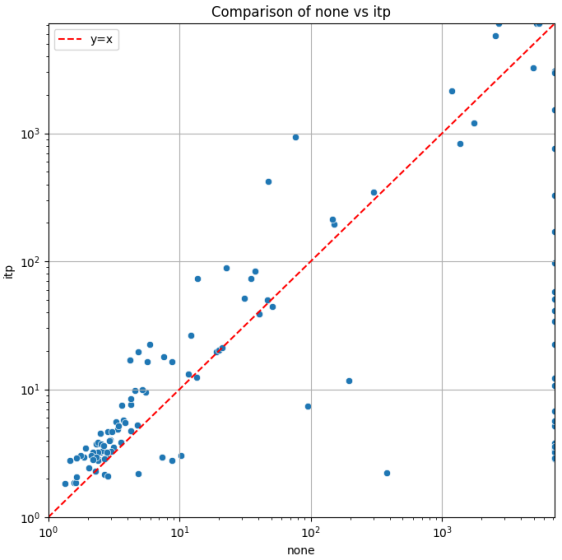
(a) Comparison between z3 and qbf configurations



(b) Comparison between k-induction without refinement and k-induction with incremental solving



(c) Comparison between k-induction with refinement and induction (with refinement)



(d) Comparison between k-induction with refinement and k-induction without refinement

Figure 7.3: Results of the empirical evaluation comparing configurations of HyperKind.

However, unrolling also increases the complexity of the proof obligations, especially in the absence of incremental solving. For instance, in complex models like Bakery, unrolling becomes not only computationally expensive but also less effective, as it may fail to eliminate enough bad configurations, while interpolation within induction can be more effective in filtering out undesired behaviours. Similarly, in GNI benchmarks—

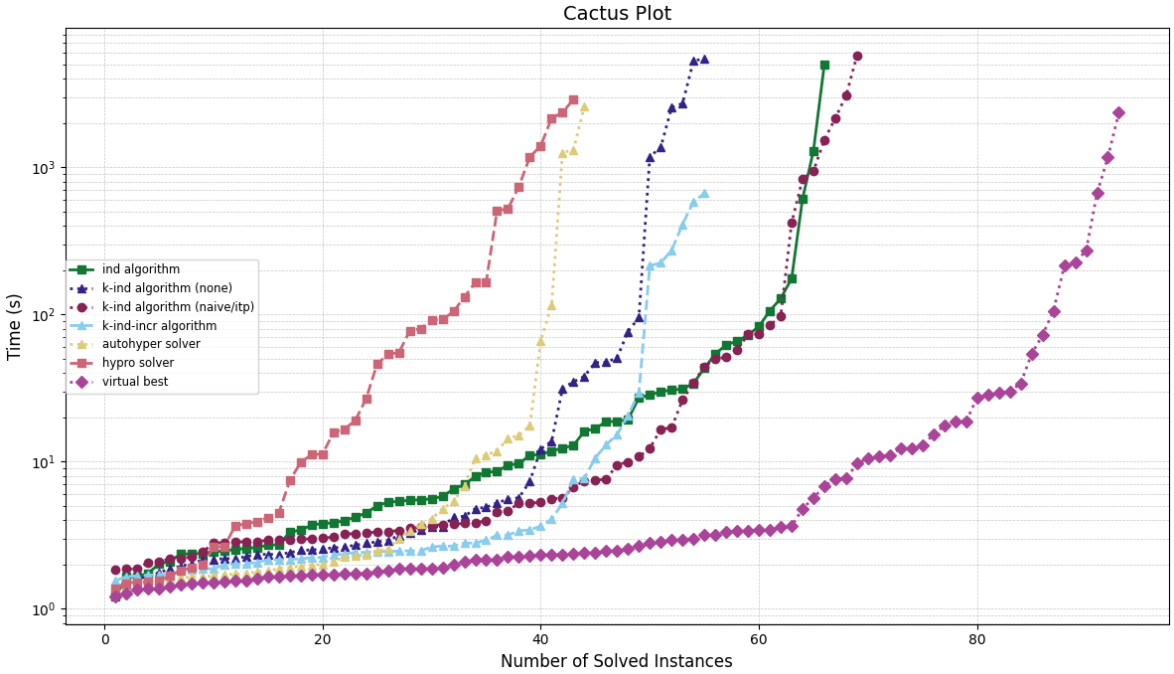


Figure 7.4: Cactus plot with overall comparison between HyperKind, `AutoHyper` and `HyPro`. Comparing the various configurations over all the instances. The x-axis represents solved instances, and the y-axis shows solving time. Virtual best considers the minimum time taken by each configuration/tool.

where some variables crucial to the property are not explicitly referenced in the property itself—k-induction struggles. For example, when program locations are not tightly connected to the specification, k-induction suffers, whereas standard induction with refinement can still succeed.

We also analyzed the role of refinement in k-induction. Figure 7.3(d) presents a scatter plot comparing k-induction with and without interpolation-based refinement. Although refinement can introduce some overhead by making the property more complex, the plot shows that it improves performance, often significantly outperforming k-induction without refinement.

Additionally, we evaluated the impact of using different backend solvers. Due to technical limitations, we were unable to use `QuAbs` for incremental property checking, and we focused on induction and k-induction proof obligations. Figure 7.3(a) shows how the solvers exhibit complementary strengths: QBF is more effective on path-planning—even without incrementality—while Z3 performs better on software-related benchmarks.

Incremental solving is shown to be particularly useful. As Figure 7.3(b) shows, the k-ind-incr variant consistently outperforms plain k-ind. Even in BMC scenarios—where incrementality is less naturally applicable—we observed noticeable speedups, which suggests that extending incremental solving to both induction with refinement and k-induction with refinement could be a valuable direction for future work.

7.3 Related Works

While the broader landscape of HyperLTL verification has been discussed in Chapter 3 (Section 3.4), we briefly revisit key lines of work most relevant to our approach.

The automata-theoretic approach, initially proposed in [120, 162], forms the basis of tools like MCHyper and AutoHyper. MCHyper targets the quantifier-alternation-free fragment using self-composition. AutoHyper [38], in contrast, supports arbitrary HyperLTL formulae via explicit automata constructions and inclusion checking [38], though its reliance on explicit-state techniques limits scalability, especially in the presence of alternations.

Game-based approaches [125, 36] tackle $\forall\exists$ verification by synthesizing a strategy for the existential trace in response to a universal trace. To achieve completeness, they need to introduce ω-regular prophecy variables. Our work shares the goal of handling $\forall\exists$ properties but avoids prophecy variables entirely. Instead, we cast the problem as invariant inference through inductive generalization, refining counterexamples to build k-inductive invariants.

Another line of work [199] adapts Bounded Model Checking (BMC) to quantified HyperLTL by encoding the finite semantics into QBF. Later, [200] introduced an approach for verifying quantified safety properties using inductive reasoning explicitly instantiating the states. Their approach resembles our basic inductive algorithm, with the key distinction that the approach in [200] instantiates quantifiers and relies on SAT solvers to verify the property. In contrast, our method provides *completeness guarantees* and incorporates *invariant strengthening* as part of the proof process. Recently, in [128], the authors proposed a coinductive approach for the verification of $\forall^*\exists^*$ safety HyperLTL formulae. Part of the proof obligations and the technical ideas are related to the work presented in this Chapter; However, their framework is built for interactive proof systems and is based on the Coq theorem prover. Finally, our work draws inspiration from classic k-induction [272], a widely used technique in trace property verification. To our knowledge, our approach is the first to lift k-inductive reasoning to $\forall\exists$ hyperproperties.

Instance	vb-ind	vb_kind-no-ref	vb-kind-ref	k-ind-incr	AutoHyper	HyPro
10_NI_big_tini	TO	2710.73	TO	**223.92**	MO	MO
10_NI_big_tsni	TO	2549.72	5771.73	**213.98**	MO	MO
10_NI_huge_tini	TO	5265.11	TO	**666.0**	TO	MO
10_NI_small_tini	TO	5.54	9.47	**3.38**	10.55	522.08
10_NI_small_tsni	TO	5.24	9.95	**3.16**	11.08	507.78
10_NI_tini	TO	34.79	73.84	**7.56**	MO	MO
10_NI_tsni	TO	37.92	84.2	**7.68**	MO	MO
bakery_3procs_lincl	TO	TO	TO	TO	MO	**1173.53**
bakery_sym_3procs	8.55	**1.86**	2.1	2.35	1313.83	UNK
bakery_2procs	1292.24	TO	TO	409.38	MO	**53.71**
bakery_2procs_strat	3.79	3.58	3.85	**2.15**	MO	54.97
bakery_3procs	TO	TO	TO	TO	MO	**2351.46**
bakery_3procs_strat	128.28	47.05	49.74	**3.63**	TO	2895.95
buffer_sched	TO	3.45	5.19	**2.15**	MO	2153.78
buffer_sched_big	TO	3.61	7.46	**2.66**	MO	MO
buffer_sched_huge	TO	4.19	16.97	**2.79**	TO	MO
buffer_sched_small	TO	2.85	4.64	**2.41**	66.05	45.89
buffer_sched_tiny	TO	2.55	3.74	1.98	**1.7**	7.42
buffer_unsched	TO	3.27	5.58	**2.47**	MO	TO
buffer_unsched_big	TO	4.24	7.58	**3.15**	MO	MO
buffer_unsched_huge	TO	5.67	16.54	**3.43**	TO	MO
buffer_unsched_small	TO	2.5	4.51	**2.33**	MO	740.58
buffer_unsched_tiny	TO	2.4	3.27	**1.69**	3.76	77.41
den_large	TO	TO	TO	MO	TO	TO
den_medium	TO	1364.3	831.79	**272.32**	MO	MO
den_small	4989.81	50.64	44.16	**15.24**	1256.58	TO
lmcs_p1_2	1.65	**1.46**	1.86	2.18	2.32	16.59
lmcs_p1_2_smpl	2.01	1.94	2.84	1.7	1.8	15.76
concur_p1_2	3.7	TO	3.23	TO	**1.73**	2.62
concur_p1_2_smpl	5.29	TO	TO	TO	1.97	**1.67**
concur_p1_2_sync	1.64	**1.64**	2.06	1.65	4.06	MO
concur_p1_4	4.47	TO	**3.57**	TO	5.35	26.84
concur_p1_4_smpl	16.04	TO	TO	TO	**2.54**	18.97
concur_p1_4_sync	2.44	2.78	2.94	**1.86**	MO	TO
concur_p1_6	7.9	TO	**2.92**	TO	TO	MO
concur_p1_6_smpl	**27.16**	TO	TO	TO	TO	1398.39
concur_p1_6_sync	2.53	2.7	2.17	**1.87**	TO	MO
concur_p2_2	30.67	TO	5.22	TO	TO	**1.36**
concur_p2_2_smpl	175.14	TO	TO	TO	TO	**1.49**
concur_p2_2_sync	2.61	**2.19**	2.81	2.47	TO	MO
concur_p2_4	82.89	TO	**10.78**	TO	MO	MO
concur_p2_4_smpl	TO	TO	TO	TO	MO	MO
concur_p2_4_sync	9.47	4.78	5.27	**2.46**	MO	MO
concur_p2_6	61.88	**2.34**	3.73	2.45	TO	MO
concur_p2_6_smpl	TO	TO	TO	MO	TO	MO
concur_p2_6_sync	11.14	3.14	3.51	**2.92**	TO	MO
concur_p3_2	4.18	TO	6.73	TO	**1.65**	3.91

Table 7.1: Table comparing virtual best of each configuration (part 1).

Instance	vb-ind	vb-kind-no-ref	vb-kind-ref	k-ind-incr	AutoHyper	HyPro
concur_p3_2_smpl	6.49	TO	TO	TO	**2.09**	3.77
concur_p3_4	65.94	TO	**5.64**	TO	TO	166.39
concur_p3_4_smpl	**18.67**	TO	TO	TO	TO	91.95
concur_p3_6	609.3	TO	**12.29**	TO	TO	MO
concur_p3_6_smpl	**18.7**	TO	TO	TO	TO	104.41
concur_p4_2	5.0	TO	1532.42	TO	**1.36**	4.49
concur_p4_2_smpl	5.57	TO	TO	TO	**2.0**	4.16
concur_p4_2_sync	**2.36**	TO	3.4	TO	TO	TO
concur_p4_4	**12.31**	TO	3070.41	TO	MO	164.19
concur_p4_4_smpl	**9.65**	TO	TO	TO	MO	131.36
concur_p4_4_sync	**3.42**	TO	3.8	TO	MO	TO
concur_p4_6	**34.03**	TO	TO	TO	TO	MO
concur_p4_6_smpl	**29.86**	TO	TO	TO	TO	MO
concur_p4_6_sync	5.79	TO	**2.85**	TO	TO	MO
lmcs_p1_2	1.74	**1.35**	1.84	2.23	1.46	1.98
lmcs_p1_2_smpl	2.56	2.34	2.97	2.06	**1.67**	1.9
lmcs_p1_4	2.46	1.76	3.02	1.67	1.94	**1.56**
lmcs_p1_4_smpl	2.35	2.59	3.34	2.79	1.62	**1.51**
lmcs_p1_6	2.7	2.03	2.42	**1.8**	11.8	11.24
lmcs_p1_6_smpl	3.97	2.91	3.95	2.22	**1.5**	1.53
lmcs_p2_2	3.31	2.5	2.24	2.01	**1.64**	9.89
lmcs_p2_2_smpl	5.36	TO	34.21	TO	**2.24**	11.13
lmcs_p2_4	6.98	TO	**3.32**	TO	MO	MO
lmcs_p2_4_smpl	11.78	TO	57.6	TO	**3.0**	MO
lmcs_p2_6	8.47	**2.34**	2.98	2.66	TO	MO
lmcs_p2_6_smpl	**28.28**	TO	97.62	TO	TO	MO
lmcs_p3_2	**3.82**	TO	TO	TO	1.78	80.05
lmcs_p3_2_smpl	5.46	TO	TO	TO	**2.29**	92.32
lmcs_p3_4	19.23	**2.14**	3.06	2.63	15.08	TO
lmcs_p3_4_smpl	**11.03**	TO	TO	TO	MO	TO
lmcs_p3_6	2.72	**2.26**	2.85	2.45	TO	MO
lmcs_p3_6_smpl	**12.87**	TO	TO	TO	MO	TO
lmcs_p4_2	31.03	2.16	2.79	1.84	**1.72**	MO
lmcs_p4_2_smpl	42.98	TO	TO	TO	**1.88**	MO
lmcs_p4_4	16.74	2.19	3.22	**1.55**	14.35	TO
lmcs_p4_4_smpl	**72.18**	TO	TO	TO	MO	TO
lmcs_p4_6	53.97	2.63	3.66	**2.41**	TO	TO
lmcs_p4_6_smpl	**105.69**	TO	TO	TO	TO	TO
mutation	**1.2**	1.61	1.86	2.03	2.48	UNK
robot_robust_100	TO	12.13	26.3	5.21	**3.4**	UNK
robot_robust_1600	TO	47.26	418.69	20.5	**4.78**	UNK
robot_robust_3600	TO	76.43	941.64	**29.46**	2585.88	UNK
robot_sp_100	TO	31.42	51.23	12.99	**1.61**	UNK
robot_sp_1600	TO	1180.04	2156.88	580.02	**6.83**	UNK
robot_sp_3600	TO	5492.98	TO	MO	**17.51**	UNK
simple_prog1	2.1	4.88	2.2	1.71	**1.53**	1.82
simple_prog2	2.37	7.32	2.94	2.16	**1.9**	2.65
simple_prog3	5.49	95.31	7.38	4.05	**1.26**	3.66
snark	TO	**13.71**	73.03	10.58	115.78	MO

Table 7.2: Table comparing virtual best of each configuration (part 2).

Solver	Benchmark	1	2	3	4	5	6	7	8	cumulative
HyperQB	mut	0.245								0.245
HyperQB-	mut	0.02								0.02
k-ind-z3	mut	0.023								0.023
k-ind-qbf	mut	0.008								0.008
k-ind-incr	mut	0.013								0.013
HyperQB	rb100	0.516	0.587	0.699	0.912	0.885	1.006	1.633	1.371	18.947
HyperQB	rb100	0.127	0.214	0.307	0.459	0.511	0.625	1.25	0.953	13.903
k-ind-qbf	rb100	0.166	0.258	0.321	0.387	0.529	0.589	0.698	1.159	10.635
k-ind-incr	rb100	0.103	0.098	0.119	0.14	0.151	0.158	0.203	0.237	3.723
HyperQB	rb1600	2.059	2.564	3.245	4.176	4.315	5.156	6.183	6.561	86.733
HyperQB-	rb1600	0.854	1.372	2.036	2.72	3.131	3.981	4.996	5.377	70.913
k-ind-qbf	rb1600	0.87	1.234	1.641	2.06	2.644	3.033	3.779	4.215	47.455
k-ind-incr	rb1600	0.455	0.464	0.557	0.559	0.713	0.734	0.982	1.302	17.484
HyperQB	rb3600	3.097	3.902	4.988	5.746	6.949	8.042	9.396	10.368	132.548
HyperQB-	rb3600	1.35	2.161	3.208	4.026	5.177	6.298	7.619	8.61	109.657
k-ind-qbf	rb3600	1.424	2.227	2.82	3.646	4.395	5.155	5.943	7.201	77.605
k-ind-incr	rb3600	0.797	0.758	0.863	1.054	1.169	1.668	1.384	2.48	26.324
HyperQB	snark1	50.232	52.435	55.431	58.734	62.223	63.156	67.013	70.204	884.666
HyperQB-	snark1	4.002	6.386	9.206	11.452	14.692	17.16	20.01	22.852	272.855
k-ind-z3	snark1	0.22	0.29	0.361	0.458	0.54	0.65	0.751	0.909	13.79
k-ind-qbf	snark1	0.369	0.534	0.771	0.988	1.214	1.47	1.663	2.051	28.927
k-ind-incr	snark1	0.224	0.235	0.271	0.299	0.344	0.424	0.482	0.467	8.412

Table 7.3: Performance results of the solvers. HyperQB- denotes HyperQB without file parsing; ‘mut‘ refers to mutation testing; ‘rb100‘, ‘rb1600‘, and ‘rb3600‘ correspond to robotic robustness in path planning with 100, 1600, and 3600 planning steps, respectively. The complete bound (total number of runs) is longer than 8: it is 14 for the rb benchmarks and 17 for snark.

Chapter 8

Asynchronous Hyperlogics over Infinite and Finite traces

In Chapter 1, we highlighted one of the central challenges in formal verification: balancing expressiveness and tractability when reasoning about complex system behaviors. Building on this theme, Chapter 4 developed model-checking techniques for LTL modulo background theories, enabling reasoning about single execution traces that include data-rich or infinite-state information. Chapter 7 then moved beyond individual executions, introducing inductive verification methods for $\forall\exists$-hyperproperties expressed in HyperLTL— thus extending single-trace reasoning to the multi-trace, *synchronous* setting.

While synchronous HyperLTL provides a powerful framework for reasoning about relations between traces, it enforces that all executions progress in perfect lockstep. This restriction often proves too rigid in practice. In many systems,e ven for synchronous programs, one may wish to abstract away irrelevant internal steps, which effectively yields an *asynchronous* perspective. To capture such behaviors faithfully, we need a more flexible semantic model. This chapter extends the study of hyperproperties in two directions: first, by introducing an *asynchronous* temporal logic capable of expressing highly expressive hyperproperties combining existing asynchronous hyperlogics; and second, by developing practical algorithms for verifying properties defined in a fragment of that logics over *finite traces*.

At the core of our theoretical development is the logic *Generalized HyperLTL with Stuttering and Contexts* (GHyperLTL$_{\mathsf{S+C}}$). GHyperLTL$_{\mathsf{S+C}}$ unifies and extends two key asynchronous variants of HyperLTL: *Stuttering HyperLTL*(HyperLTL$_S$) [72] and *Context HyperLTL* (HyperLTL$_C$) [72]. It also introduces two relevant features:

- *Past temporal modalities*, enabling backward reasoning over trace histories.
- *Non-prenex quantification*, allowing trace quantifiers to appear within the scope of temporal operators.

GHyperLTL$_{\mathsf{S+C}}$ is expressive enough to subsume KLTL [190] (LTL with knowledge modalities) under both synchronous and asynchronous perfect recall semantics (with one agent). It supports the specification of rich properties such as *global promptness*, *finite-delay diagnosability* [65, 49], and *asynchronous noninterference*.

A key theoretical contribution of this work is the identification of a decidable fragment, called **Simple GHyperLTL$_{\mathsf{S+C}}$**, which strikes a balance between expressiveness and tractability. This fragment is strictly more expressive than Simple HyperLTL$_S$ (and consequently HyperLTL), and can express *non-regular* trace properties and still subsumes KLTL [190]. We show that model checking remains decidable for this fragment and present a general translation into QPTL. Detailed results about complexity are out of the scope of this thesis but can be found in [55].

Building on this foundation, we introduce a second contribution: the temporal hyperlogic over finite traces **SC-HyperLTL**. SC-HyperLTL can be seen as a practical and syntactically restricted fragment of Simple GHyperLTL$_{\mathsf{S+C}}$, adapted specifically to the *finite-trace setting*.

While the first part of this chapter focuses on the theoretical development of the logic GHyperLTL$_{\mathsf{S+C}}$ and its decidable fragment, the second part presents two *model-checking algorithms* for SC-HyperLTL:

1. A *stuttering-based translation* for quantifier alternation-free formulae, reducing verification to standard HyperLTL model checking via self-composition techniques.
2. An algorithm using *auxiliary history variables*, capable of handling formulae with arbitrary quantifier alternation by reducing to HyperLTL verification.

We demonstrate the feasibility of our approach through a prototype implementation and experimental validation.

Contributions In summary, this chapter delivers both a comprehensive logical foundation (via GHyperLTL$_{\mathsf{S+C}}$) and a verification framework (via SC-HyperLTL) for reasoning about asynchronous hyperproperties, especially in finite-trace settings. Together, they provide a unified theoretical and practical toolkit for advancing the verification of complex relational system properties.

The content presented in this chapter is the result of these works [55, 56]. All the work presented in this Chapter was done in collaboration with Laura Bozzelli, César Sánchez and Stefano Tonetta. In particular, Laura Bozzelli contributed significantly in the first part of this chapter, with rigorous proofs of the reductions from SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ to QPTL and with an in-depth analysis of the expressibility of the various formalisms. In this chapter, we left out the complete analysis of the complexity of QPTL detailed in [55].

Motivating Example Consider the following terminating function P, with input *in* and output *out* which iteratively set *out* to $10(in + 1)$.

```
1  int tmp = 1;
2  out = in;
3  for (int i=0;i<10;i++){
4    int j = 0;
5    while (j++ < in) tmp++;
6    out = tmp;
7  }
```

For each couple of executions, we would like to show that—in spite of the internal program dynamics—if *in* is smaller in one of the two executions at the beginning, the n-th output will be also smaller during the whole execution. We call this property *reactive monotonicity*, which requires relating multiple executions of the same program and, since the program has to terminate, to reason with finite semantics. One attempt using a variation of HyperLTL with finite semantics is the following.

$$\varphi^{sync}_{rmon} := \forall x.\forall y.in[x] < in[y] \rightarrow \mathbf{G}(out[x] < out[y])$$

This formula expresses expresses that, for each pair of executions of P (here represented by trace variables x and y) if x starts with an input lower than that of y ($in[x] < in[y]$), then it is always the case that the output of x is smaller than the output of y: $\mathbf{G}(out[x] < out[y])$. During its execution, the program might perform internal computations that are not relevant to the desired property but that can change the outcome due to the synchronous alignment. Therefore, even if P guarantees that the same assignment sequences of the variable *out* are "reactive monotonic" in two executions, the property

can be violated:

$$\sigma_1 := \cdots \to (j = 2, l = 5, out = 2, in = 2) \to (j = 3, l = 6, out = 1) \to (out = 3)$$
$$\sigma_2 := \cdots \to (j = 2, l = 5, out = 3, in = 3) \to (j = 3, l = 5, out = 1) \to (out = 3)$$

To fix this excessive synchrony, we adopt a semantics that evaluates the traces *asynchronously* in general but *synchronously* over a set of interesting points. In our example, we want to check the values of *out* at those points in which *out* changes, which is achieved with the *stuttering* variants of the temporal modalities. We also introduce *context* temporal modalities to reason over a subset of the traces in a formula, restricting the temporal progress to those traces. The use of contexts in this example is also motivated by the additional challenge that finite traces introduce. The resulting formula is:

$$\varphi^{SC}_{rmon} := \forall x.\forall y.\{out\}.in[x] < in[y] \to \mathbf{G}((end[x] \leftrightarrow end[y]) \wedge out[x] < out[y])$$

where $end[x]$ captures whether trace x has a successor, defined as $\langle\{x\}\rangle\mathbf{X}\top$ using the context modality. The set $\{out\}$ denotes that the "interesting" positions are the points in which *out* changes. The "always" modality is then evaluated only over interesting positions. Finally, φ^{SC}_{rmon} uses singleton contexts with next operator in $end[x]$ and $end[y]$ to ensure that the traces have the same amount of "interesting points", by requiring that x has a successor iff y has a successor.

Outline This chapter is organized as follows. In Section 8.1, we introduce the logic GHyperLTL$_{\mathsf{S+C}}$, its decidable fragment SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ and we compare the expressiveness of the fragments of the logics with other formalisms. In Section 8.2, we show that SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ is decidable providing a reduction to QPTL. In Section 8.3, we introduce a fragment of SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ called SC-HyperLTL over finite traces showing its decidability, some specifications and a theoretical property. In Section 8.4 introduces two practical model checking techniques for SC-HyperLTL implemented in a proof-of-concept tool. Finally, in Section 8.5, we compare the contribution presented in this chapter with related publications.

8.1 Unifying Framework for Asynchronous Extensions of HyperLTL

In this section, we introduce a novel logical framework for specifying both asynchronous and synchronous linear-time hyperproperties which unifies two known more expressive extensions of HyperLTL [121], namely *Stuttering HyperLTL* (HyperLTL$_\mathsf{S}$ for short) [72] and *Context HyperLTL* (HyperLTL$_\mathsf{C}$ for short) [72]. The proposed hyper logic, that we call *generalized HyperLTL with stuttering and contexts* (GHyperLTL$_{\mathsf{S}+\mathsf{C}}$ for short), merges HyperLTL$_\mathsf{S}$ and HyperLTL$_\mathsf{C}$ and adds two new features: past temporal modalities for asynchronous/synchronous hyperproperties and general trace quantification where trace quantifiers can occur in the scope of temporal modalities. Since model checking of the logics HyperLTL$_\mathsf{S}$ and HyperLTL$_\mathsf{C}$ is already undecidable [72], we also consider a meaningful fragment of GHyperLTL$_{\mathsf{S}+\mathsf{C}}$ which is strictly more expressive than the known *simple fragment* of HyperLTL$_\mathsf{S}$ [72]. Our fragment is able to express relevant classes of hyperproperties and, as we show in Section 8.2, its model checking problem is decidable.

8.1.1 PLTL-Relativized Stuttering and Context Modalities

Classically, a trace is stutter-free if there are no consecutive positions having the same propositional valuation unless the valuation is repeated ad-infinitum. We can associate to each trace a unique stutter-free trace by removing "redundant" positions. The logic HyperLTL$_\mathsf{S}$ [72] generalizes these notions with respect to the pointwise evaluation of a finite set of LTL formulae. Here, we consider LTL with past (PLTL).

Definition 8.1 (PLTL stutter factorization [72]) *Let Γ be a finite set of PLTL formulae and σ a trace. The* Γ-stutter factorization of σ *is the unique increasing sequence of positions $\{i_k\}_{k\in[0,m_\infty]}$ for some $m_\infty \in \mathbb{N}\cup\{\infty\}$ such that the following holds for all $j < m_\infty$:*

- *$i_0 = 0$ and $i_j < i_{j+1}$;*
- *for each $\theta \in \Gamma$, the truth value of θ along the segment $[i_j, i_{j+1})$ does not change, that is, for all $h, k \in [i_j, i_{j+1})$, $(\sigma, h) \models \theta$ iff $(\sigma, k) \models \theta$, and the same holds for the infinite segment $[m_\infty, \infty]$ in case $m_\infty \neq \infty$;*
- *the truth value of some formula in Γ changes along adjacent segments, that is, for some $\theta \in \Gamma$ (depending on j), $(\sigma, i_j) \models \theta$ iff $(\sigma, i_{j+1}) \not\models \theta$.*

Thus, the Γ-stutter factorization $\{i_k\}_{k\in[0,m_\infty]}$ of σ partitions the trace in adjacent non-empty segments such that the valuation of formulae in Γ does not change within a segment, and changes in moving from a segment to the adjacent ones. This factorization induces naturally a trace obtained by selecting the first positions of the finite segments and all the positions of the unique tail infinite segment, if any. These positions form an infinite increasing sequence $\{\ell_k\}_{k\in\mathbb{N}}$ called (Γ,ω)-*stutter factorization* of σ, where:

$$\ell_0,\ell_1,\ldots \doteq \begin{cases} i_0,i_1,\ldots & \text{if } m_\infty=\infty \\ i_0,i_1,\ldots,i_{m_\infty},i_{m_\infty}+1,\ldots & \text{otherwise} \end{cases}$$

The Γ-*stutter trace* $stfr_\Gamma(\sigma)$ *of* σ (see [72]) is defined as follows: $stfr_\Gamma(\sigma)\doteq\sigma(\ell_0)\sigma(\ell_1)\ldots$. Note that for $\Gamma=\emptyset$, $stfr_\Gamma(\sigma)=\sigma$. A trace σ is Γ-*stutter free* if it coincides with its Γ-stutter trace, i.e. $stfr_\Gamma(\sigma)=\sigma$.

As an example, assume that $V=\{p,q,r\}$ and let $\Gamma=\{p\,\mathbf{U}\,q\}$. Given $h,k\geq 1$, let $\sigma_{h,k}$ be the trace $\sigma_{h,k}=p^hq^kr^\omega$. These traces have the same Γ-stutter trace given by pr^ω.

The semantics of the Γ-relativized temporal modalities in HyperLTL$_\mathsf{S}$ is based on the notion of Γ-*successor* $succ_\Gamma(\sigma,i)$ of a pointed trace (σ,i) [72]: $succ_\Gamma(\sigma,i)$ is the pointed trace (σ,ℓ) where ℓ is the smallest position ℓ_j in the (Γ,ω)-stutter factorization $\{\ell_k\}_{k\in\mathbb{N}}$ of σ which is greater than i. Note that for $\Gamma=\emptyset$, $succ_\emptyset(\sigma,i)=succ(\sigma,i)=(\sigma,i+1)$. Hence, $\emptyset$-relativized temporal modalities in HyperLTL$_\mathsf{S}$ correspond to the temporal modalities of HyperLTL.

In this chapter, we extend HyperLTL$_\mathsf{S}$ with past temporal modalities, so that we introduce the past counterpart of the successor function. The Γ-*predecessor* $pred_\Gamma(\sigma,i)$ of a pointed trace (σ,i) is undefined if $i=0$ (written $pred_\Gamma(\sigma,i)=\mathtt{und}$); otherwise, $pred_\Gamma(\sigma,i)$ is the pointed trace (σ,ℓ) where ℓ is the greatest position ℓ_j in the (Γ,ω)-stutter factorization $\{\ell_k\}_{k\in\mathbb{N}}$ of σ which is smaller than i (since $\ell_0=0$ such an ℓ_j exists). Note that for $\Gamma=\emptyset$, $pred_\emptyset(\sigma,i)$ captures the standard local predecessor of a position along a trace.

Successors and predecessors of trace assignments. We now define a generalization of the successor $succ(\Pi)$ of a trace assignment Π in HyperLTL. This generalization is based on the notion of Γ-successor $succ_\Gamma(\sigma,i)$ of a pointed trace (σ,i) and also takes into account the context modalities $\langle C\rangle$ of HyperLTL$_\mathsf{C}$ [72], where a *context* C is a non-empty subset of VAR. Intuitively, modality $\langle C\rangle$ allows reasoning over a subset of the traces assigned to the variables in the formula, by restricting the temporal progress to those traces.

Formally, let Π be a trace assignment over some set of traces T, Γ be a finite set of PLTL formulae, and C be a context. The (Γ, C)-*successor of* Π, denoted by $succ_{(\Gamma,C)}(\Pi)$, is the trace assignment over T having domain $Dom(\Pi)$, and defined as follows for each $x \in Dom(\Pi)$:

$$succ_{(\Gamma,C)}(\Pi)(x) := \begin{cases} succ_\Gamma(\Pi(x)) & \text{if } x \in C \\ \Pi(x) & \text{otherwise} \end{cases}$$

Note that $succ_{(\emptyset,\mathsf{VAR})}(\Pi) = succ(\Pi)$. Moreover, we define the (Γ, C)-*predecessor* of Π, denoted by $pred_{(\Gamma,C)}(\Pi)$ as follows: $pred_{(\Gamma,C)}(\Pi)$ is *undefined*, written $pred_{(\Gamma,C)}(\Pi) = \mathtt{und}$, if there is $x \in Dom(\Pi)$ such that $pred_\Gamma(\Pi(x)) = \mathtt{und}$. Otherwise, $pred_{(\Gamma,C)}(\Pi)$ is the trace assignment over T having domain $Dom(\Pi)$, and defined as follows for each $x \in Dom(\Pi)$:

$$pred_{(\Gamma,C)}(\Pi)(x) := \begin{cases} pred_\Gamma(\Pi(x)) & \text{if } x \in C \\ \Pi(x) & \text{otherwise} \end{cases}$$

Finally, for each $i \geq 0$, we define the i-th application $succ^i_{(\Gamma,C)}$ of $succ_{(\Gamma,C)}$ and the i-th application $pred^i_{(\Gamma,C)}$ of $pred_{(\Gamma,C)}$ as follows, where $pred_{(\Gamma,C)}(\mathtt{und}) \doteq \mathtt{und}$:

- $succ^0_{(\Gamma,C)}(\Pi) \doteq \Pi$ and $succ^{i+1}_{(\Gamma,C)}(\Pi) \doteq succ_{(\Gamma,C)}(succ^i_{(\Gamma,C)}(\Pi))$.
- $pred^0_{(\Gamma,C)}(\Pi) \doteq \Pi$ and $pred^{i+1}_{(\Gamma,C)}(\Pi) \doteq pred_{(\Gamma,C)}(pred^i_{(\Gamma,C)}(\Pi))$.

8.1.2 Generalized HyperLTL with Stuttering and Contexts

We introduce now the novel logic GHyperLTL$_{\mathsf{S+C}}$. GHyperLTL$_{\mathsf{S+C}}$ formulae φ over V and a finite set VAR of trace variables are defined by the following syntax:

$$\varphi := \top \mid p[x] \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\, \varphi \mid \langle C \rangle \varphi \mid \mathbf{X}_\Gamma \varphi \mid \mathbf{Y}_\Gamma \varphi \mid \varphi\, \mathbf{U}_\Gamma\, \varphi \mid \varphi\, \mathbf{S}_\Gamma\, \varphi$$

where $p \in V$, $x \in \mathsf{VAR}$, $\langle C \rangle$ is the context modality with $\emptyset \neq C \subseteq \mathsf{VAR}$, Γ is a finite set of PLTL formulae, and $\mathbf{X}_\Gamma$, $\mathbf{Y}_\Gamma$, $\mathbf{U}_\Gamma$ and $\mathbf{S}_\Gamma$ are the stutter-relativized versions of the PLTL temporal modalities. Intuitively, the context modality $\langle C \rangle$ restricts the evaluation of the temporal modalities to the traces associated with the variables in C, while the temporal modalities $\mathbf{X}_\Gamma$, $\mathbf{Y}_\Gamma$, $\mathbf{U}_\Gamma$ and $\mathbf{S}_\Gamma$ are evaluated by a lockstepwise traversal of the Γ-stutter traces associated to the traces assigned to the variables in the current context C. Note that the hyper universal quantifier $\forall x$ can be introduced as an abbreviation: $\forall x.\, \varphi \equiv \neg\exists x.\, \neg\varphi$. For a variable x, we write $\langle x \rangle$ instead of $\langle \{x\} \rangle$. Moreover, we write $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{U}$ and $\mathbf{S}$ instead of $\mathbf{X}_\emptyset$, $\mathbf{Y}_\emptyset$, $\mathbf{U}_\emptyset$ and $\mathbf{S}_\emptyset$, respectively. Furthermore, for a PLTL

formula ψ and a variable x, $\psi[x]$ is the formula obtained from ψ by replacing each proposition p with its x-version $p[x]$. A *sentence* is a formula where each relativized proposition $p[x]$ occurs in the scope of trace quantifier $\exists x$ or $\forall x$, and each temporal modality occurs in the scope of a trace quantifier.

The known logics HyperLTL$_\mathsf{S}$, HyperLTL$_\mathsf{C}$, and simple HyperLTL$_\mathsf{S}$. A formula φ of GHyperLTL$_{\mathsf{S}+\mathsf{C}}$ is in *prenex* form if it is of the form $\mathsf{Q}_1 x_1. \ldots \mathsf{Q}_n x_n. \psi$ where ψ is quantifier-free and $\mathsf{Q}_i \in \{\exists, \forall\}$ for all $i \in [1, n]$. The logics HyperLTL$_\mathsf{S}$ and HyperLTL$_\mathsf{C}$ introduced in [72] correspond to syntactical fragments of GHyperLTL$_{\mathsf{S}+\mathsf{C}}$ where the formulae are in prenex form and past temporal modalities are not used. Moreover, in HyperLTL$_\mathsf{S}$, the context modalities are not allowed, while in HyperLTL$_\mathsf{C}$, the subscript Γ of every temporal modality must be the empty set. Note that in HyperLTL [121], both the context modalities and the temporal modalities where the subscript Γ is not empty are disallowed. Finally, we recall the *simple* fragment of HyperLTL$_\mathsf{S}$ [72], which is more expressive than HyperLTL and is parameterized by a finite set Γ of LTL formulae. The *quantifier-free* formulae of simple HyperLTL$_\mathsf{S}$ for the parameter Γ are defined as Boolean combinations of formulae of the form $\psi[x]$, where ψ is an LTL formula, and formulae ψ_Γ defined by the following grammar:

$$\psi_\Gamma := \top \mid p[x] \mid \neg\psi_\Gamma \mid \psi_\Gamma \vee \psi_\Gamma \mid \mathbf{X}_\Gamma \psi_\Gamma \mid \psi_\Gamma \, \mathbf{U}_\Gamma \, \psi_\Gamma$$

Semantics of GHyperLTL$_{\mathsf{S}+\mathsf{C}}$. Given a formula φ, a set of traces T, a trace assignment Π over T such that $\mathit{Dom}(\Pi)$ contains all the trace variables occurring free in φ, and a context $C \subseteq \mathsf{VAR}$, the satisfaction relation $(\Pi, C) \models_T \varphi$, meaning that the assignment Π over T satisfies φ under the context C, is inductively defined as follows (we again omit the semantics of the Boolean connectives):

$$\begin{aligned}
(\Pi, C) &\models_T p[x] && \Leftrightarrow \Pi(x) = (\sigma, i) \text{ and } p \in \sigma(i) \\
(\Pi, C) &\models_T \exists x. \varphi && \Leftrightarrow \text{ for some } \sigma \in T,\ (\Pi[x \mapsto (\sigma, 0)], C) \models_T \varphi \\
(\Pi, C) &\models_T \langle C' \rangle \varphi && \Leftrightarrow (\Pi, C') \models_T \varphi \\
(\Pi, C) &\models_T \mathbf{X}_\Gamma \varphi && \Leftrightarrow (\mathit{succ}_{(\Gamma,C)}(\Pi), C) \models \varphi \\
(\Pi, C) &\models_T \mathbf{Y}_\Gamma \varphi && \Leftrightarrow \mathit{pred}_{(\Gamma,C)}(\Pi) \neq \mathtt{und} \text{ and } (\mathit{pred}_{(\Gamma,C)}(\Pi), C) \models_T \varphi \\
(\Pi, C) &\models_T \varphi_1 \, \mathbf{U}_\Gamma \, \varphi_2 && \Leftrightarrow \text{for some } i \geq 0\text{: } (\mathit{succ}^i_{(\Gamma,C)}(\Pi), C) \models_T \varphi_2 \text{ and} \\
& && \quad (\mathit{succ}^j_{(\Gamma,C)}(\Pi), C) \models_T \varphi_1 \text{ for all } 0 \leq j < i, \\
(\Pi, C) &\models_T \varphi_1 \, \mathbf{S}_\Gamma \, \varphi_2 && \Leftrightarrow \text{for some } i \geq 0 \text{ such that } \mathit{pred}^i_{(\Gamma,C)}(\Pi) \neq \mathtt{und}\text{:} \\
& && \quad (\mathit{pred}^i_{(\Gamma,C)}(\Pi), C) \models_T \varphi_2 \\
& && \quad \text{and } (\mathit{pred}^j_{(\Gamma,C)}(\Pi), C) \models_T \varphi_1 \text{ for all } 0 \leq j < i
\end{aligned}$$

For a sentence φ and a set of traces T, T is a *model* of φ, written $T \models \varphi$, if $(\Pi_\emptyset, \mathsf{VAR}) \models_T \varphi$ where $\Pi_\emptyset$ is the trace assignment with empty domain.

Fair model checking and standard model checking. For a fragment $\mathcal{F}$ of the logic GHyperLTL$_{\mathsf{S+C}}$, the *fair model checking problem* for $\mathcal{F}$ consists on deciding, given a fair STS $M = \langle V, I, T, F\rangle$ and a sentence φ of $\mathcal{F}$, whether $\mathcal{L}(M) \models \varphi$. The previous problem is simply called *model checking problem* whenever F coincides with the empty set. We consider fair model checking just for technical convenience. For the decidable fragment of GHyperLTL$_{\mathsf{S+C}}$ introduced in Section 8.1.3, we will obtain the same complexity bounds for both fair model checking and standard model checking (see Section 8.2).

8.1.3 The Simple Fragment of GHyperLTL$_{\mathsf{S+C}}$

We introduce now a fragment of GHyperLTL$_{\mathsf{S+C}}$, that we call *simple GHyperLTL$_{\mathsf{S+C}}$*, which syntactically subsumes the simple fragment of HyperLTL$_{\mathsf{S}}$ [72].

In order to define the syntax of simple GHyperLTL$_{\mathsf{S+C}}$, we first consider some shorthands, obtained by a restricted use of the context modalities. The *pointed existential quantifier* $\exists^{\mathsf{P}} x$ and the *pointed universal quantifier* $\forall^{\mathsf{P}} x$ are defined as follows: $\exists^{\mathsf{P}} x.\, \varphi \doteq \exists x.\, \langle x\rangle \mathbf{F} \langle \mathsf{VAR}\rangle \varphi$ and $\forall^{\mathsf{P}} x.\, \varphi ::= \neg \exists^{\mathsf{P}} x.\, \neg\varphi$. Thus the pointed quantifiers quantify on arbitrary pointed traces over the given set of traces and set the global context for the given operand. Formally, $(\Pi, C) \models_T \exists^{\mathsf{P}} x.\, \varphi$ if for some pointed trace (σ, i) with $\sigma \in T$, $(\Pi[x \mapsto (\sigma, i)], \mathsf{VAR}) \models_T \varphi$.
For example, the sentence $\exists x_1.\, \exists^{\mathsf{P}} x_2.\, \big(\bigwedge_{p\in V} \mathbf{G}(p[x_1] \leftrightarrow p[x_2]) \wedge \langle x_2\rangle \mathbf{Y}\top\big)$ asserts that there are two traces σ_1 and σ_2 in the given model s.t. some *proper* suffix of σ_2 coincides with σ_1.

Simple GHyperLTL$_{\mathsf{S+C}}$ is parameterized by a finite set Γ of PLTL formulae. The set of formulae φ_Γ in the Γ-fragment is defined as follows:

$$\varphi_\Gamma := \top \mid \langle x\rangle \psi[x] \mid \neg\varphi_\Gamma \mid \varphi_\Gamma \vee \varphi_\Gamma \mid \exists^{\mathsf{P}} x.\, \varphi_\Gamma \mid \mathbf{X}_\Gamma \varphi_\Gamma \mid \mathbf{Y}_\Gamma \varphi_\Gamma \mid \varphi_\Gamma\, \mathbf{U}_\Gamma\, \varphi_\Gamma \mid \varphi_\Gamma\, \mathbf{S}_\Gamma\, \varphi_\Gamma$$

where ψ is a PLTL formula. Note that $\exists x.\, \varphi$ can be expressed as $\exists^{\mathsf{P}} x.\, (\varphi \wedge \langle x\rangle \neg \mathbf{Y}\top)$. SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ is the class of formulae obtained with the syntax above for a given Γ. Simple GHyperLTL$_{\mathsf{S+C}}$ is the union SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ for all Γ. We say that a formula φ of simple GHyperLTL$_{\mathsf{S+C}}$ is *singleton-free* if for each subformula $\langle x\rangle\psi[x]$ of φ, ψ is an atomic proposition. Evidently, for an atomic proposition p, $\langle x\rangle p[x]$ is equivalent to $p[x]$.

In Section 8.2, we will show that (fair) model checking of simple GHyperLTL$_{\mathsf{S+C}}$ is

decidable. Simple GHyperLTL$_{\mathsf{S+C}}$ can be seen as a very large fragment of GHyperLTL$_{\mathsf{S+C}}$ with a decidable model checking problem which subsumes the simple fragment of HyperLTL$_{\mathsf{S}}$, is closed under Boolean connectives, and allows an unrestricted nesting of temporal modalities. We conjecture (without proof) that this is the largest such sub-class of GHyperLTL$_{\mathsf{S+C}}$ because:

1. Model checking of HyperLTL$_{\mathsf{S}}$ is already undecidable [72] for sentences whose relativized temporal modalities exploit two distinct sets of LTL formulae and, in particular, for two-variable quantifier alternation-free sentences of the form $\exists x_1.\, \exists x_2.\, (\varphi \wedge \mathbf{G}_\Gamma \psi)$, where ψ is a propositional formula, Γ is a nonempty set of propositions, and φ is a quantifier-free formula which use only the temporal modalities $\mathbf{F}_\emptyset$ and $\mathbf{G}_\emptyset$.

2. Model checking of HyperLTL$_{\mathsf{C}}$ is undecidable [74] even for the fragment consisting of two-variable quantifier alternation-free sentences of the form $\exists x_1. \exists x_2.\, \psi_0 \wedge \langle x_2 \rangle \mathbf{F} \langle \{x_1, x_2\} \rangle \psi$, where ψ_0 and ψ are quantifier-free HyperLTL formulae (note that ψ_0 is evaluated in the global context $\langle \{x_1, x_2\} \rangle$).

The second undecidability result suggests a potential extension of the logic simple GHyperLTL$_{\mathsf{S+C}}$; singleton-context subformulae of the form $\langle x \rangle \psi[x]$ can be replaced with *quantifier-free* GHyperLTL$_{\mathsf{S+C}}$ formulae with multiple variables of the form $\langle x \rangle \xi$, where ξ only uses singleton contexts $\langle y \rangle$ and temporal modalities with subscript $\emptyset$. However, we can show that the resulting logic is not more expressive than simple GHyperLTL$_{\mathsf{S+C}}$: a sentence in the considered extension can be translated into an equivalent simple GHyperLTL$_{\mathsf{S+C}}$ sentence though with a non-elementary blowup.

Equally-expressive extension of simple GHyperLTL$_{\mathsf{S+C}}$ We now show that the extension of simple GHyperLTL$_{\mathsf{S+C}}$ is not more expressive than simple GHyperLTL$_{\mathsf{S+C}}$. In the following, a *singleton-context* formula is a quantifier-free GHyperLTL$_{\mathsf{S+C}}$ formula of the form $\langle x \rangle \xi$, where ξ only uses singleton contexts $\langle y \rangle$ and temporal modalities with subscript $\emptyset$. A singleton-context formula is *simple* if it is of the form $\langle x \rangle \psi[x]$ for some PLTL formula ψ. Thus, the considered extension of simple GHyperLTL$_{\mathsf{S+C}}$ is obtained by replacing simple singleton-context sub-formulae with arbitrary singleton-context sub-formulae. Given two quantifier-free GHyperLTL$_{\mathsf{S+C}}$ formulae φ and φ', we say that φ and φ' are *equivalent* if (i) φ and φ' use the same trace variables, and (ii) for each trace assignment Π whose domain contains the variables of φ, $(\Pi, \mathsf{VAR}) \models \varphi$ *iff* $(\Pi, \mathsf{VAR}) \models \varphi'$. In order to show that the considered extension of simple GHyperLTL$_{\mathsf{S+C}}$

has the same expressiveness as simple GHyperLTL$_{\mathsf{S+C}}$, it suffices to show the following result.

Proposition 8.1 *Given a singleton-context formula $\langle x \rangle \xi$, one can construct a Boolean combination φ of simple singleton-context formulae and relativized propositions such that φ and $\langle x \rangle \xi$ are equivalent.*

Proof: For the proof, we need additional definitions. Let Λ be a finite set whose elements are simple singleton-context formulae or relativized atomic propositions. A *guess* for Λ is a mapping $\mathsf{H} : \Lambda \mapsto \{\top, \neg\top\}$ assigning to each formula in Λ a Boolean value ($\top$ for *true*, and $\neg\top$ for *false*). We denote by G_Λ the finite set of guesses for Λ.

Fix a singleton-context formula $\langle x \rangle \xi$ which is not simple. We prove Proposition 8.1 by induction on the nesting depth of the context modalities in ξ.

For the base case, ξ does not contain context modalities. Let Λ be the set of relativized atomic propositions $p[y]$ occurring in ξ such that $y \neq x$. Then, the formula φ satisfying Proposition 8.1 is given by

$$\varphi \doteq \bigvee_{\mathsf{H} \in G_\Lambda} (\xi(\mathsf{H}) \wedge \bigwedge_{\theta \in \{\theta \in \Lambda \mid \mathsf{H}(\theta) = \top\}} \theta \wedge \bigwedge_{\theta \in \{\theta \in \Lambda \mid \mathsf{H}(\theta) = \neg\top\}} \neg\theta)$$

where $\xi(\mathsf{H})$ is obtained from ξ by replacing all occurrences of the sub-formulae $\theta \in \Lambda$ in ξ with $\mathsf{H}(\theta)$. Note that φ is a Boolean combination of simple singleton-context formulae and relativized propositions. Correctness of the construction follows from the fact that by the semantics of GHyperLTL$_{\mathsf{S+C}}$, the position of the pointed trace assigned to a variable y distinct from x remains unchanged during the valuation of the formula ξ within the singleton context $\langle x \rangle$.

For the induction step, assume that ξ contains singleton context modalities. By the induction hypothesis, one can construct a formula ξ' which is a Boolean combination of simple singleton-context formulae and relativized propositions such that $\langle x \rangle \xi$ and $\langle x \rangle \xi'$ are equivalent. Let Λ be the set consisting of the relativized atomic propositions $p[y]$ occurring in ξ' with $y \neq x$ which are not in scope of a context modality, and the sub-formulae of ξ' of the form $\langle y \rangle \theta$ such that $y \neq x$ (note that by hypothesis $\langle y \rangle \theta$ is a simple singleton-context formula). Then, the formula φ satisfying Proposition 8.1 is given by

$$\varphi \doteq \bigvee_{\mathsf{H} \in G_\Lambda} (\xi'(\mathsf{H}) \wedge \bigwedge_{\theta \in \{\theta \in \Lambda \mid \mathsf{H}(\theta) = \top\}} \theta \wedge \bigwedge_{\theta \in \{\theta \in \Lambda \mid \mathsf{H}(\theta) = \neg\top\}} \neg\theta)$$

where $\xi'(\mathsf{H})$ is obtained from ξ' by replacing all occurrences of the sub-formulae $\theta \in \Lambda$

in ξ' with $\mathsf{H}(\theta)$, and by removing the singleton context modality $\langle x \rangle$. Correctness of the construction directly follows from the induction hypothesis and the semantics of GHyperLTL$_{\mathsf{S+C}}$. □

8.1.4 Examples of Specifications in Simple GHyperLTL$_{\mathsf{S+C}}$

We consider some relevant properties which can be expressed in simple GHyperLTL$_{\mathsf{S+C}}$. Simple GHyperLTL$_{\mathsf{S+C}}$ subsumes the simple fragment of HyperLTL$_{\mathsf{S}}$, and this fragment (as shown in [72]) can express relevant information-flow security properties for asynchronous frameworks such as distributed systems or cryptographic protocols. An example is the asynchronous variant of the *noninterference* property, as defined by Goguen and Meseguer [186], which asserts that the observations of low users (users accessing only to public information) do not change when all inputs of high users (users accessing secret information) are removed.

Observational Determinism. An important information-flow property is observational determinism, which states that traces which have the same initial low inputs are indistinguishable to a low user. In an asynchronous setting, a user cannot infer that a transition occurred if consecutive observations remain unchanged. Thus, for instance, observational determinism with equivalence of traces up to stuttering (as formulated in [298]) can be captured by the following simple HyperLTL$_{\mathsf{S}}$ sentence (where LI is the set of propositions describing inputs of low users and LO is set of propositions describing outputs of low users):

$$\forall x. \forall y. \bigwedge_{p \in LI} (p[x] \leftrightarrow p[y]) \rightarrow \mathbf{G}_{LO} \bigwedge_{p \in LO} (p[x] \leftrightarrow p[y])$$

After-initialization Properties. Simple GHyperLTL$_{\mathsf{S+C}}$ also allows to specify complex combinations of asynchronous and synchronous constraints. As an example, we consider the property [189] that for an HyperLTL sentence $\mathsf{Q}_1 x_1. \ldots \mathsf{Q}_n x_n. \psi(x_1, \ldots, x_n)$, the quantifier-free formula $\psi(x_1, \ldots, x_n)$ holds along the traces bound by variables $x_1, \ldots, x_n$ after an initialization phase. Note that this phase can take a different (and unbounded) number of steps on each trace. Let in be a proposition characterizing the initialization phase. The formula $PI(x) \doteq \langle x \rangle (\neg in[x] \wedge (\neg \mathbf{Y} \top \vee \mathbf{Y} in[x]))$ is a simple GHyperLTL$_{\mathsf{S+C}}$ formula that asserts that for the pointed trace (σ, i) assigned to variable x, the position i is the first position of σ following the initialization phase. In other

words, i is the first position at which $\neg in[]$ holds. Then, the previous requirement can be expressed in simple GHyperLTL$_{\mathsf{S+C}}$ as follows:

$$\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n . \big(PI(x_1) \circ_1 (\ldots PI(x_n) \circ_n \psi(x_1, \ldots x_n)) \big)$$

where $\circ_i$ is $\wedge$ if $\mathsf{Q}_i = \exists$ and $\circ_i$ is $\rightarrow$ if $\mathsf{Q}_i = \forall$.

Global Promptness. As another meaningful example, we consider global promptness (in the style of Prompt LTL [217]), where one need to check that there is a uniform bound on the response time in all the traces of the system, that is, "*there is k such that for every trace, each request q is followed by a response p within k steps*". Global promptness is expressible in simple GHyperLTL$_{\mathsf{S+C}}$ as follows:

$$\exists^{\mathsf{P}} x . \big(q[x] \wedge \forall^{\mathsf{P}} y . (q[y] \rightarrow (\neg p[x] \ \mathbf{U} \ p[y])) \big)$$

The previous sentence asserts that there is request (x in the formula) that has the longest response. Note that y is quantified universally (so it can be instantiated to the same trace as x), and that the use of until in $(\neg p[x] \ \mathbf{U} \ p[y]))\big)$ implies that the response $p[y]$ eventually happens. Hence, all requests, including receive a response. Now, the pointed trace (σ_x, i) assigned to x is such that $\sigma_x(i)$ is a request ($q[x]$) and for every pointed trace (σ_y, j) in the model such that $\sigma_y(j)$ is a request ($q[y]$), it holds that (i) the request $\sigma_y(j)$ is followed by a response $\sigma_y(j+k)$ for some $k \geq 0$, and (ii) no response occurs in σ_x in the interval $[i, i+k)$. Therefore, the response time h for x is the smallest h such that $\sigma_x(i+h)$ is a response is a global bound on the response time.

Diagnosability. We now show that simple GHyperLTL$_{\mathsf{S+C}}$ is also able to express *diagnosability* of systems [270, 65, 49] in a general asynchronous setting. In the diagnosis process, faults of a critical system (referred as the plant) are handled by a dedicated module (the *diagnoser*) which runs in parallel with the plant. The diagnoser analyzes the observable information from the plant—made available by predefined sensors—and triggers suitable alarms in correspondence to (typically unobservable) conditions of interest, called faults. An alarm condition specifies the relation (delay) between a given diagnosis condition and the raising of an alarm. A plant $\mathcal{P}$ is *diagnosable* with respect to a given alarm condition α, if there is a diagnoser $\mathcal{D}$ which satisfies α when $\mathcal{D}$ runs in parallel with $\mathcal{P}$.

The given set of propositions V is partitioned into a set of observable propositions Obs and a set of unobservable propositions Int. Two finite traces σ and σ' are *observa-*

tionally equivalent iff the projections of $stfr_{\mathsf{Obs}}(\sigma \cdot P^\omega)$ and $stfr_{\mathsf{Obs}}(\sigma' \cdot (P')^\omega)$ over Obs coincide, where P is the last symbol of σ and similarly P' is the last symbol of σ'. Given a pointed trace (σ, i), i is an *observation point* of σ if either $i = 0$, or $i > 0$ and $\sigma(i-1) \cap \mathsf{Obs} \neq \sigma(i) \cap \mathsf{Obs}$. Then a plant $\mathcal{P}$ can be modeled as an STS $\langle V, I, T, F\rangle$, where T is partitioned into internal transitions (s, s') where $s(\mathsf{Obs}) = s'(\mathsf{Obs})$ and observable transitions where $s(\mathsf{Obs}) \neq s'(\mathsf{Obs})$. A diagnoser $\mathcal{D}$ is modelled as an STS over $V' \supseteq \mathsf{Obs}$ (with $V' \cap \mathsf{Int} = \emptyset$). In the behavioural composition of the plant $\mathcal{P}$ with $\mathcal{D}$, the diagnoser only reacts to the observable transitions of the plant, that is, every transition of the diagnoser is associated with an observable transition of the plant. Simple GHyperLTL$_{\mathsf{S+C}}$ can express diagnosability with *finite delay*, *bounded delay*, or *exact delay* as defined in [65, 49]. Here, we focus for simplicity on finite delay diagnosability. Consider a diagnosis condition specified by a PLTL formula β. A plant $\mathcal{P}$ is *finite delay diagnosable* with respect to β whenever for every pointed trace (σ, i) of $\mathcal{P}$ such that $(\sigma, i) \models \beta$, there exists an observation point $k \geq i$ of σ such that for all pointed traces (σ', k') of $\mathcal{P}$ so that k' is an observation point of σ' and $\sigma[0,k]$ and $\sigma'[0,k']$ are observationally equivalent, it holds that $(\sigma', i') \models \beta$ for some $i' \leq k'$. Finite delay diagnosability w.r.t. β can be expressed in simple GHyperLTL$_{\mathsf{S+C}}$ as follows:

$$\forall^{\mathsf{P}} x.\, \Big(\langle x\rangle \beta[x] \rightarrow \mathbf{F}_{\mathsf{Obs}}\big(\mathsf{ObsPt}(x) \wedge \forall^{\mathsf{P}} y.\, \{(\mathsf{ObsPt}(y) \wedge \theta_{\mathsf{Obs}}(x,y)) \rightarrow \langle y\rangle \mathbf{O}\beta[y]\}\big)\Big)$$

where

$$\theta_{\mathsf{Obs}}(x,y) \doteq \bigwedge_{p \in \mathsf{Obs}} \mathbf{H}_{\mathsf{Obs}}(p[x] \leftrightarrow p[y]) \wedge \mathbf{O}_{\mathsf{Obs}}(\langle x\rangle \neg \mathbf{Y}\top \wedge \langle y\rangle \neg \mathbf{Y}\top)$$

$$\mathsf{ObsPt}(x) \doteq \langle x\rangle(\neg \mathbf{Y}\top \wedge \bigvee_{p \in \mathsf{Obs}} (p[x] \leftrightarrow \neg \mathbf{Y} p[x]))$$

Essentially $\mathsf{ObsPt}(x)$ determines the observation points and θ_{Obs} captures that both traces have the same history of observations. The main formula establishes that if x detects a failure β then there is future observation point in x and for all other traces that are observationally equivalent to x have also detected β.

8.1.5 Expressiveness Issues

In this section, we present some results and conjectures about the expressiveness comparison among GHyperLTL$_{\mathsf{S+C}}$ (which subsumes HyperLTL$_{\mathsf{S}}$ and HyperLTL$_{\mathsf{C}}$), its sim-

ple fragment and the logic HyperLTL$_\mathsf{S}$. We also consider the logics for linear-time hyperproperties based on the equal-level predicate whose most powerful representative is S1S[E]. Recall that the first-order fragment FO[<,E] of S1S[E] is already strictly more expressive than HyperLTL [163] and, unlike S1S[E], its model-checking problem is decidable [124]. Moreover, we show that GHyperLTL$_{\mathsf{S+C}}$ and its simple fragment represent a unifying framework in the linear-time setting for specifying both hyperproperties and the knowledge modalities of epistemic temporal logics under both the synchronous and asynchronous perfect recall semantics.

Our expressiveness results about linear-time hyper logics can be summarized as follows.

Theorem 8.1 *The following hold:*

- *GHyperLTL$_{\mathsf{S+C}}$ is more expressive than HyperLTL$_\mathsf{S}$, simple GHyperLTL$_{\mathsf{S+C}}$, and FO[<,E].*
- *Simple GHyperLTL$_{\mathsf{S+C}}$ is more expressive than simple HyperLTL$_\mathsf{S}$.*
- *Simple GHyperLTL$_{\mathsf{S+C}}$ and HyperLTL$_\mathsf{S}$ are expressively incomparable.*
- *Simple GHyperLTL$_{\mathsf{S+C}}$ and S1S[E] are expressively incomparable.*

Proof: We first show that there are hyperproperties that can be expressed in simple GHyperLTL$_{\mathsf{S+C}}$ but not in HyperLTL$_\mathsf{S}$ and in S1S[E]. Given a sentence φ, the *trace property denoted by* φ is the set of traces σ such that the singleton set of traces $\{\sigma\}$ satisfies φ. It is known that HyperLTL$_\mathsf{S}$ and S1S[E] capture only *regular* trace properties [74]. In contrast, simple GHyperLTL$_{\mathsf{S+C}}$ can express powerful non-regular trace properties. For example, consider the so called *suffix property* over $V = \{p\}$: a trace σ satisfies the suffix property if there exists a proper suffix σ^k of σ for some $k > 0$ such that $\sigma^k = \sigma$. This non-regular trace property can be expressed in SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ as follows:

$$\forall x_1.\, \forall x_2.\, \bigwedge_{p \in V} \mathbf{G}(p[x_1] \leftrightarrow p[x_2]) \;\wedge\; \forall x_1.\, \exists^{\mathsf{P}} x_2.\, \big(\bigwedge_{p \in V} \mathbf{G}(p[x_1] \leftrightarrow p[x_2]) \wedge \langle x_2 \rangle \mathbf{Y} \top\big)$$

The first conjunct asserts that each model is a singleton, and the second conjunct requires that for the unique trace σ in a model, there is $k > 0$ such that $\sigma(i) = \sigma(i+k)$ for all $i \geq 0$.

Next, we observe that FO[<,E] can be easily translated into GHyperLTL$_{\mathsf{S+C}}$, since the pointer quantifiers of GHyperLTL$_{\mathsf{S+C}}$ correspond to the quantifiers of FO[<,E]. Moreover, the predicate $x \leq x'$ of FO[<,E], expressing that for the pointed traces (σ, i) and (σ', i') bound to x and x', $\sigma = \sigma'$ and $i \leq i'$, can be easily captured in GHyperLTL$_{\mathsf{S+C}}$. This is also the case for the equal-level predicate $\mathsf{E}(x, x')$, which can be expressed as $\langle\{x, x'\}\rangle \mathbf{O}(\langle x\rangle \neg \mathbf{Y} \top \wedge \langle x'\rangle \neg \mathbf{Y} \top)$.

In Section 8.2 we show that model checking of simple GHyperLTL$_{\mathsf{S+C}}$ is decidable. Thus, since model checking of both HyperLTL$_{\mathsf{S}}$ and S1S[E] are undecidable [72, 124] and GHyperLTL$_{\mathsf{S+C}}$ subsumes HyperLTL$_{\mathsf{S}}$, by the previous argumentation, the theorem follows. □

It remains an open question whether FO[<,E] is subsumed by simple GHyperLTL$_{\mathsf{S+C}}$. We conjecture that neither HyperLTL$_{\mathsf{C}}$ nor the fix-point calculus H$_\mu$ [189] (which captures both HyperLTL$_{\mathsf{C}}$ and HyperLTL$_{\mathsf{S}}$ [74]) subsume simple GHyperLTL$_{\mathsf{S+C}}$. The motivation for our conjecture is that H$_\mu$ sentences consist of a prefix of quantifiers followed by a quantifier-free formula where quantifiers range over *initial* pointed traces $(\sigma, 0)$. Thus, unlike simple GHyperLTL$_{\mathsf{S+C}}$, H$_\mu$ cannot express requirements which relate at some point in time an unbounded number of traces. Diagnosability (see Subsection 8.1.4) falls in this class of requirements. It is known that the following property, which can be easily expressed in simple GHyperLTL$_{\mathsf{S+C}}$, is not definable in HyperLTL [71]: for some $i > 0$, every trace in the given set of traces does not satisfy proposition p at position i. We conjecture that similarly to HyperLTL, such a property (and diagnosability as well) cannot be expressed in H$_\mu$.

Epistemic Temporal Logic KLTL and its relation with GHyperLTL$_{\mathsf{S+C}}$. The logic KLTL [190] is a well-known extension of LTL obtained by adding the unary knowledge modalities $\mathbf{K}_a$, where a ranges over a finite set Agts of agents, interpreted under the synchronous or asynchronous (perfect recall) semantics. The semantics is given with respect to an observation map $\mathsf{Obs} : \mathsf{Agts} \mapsto 2^V$ that assigns to each agent a the set of propositions which agent a can observe. Given two finite traces σ and σ' and $a \in \mathsf{Agts}$, σ and σ' are *synchronously equivalent for agent* a, written $\sigma \sim_a^{sy} \sigma'$, if the projections of σ and σ' over $\mathsf{Obs}(a)$ coincide. The finite traces σ and σ' are *asynchronously equivalent for agent* a, written $\sigma \sim_a^{as} \sigma'$, if the projections of $stfr_{\mathsf{Obs}(a)}(\sigma \cdot P^\omega)$ and $stfr_{\mathsf{Obs}(a)}(\sigma' \cdot (P')^\omega)$ over $\mathsf{Obs}(a)$ coincide, where P is the last symbol of σ and P' is the last symbol of σ'. For a set of traces $\mathcal{L}$ and a pointed trace (σ, i) over $\mathcal{L}$, the semantics of the knowledge modalities is as follows, where $\sim_a$ is $\sim_a^{sy}$ under the synchronous semantics, and $\sim_a^{as}$

otherwise: $(\sigma, i) \models_{\mathcal{L},\mathsf{Obs}} \mathbf{K}_a\,\varphi \Leftrightarrow$ for all pointed traces (σ', i') on $\mathcal{L}$ such that $\sigma[0,i] \sim_a \sigma'[0,i']$, $(\sigma', i') \models_{\mathcal{L},\mathsf{Obs}} \varphi$.

We say that $\mathcal{L}$ *satisfies* φ *w.r.t. the observation map* Obs, written $\mathcal{L} \models_{\mathsf{Obs}} \varphi$, if for all traces $\sigma \in \mathcal{L}$, $(\sigma, 0) \models_{\mathcal{L},\mathsf{Obs}} \varphi$. The logic KLTL can be easily embedded into $\text{GHyperLTL}_{\mathsf{S+C}}$. In particular, the following holds

Theorem 8.2 *Given an observation map* Obs *and a KLTL formula* ψ *over* V*, one can construct in linear time a* $\text{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ *sentence* $\varphi_\emptyset$ *and a* $\text{GHyperLTL}_{\mathsf{S+C}}$ *sentence* φ *such that* $\varphi_\emptyset$ *is equivalent to* ψ *w.r.t.* Obs *under the synchronous semantics and* φ *is equivalent to* ψ *w.r.t.* Obs *under the asynchronous semantics. Moreover,* φ *is a simple* $\text{GHyperLTL}_{\mathsf{S+C}}$ *sentence if* ψ *is in the single-agent fragment of KLTL.*

Proof: We focus on the construction of the $\text{GHyperLTL}_{\mathsf{S+C}}$ sentence φ capturing the KLTL formula ψ under the asynchronous semantics w.r.t. the given observation map Obs. The construction of the $\text{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ sentence $\varphi_\emptyset$ capturing ψ under the synchronous semantics w.r.t. Obs is similar. We inductively define a mapping $\mathsf{T}_{\mathsf{Obs}}$ assigning to each pair (ϕ, x) consisting of a KLTL formula ϕ and a trace variable $x \in \mathsf{VAR}$, a $\text{GHyperLTL}_{\mathsf{S+C}}$ formula $\mathsf{T}_{\mathsf{Obs}}(\phi, x)$. Intuitively, x is associated to the current evaluated pointed trace. The mapping $\mathsf{T}_{\mathsf{Obs}}$ is homomorphic w.r.t. the Boolean connectives and the LTL temporal modalities (i.e., $\mathsf{T}(\mathcal{O}\phi, x) \doteq \mathcal{O}\mathsf{T}(\phi, x)$ for each $\mathcal{O} \in \{\neg, \mathbf{X}\}$ and $\mathsf{T}(\phi_1\,\mathcal{O}\,\phi_2, x) \doteq \mathsf{T}(\phi_1, x)\,\mathcal{O}\,\mathsf{T}(\phi_2, x)$ for each $\mathcal{O} \in \{\vee, \mathbf{U}\}$) and is defined as follows when the given KLTL formula is an atomic proposition or its root operator is a knowledge modality:

- $\mathsf{T}(p, x) \doteq p[x]$;
- $\mathsf{T}(\mathbf{K}_a\phi, x) \doteq \forall^{\mathsf{P}} y.\,(\theta_{\mathsf{Obs}}(a, x, y) \rightarrow \mathsf{T}_{\mathsf{Obs}}(\phi, y))$ where $y \neq x$ and $\theta_{\mathsf{Obs}}(a, x, y) \doteq \bigwedge_{p \in \mathsf{Obs}(a)} \mathbf{H}_{\mathsf{Obs}(a)}(p[x] \leftrightarrow p[y]) \wedge \mathbf{O}_{\mathsf{Obs}(a)}(\langle x\rangle\neg\mathbf{Y}\top \wedge \langle y\rangle\neg\mathbf{Y}\top)$

Intuitively, $\theta_{\mathsf{Obs}}(a, x, y)$ asserts that for the pointed traces (σ, i) and (σ', i') bound to the trace variables x and y, respectively, $\sigma[0,i]$ and $\sigma[0,i']$ are asynchronously equivalent for agent a w.r.t. the given observation map Obs. Note that $\mathsf{T}_{\mathsf{Obs}}(\phi, x)$ is a simple $\text{GHyperLTL}_{\mathsf{S+C}}$ formula if ϕ is in the one-agent fragment of KLTL. Let $\mathcal{L}$ be a set of traces, $x \in \mathsf{VAR}$, (σ, i) be a pointed trace over $\mathcal{L}$, and Π be a trace assignment over $\mathcal{L}$ such that $x \in \mathit{Dom}(\Pi)$ and $\Pi(x) = (\sigma, i)$. By a straightforward induction on the structure of a KLTL formula φ, one can show that $(\sigma, i) \models_{\mathcal{L},\mathsf{Obs}} \phi$ if and only if $(\Pi, \mathsf{VAR}) \models_{\mathcal{L}} \mathsf{T}_{\mathsf{Obs}}(\phi, x)$. Hence, the desired $\text{GHyperLTL}_{\mathsf{S+C}}$ sentence φ is given by $\forall x.\,\mathsf{T}_{\mathsf{Obs}}(\psi, x)$ and we are done. □

8.2 Decidability of Model Checking against Simple GHyperLTL$_{\mathsf{S+C}}$

In this section, we show that (fair) model checking against simple GHyperLTL$_{\mathsf{S+C}}$ is decidable. We first prove the result for the fragment SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ of simple GHyperLTL$_{\mathsf{S+C}}$ by a linear-time reduction to satisfiability of *full* Quantified Propositional Temporal Logic (QPTL, for short) [275], where the latter extends PLTL by quantification over propositions. Then, we show that (fair) model checking of simple GHyperLTL$_{\mathsf{S+C}}$ can be reduced in time singly exponential in the size of the formula to fair model checking of SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$. To simplify the proofs of Propositions of Theorems presented in this section (contrary to the rest of thesis), we adopt an explicit state representations of our systems considering *Kripke* structures and Buchi Automata. Moreover, propositions $v \in V$ are denoted as true by saying $v \in \sigma(i)$ and, the set of *true* variables of a pointed trace (σ, i) would be represented by $\sigma(i)$. This notation can be seen as a simplified version for Boolean systems of the general notion of trace introduced in Chapter 2.

Definition 8.2 (Kripke structure) *We define the dynamic behaviour of reactive systems by* Kripke structures $\mathcal{K} = \langle S, S_0, E, Lab\rangle$ *over a finite set* V *of atomic propositions, where* S *is a set of states,* $S_0 \subseteq S$ *is the set of initial states,* $E \subseteq S \times S$ *is a transition relation which is total in the first argument (i.e., for each* $s \in S$ *there is* $s' \in S$ *with* $(s, s') \in E$*), and* $Lab : S \to 2^V$ *is a labeling map assigning to each state* s *the set of propositions holding at* s*. The Kripke structure* $\mathcal{K}$ *is finite if* S *is finite. A* path π *of* $\mathcal{K}$ *is an infinite word* $\pi = s_0, s_1, \ldots$ *over* S *such that* $s_0 \in S_0$ *and for all* $i \geq 0$*,* $(s_i, s_{i+1}) \in E$*. The path* $\pi = s_0, s_1, \ldots$ *induces the trace* $Lab(s_0)Lab(s_1)\ldots$*. A* trace of $\mathcal{K}$ *is a trace induced by some path of* $\mathcal{K}$*. We denote by* $T(\mathcal{K})$ *the set of traces of* $\mathcal{K}$*. A* finite path *of* $\mathcal{K}$ *is a non-empty infix of some path of* $\mathcal{K}$*. We also consider* fair finite Kripke structures $(\mathcal{K}, F)$*, that is, finite Kripke structures* $\mathcal{K}$ *equipped with a subset* F *of* $\mathcal{K}$*-states. A path* π *of* $\mathcal{K}$ *is* F-fair *if* π *visits infinitely many times some state in* F*. We denote by* $T(\mathcal{K}, F)$ *the set of traces of* $\mathcal{K}$ *associated with the* F*-fair paths of* $\mathcal{K}$*.*

The syntax of QPTL formulae φ over a finite set V of atomic propositions is as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{Y}\varphi \mid \varphi\, \mathbf{U}\, \varphi \mid \varphi\, \mathbf{S}\, \varphi \mid \exists p\, .\varphi$$

where $p \in V$ and $\exists p$ is the propositional existential quantifier. A QPTL formula φ is a *sentence* if each proposition p occurs in the scope of a quantifier binding p and each

temporal modality occurs in the scope of a quantifier. By introducing $\wedge$ and the operators $\mathbf{R}$ (*release*, dual of $\mathbf{U}$), $\mathbf{T}$ (*past release*, dual of $\mathbf{S}$) and $\forall p$ (propositional universal quantifier), every QPTL formula can be converted into an equivalent formula in *negation normal form*, where negation only appears in front of atomic propositions. QPTL formulae are interpreted over pointed traces (σ, i) over V. All QPTL temporal operators have the same semantics as in PLTL. The semantics of propositional quantification is as follows:

$$(\sigma, i) \models \exists p.\varphi \;\Leftrightarrow\; \text{there is a trace } \sigma' \text{ such that } \sigma =_{V\setminus\{p\}} \sigma' \text{ and } (\sigma', i) \models \varphi$$

where $\sigma =_{V\setminus\{p\}} \sigma'$ means that the projections of σ and σ' over $V \setminus \{p\}$ coincide.A formula φ is satisfiable if $(\sigma, 0) \models \varphi$ for some trace σ. We now give a generalization of the standard notion of alternation depth between existential and universal quantifiers which corresponds to the one given in [260] for HyperCTL*. Our notion takes into account also the occurrences of temporal modalities between quantifier occurrences, but the nesting depth of temporal modalities is not considered (intuitively, it is collapsed to one). Formally, the *strong alternation depth* $sad(\varphi)$ of a QPTL formula φ in negation normal form is inductively defined as follows, where an existential formula is a formula of the form $\exists p.\,\psi$, a universal formula is of the form $\forall p.\,\psi$, and for a formula ψ, $\widetilde{\psi}$ denotes the negation normal form of $\neg\psi$:

- For $\varphi = p$ and $\varphi = \neg p$ for a given $p \in V$: $sad(\varphi) \doteq 0$.
- For $\varphi = \varphi_1 \vee \varphi_2$ and for $\varphi = \varphi_1 \wedge \varphi_2$: $sad(\varphi) \doteq \max(\{sad(\varphi_1), sad(\varphi_2)\})$.
- For $\varphi = \exists p.\,\varphi_1$: if there is no universal sub-formula $\forall\psi$ of φ_1 such that $sad(\forall\psi) = sad(\varphi_1)$, then $sad(\varphi) \doteq sad(\varphi_1)$. Otherwise, $sad(\varphi) \doteq sad(\varphi_1) + 1$.
- For $\varphi = \forall p.\,\varphi_1$: $sad(\varphi) \doteq sad(\exists p.\,\widetilde{\varphi_1})$.
- For $\varphi = \mathbf{X}\varphi_1$ or $\varphi = \mathbf{Y}\varphi_1$: $sad(\varphi) \doteq sad(\varphi_1)$.
- For $\varphi = \varphi_1\,\mathbf{U}\,\varphi_2$ or $\varphi = \varphi_1\,\mathbf{S}\,\varphi_2$: let h be the maximum over the strong alternation depths of the universal and existential sub-formulae of φ_1 and φ_2 (the maximum of the empty set is 0). If the following conditions are met, then $sad(\varphi) \doteq h$; otherwise, $sad(\varphi) \doteq h + 1$:
  - there is no existential or universal sub-formula ψ of φ_1 with $sad(\psi) = h$;
  - there is no universal sub-formula ψ of φ_2 with $sad(\psi) = h$;

  - no existential formula ψ with $sad(\psi) = h$ occurs in the left operand (resp., right operand) of a sub-formula of φ_2 of the form $\psi_1 \mathcal{O} \psi_2$, where $\mathcal{O} \in \{\mathbf{U}, \mathbf{S}\}$ (resp., $\mathcal{O} \in \{\mathbf{R}, \mathbf{T}\}$).

- Finally, for $\varphi = \varphi_1 \,\mathbf{R}\, \varphi_2$ or $\varphi = \varphi_1 \,\mathbf{T}\, \varphi_2$: $sad(\varphi) \doteq sad(\widetilde{\varphi})$.

For example, $sad(\exists p.(p \,\mathbf{U}\, \exists q.q)) = 0$ and $sad(\exists p.(\exists p.p \,\mathbf{U}\, q)) = 1$. The strong alternation depth of an arbitrary QPTL formula corresponds to the one of its negation normal form. The strong alternation depth of a simple GHyperLTL$_{\mathsf{S+C}}$ formula is defined similarly but we replace quantification over propositions with quantification over trace variables.

8.2.1 (Fair) Model checking against SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$.

We provide now linear-time reductions of (fair) model checking against SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ to (and from) satisfiability of QPTL which preserve the strong alternation depth. We start with the reduction of (fair) model checking SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ to QPTL satisfiability.

Theorem 8.3 *Given a fair finite Kripke structure $(\mathcal{K}, F)$ and a SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ sentence φ, one can construct in linear time a QPTL sentence ψ with the same strong alternation depth as φ such that ψ is satisfiable if and only if $T(\mathcal{K}, F) \models \varphi$.*

We first give a sketch of the proof.

Proof: [Sketched proof] Let $\mathcal{K} = \langle S, S_0, E, Lab \rangle$. In the reduction of model checking $(\mathcal{K}, F)$ against φ to QPTL satisfiability, we need to merge multiple traces into a unique trace where just one position is considered at any time. An issue is that the hyper quantifiers range over arbitrary pointed traces so that the positions of the different pointed traces in the current trace assignment do not necessarily coincide (intuitively, the different pointed traces are not aligned with respect to the relative current positions). However, we can solve this issue because the offsets between the positions of the pointed traces in the current trace assignment remain constant during the evaluation of the temporal modalities. In particular, assume that (σ, i) is the first pointed trace selected by a hyper quantifier during the evaluation along a path in the syntax tree of φ. We encode σ by keeping track also of the variable x to which (σ, i) is bound and the F-fair path of $\mathcal{K}$ whose associated trace is σ. Let (σ', i') be another pointed trace introduced by another hyper quantifier y during the evaluation of φ. If $i' < i$, we consider an encoding of σ' which is similar to the previous encoding but we precede it with a *padding prefix* of length $i - i'$ of the form $\{\#_{\overrightarrow{y}}\}^{i-i'}$. The arrow $\rightarrow$ indicates

that the encoding is along the *forward direction*. Now, assume that $i' > i$. In this case, the encoding of σ' is the merging of two encodings over disjoint sets of propositions: one along the forward direction which encodes the suffix $(\sigma')^{i'-i}$ and another one along the *backward direction* which is of the form $\{\#_{\overleftarrow{y}}\} \cdot \rho \cdot \{\#_{\overleftarrow{y}}\}^\omega$, where ρ is a backward encoding of the *reverse* of the prefix of σ' of length $i'-i$. In such a way, the encodings of the pointed traces later introduced in the evaluation of φ are aligned with the reference pointed trace (σ, i). Since the positions in the backward direction overlap some positions in the forward direction, in the translation, we keep track of whether the current position refers to the forward or to the backward direction. □

We now proceed to the full proof.

Proof: Let $\mathcal{K} = \langle S, S_0, E, Lab\rangle$. The high-level description of the reduction of model checking $(\mathcal{K}, F)$ against φ to QPTL satisfiability has been given in Section 8.2. Here, we provide the details of the reduction. We consider a new finite set V' of atomic propositions defined as follows:

$$V' := \bigcup_{x \in \mathsf{VAR}} V_x \cup S_x \cup \{\#_{\overleftarrow{x}}, \#_{\overrightarrow{x}}\}$$
$$V_x := \{p_{\overleftarrow{x}}, p_{\overrightarrow{x}} \mid p \in V\} \text{ and } S_x := \{s_{\overleftarrow{x}}, s_{\overrightarrow{x}} \mid s \in S\}$$

Let $V_{\overleftarrow{x}} := \{p_{\overleftarrow{x}} \mid p \in V\}$, $V_{\overrightarrow{x}} := \{p_{\overrightarrow{x}} \mid p \in V\}$, $S_{\overleftarrow{x}} := \{s_{\overleftarrow{x}} \mid s \in S\}$, and $S_{\overrightarrow{x}} := \{s_{\overrightarrow{x}} \mid s \in S\}$. Thus, we associate to each variable $x \in \mathsf{VAR}$ and atomic proposition $p \in V$, two fresh atomic propositions $p_{\overleftarrow{x}}$ and $p_{\overrightarrow{x}}$, and to each variable $x \in \mathsf{VAR}$ and state s of $\mathcal{K}$, two fresh atomic proposition $s_{\overleftarrow{x}}$ and $s_{\overrightarrow{x}}$. Moreover, for each $x \in \mathsf{VAR}$, $\#_{\overleftarrow{x}}$ and $\#_{\overrightarrow{x}}$ are exploited as padding propositions.

Encoding of paths and pointed paths of $\mathcal{K}$. For each $x \in \mathsf{VAR}$ and an infinite word $\pi = s_0, s_1, \ldots$ over S, we denote by $\overrightarrow{\sigma}(x, \pi)$ the trace over $V_{\overrightarrow{x}} \cup S_{\overrightarrow{x}}$ defined as follows for all $i \geq 0$:

$$\overrightarrow{\sigma}(x, \pi)(i) := \{(s_i)_{\overrightarrow{x}}\} \cup \{p_{\overrightarrow{x}} \mid p \in Lab(s_i)\}$$

For each finite word π over S, let $\overleftarrow{\sigma}(x, \pi)$ be the trace over $V_{\overleftarrow{x}} \cup S_{\overleftarrow{x}}$ given by $\{\#_{\overleftarrow{x}}\} \cdot w \cdot \{\#_{\overleftarrow{x}}\}^\omega$, where $|w| = |\pi|$ and for each $0 \leq i < |\pi|$, $w(i) = \{(s_i)_{\overleftarrow{x}}\} \cup \{p_{\overleftarrow{x}} \mid p \in Lab(s_i)\}$. Now, we define the forward and backward encodings of a path π of $\mathcal{K}$. For each $k \in \mathbb{N}$, the *forward x-encoding of π with offset k* is the trace over $V_x \cup S_x \cup \{\#_{\overleftarrow{x}}, \#_{\overrightarrow{x}}\}$, denoted by $\overrightarrow{\sigma}(x, \pi, k)$, defined as follows:

- the projection of $\overrightarrow{\sigma}(x, \pi, k)$ over $V_{\overrightarrow{x}} \cup S_{\overrightarrow{x}} \cup \{\#_{\overrightarrow{x}}\}$ is $\{\#_{\overrightarrow{x}}\}^k \cdot \overrightarrow{\sigma}(x, \pi)$;

- the projection of $\overrightarrow{\sigma}(x,\pi,k)$ over $V_{\overleftarrow{x}} \cup S_{\overleftarrow{x}} \cup \{\#_{\overleftarrow{x}}\}$ is $\{\#_{\overleftarrow{x}}\}^\omega$.

For each $k > 0$, the *backward x-encoding of π with offset $k > 0$*, is the trace over $V_x \cup S_x \cup \{\#_{\overleftarrow{x}}, \#_{\overrightarrow{x}}\}$, denoted by $\overleftarrow{\sigma}(x,\pi,k)$, defined as follows, where $\pi^R_{\leq k}$ denotes the reverse of the prefix $\pi(0)\dots\pi(k-1)$ of π until position $k-1$:

- the projection of $\overrightarrow{\sigma}(x,\pi,k)$ over $V_{\overrightarrow{x}} \cup S_{\overrightarrow{x}} \cup \{\#_{\overrightarrow{x}}\}$ is $\overrightarrow{\sigma}(x,\pi^k)$;
- the projection of $\overrightarrow{\sigma}(x,\pi,k)$ over $V_{\overleftarrow{x}} \cup S_{\overleftarrow{x}} \cup \{\#_{\overleftarrow{x}}\}$ is $\overleftarrow{\sigma}(x,\pi^R_{\leq k})$.

Claim 1. For all $x \in \mathsf{VAR}$, one can construct in linear time two PLTL formulae $\theta(x,\rightarrow)$ and $\theta(x,\leftarrow)$ such that for all pointed traces (σ,i) over V', we have:

- $(\sigma,i) \models \theta(x,\rightarrow)$ *iff* there is a F-fair path π of $\mathcal{K}$ such that the projection of σ over $S_x \cup V_x \cup \{\#_{\overleftarrow{x}}, \#_{\overrightarrow{x}}\}$ is the forward x-encoding of π for some offset $k \geq 0$.
- $(\sigma,i) \models \theta(x,\leftarrow)$ *iff* there is a F-fair path π of $\mathcal{K}$ such that the projection of σ over $S_x \cup V_x \cup \{\#_{\overleftarrow{x}}, \#_{\overrightarrow{x}}\}$ is the backward x-encoding of π for some offset $k > 0$.

We give the details on the construction of the PLTL formula $\theta(x,\leftarrow)$ in Claim 1 (the construction of $\theta(x,\rightarrow)$ is similar). For each state $s \in S$, $E(s)$ denotes the set of successors of s in $\mathcal{K}$, while $E^{-1}(s)$ denotes the set of predecessors of s in $\mathcal{K}$. The PLTL formula $\theta(x,\leftarrow)$ is defined as follows:

$$
\begin{aligned}
\theta(x,\leftarrow) \;:=\; & \mathbf{O}\big((\neg\mathbf{Y}\top) \wedge \#_{\overleftarrow{x}} \wedge \mathbf{G}\neg\#_{\overrightarrow{x}} \wedge \bigvee_{s\in S}[\xi(x,s,\rightarrow) \wedge \bigvee_{s'\in E^{-1}(s)} \mathbf{X}\xi(x,s',\leftarrow)]\big) \\
\xi(x,s,\rightarrow) \;:=\; & s_{\overrightarrow{x}} \wedge \bigvee_{s'\in F} \mathbf{GF}\, s'_{\overrightarrow{x}} \;\wedge\; \bigwedge_{s'\in S} \mathbf{G}\Big(s'_{\overrightarrow{x}} \rightarrow \\
& \Big[\bigwedge_{p\in Lab(s')} p_{\overrightarrow{x}} \wedge \bigwedge_{p\in V\setminus Lab(s')} \neg p_{\overrightarrow{x}} \wedge \bigwedge_{s''\in S\setminus\{s'\}} \neg s''_{\overrightarrow{x}} \wedge \bigvee_{s''\in E(s')} \mathbf{X}\, s''_{\overrightarrow{x}}\Big]\Big) \\
\xi(x,s,\leftarrow) \;:=\; & s_{\overleftarrow{x}} \wedge \mathbf{FG}\#_{\overleftarrow{x}} \wedge \bigvee_{s'\in S_0} \mathbf{F}(s'_{\overleftarrow{x}} \wedge \mathbf{X}\#_{\overleftarrow{x}}) \;\wedge\; \mathbf{HG}[\#_{\overleftarrow{x}} \rightarrow \bigwedge_{q\in V\cup S} \neg q_{\overleftarrow{x}}] \;\wedge \\
& \bigwedge_{s'\in S} \mathbf{G}\Big(s'_{\overleftarrow{x}} \longrightarrow \Big[\bigwedge_{p\in Lab(s')} p_{\overleftarrow{x}} \;\wedge \bigwedge_{p\in V\setminus Lab(s')} \neg p_{\overleftarrow{x}} \wedge \neg\#_{\overleftarrow{x}} \wedge \\
& \bigwedge_{s''\in S\setminus\{s'\}} \neg s''_{\overleftarrow{x}} \wedge \mathbf{X}\big(\#_{\overleftarrow{x}} \vee \bigvee_{s''\in E^{-1}(s')} s''_{\overleftarrow{x}}\big)\Big]\Big)
\end{aligned}
$$

We now extend the notions of backward and forward encodings of paths of $\mathcal{K}$ to pointed paths (π,i) of $\mathcal{K}$. For each $k \in \mathbb{N}$, the *forward x-encoding of (π,i) with offset k* is the pointed trace given by $(\overrightarrow{\sigma}(x,\pi,k), i+k)$. We say that the position $i+k$ is

in *forward mode* in the encoding. For each $k > 0$, the *backward x-encoding of* (π, i) *with offset* k is the pointed trace given by $(\overleftarrow{\sigma}(x, \pi, k), j)$, where $j = i - k$ if $i \geq k$, and $j = k - i$ otherwise. In the first case, we say that position j is in *forward mode* in the encoding, and the second case, we say that position j is in *backward mode* in the encoding. Intuitively, when (σ, ℓ) is a backward or forward encoding of a pointed path (π, i) of $\mathcal{K}$, then ℓ represents the position in the encoding associated to position i of π.

Encoding of $(\mathcal{K}, F)$-path assignments. Let Π be a $(\mathcal{K}, F)$-path assignment. A pointed trace (σ, i) over V' is a *forward encoding of* Π if for each $x \in \mathsf{VAR}$, the following holds, where σ_x denotes the projection of σ over $S_x \cup V_x \cup \{\#_{\overleftarrow{x}}, \#_{\overrightarrow{x}}\}$:

- if $x \notin \mathit{Dom}(\Pi)$, then σ_x is $\emptyset^\omega$;
- if $x \in \mathit{Dom}(\Pi)$, then (σ_x, i) is a forward or backward encoding of $\Pi(x)$ where i is in forward mode in the encoding.

When $\mathit{Dom}(\Pi) \neq \emptyset$, we also consider the notion of a *backward encoding* (σ, i) of Π which is defined as a forward encoding but we require that for each $x \in \mathit{Dom}(\Pi)$, (σ_x, i) is a backward encoding of $\Pi(x)$ where i is in backward mode in the encoding. Note that in this case, we have that $i \geq 1$ holds.

Let Π be a $(\mathcal{K}, F)$-path assignment and Π' be the trace assignment over $T(\mathcal{K}, F)$ obtained from Π in the obvious way. For each $\ell \geq 0$, we write $\mathit{succ}^\ell(\Pi)$ to mean $\mathit{succ}^\ell_{(\emptyset, \mathsf{VAR})}(\Pi')$ and $\mathit{pred}^\ell(\Pi)$ to mean $\mathit{pred}^\ell_{(\emptyset, \mathsf{VAR})}(\Pi')$. By construction, we easily obtain the following result, where $\mathsf{Halt}_\rightarrow \doteq \bigvee_{x \in \mathsf{VAR}} \#_{\overrightarrow{x}}$ and $\mathsf{Halt}_\leftarrow \doteq \bigvee_{x \in \mathsf{VAR}} \#_{\overleftarrow{x}}$.

Claim 2. Let Π be a $(\mathcal{K}, F)$-path assignment and $\ell \geq 0$.

- If (σ, i) is a forward encoding of Π, then:
  - $(\sigma, i + \ell)$ is a forward encoding of $\mathit{succ}^\ell(\Pi)$;
  - if $\ell \leq i$, then $\mathit{pred}^\ell(\Pi) \neq \mathtt{und}$ iff $(\sigma, i - \ell) \models \neg\mathsf{Halt}_\rightarrow$. Moreover, if $\mathit{pred}^\ell(\Pi) \neq \mathtt{und}$, then $(\sigma, i - \ell)$ is a forward encoding of $\mathit{pred}^\ell(\Pi)$;
  - if $\ell > i$, then $\mathit{pred}^\ell(\Pi) \neq \mathtt{und}$ iff $(\sigma, \ell - i) \models \neg\mathsf{Halt}_\leftarrow$. Moreover, if $\mathit{pred}^\ell(\Pi) \neq \mathtt{und}$, then $(\sigma, \ell - i)$ is a backward encoding of $\mathit{pred}^\ell(\Pi)$.
- If (σ, i) is a backward encoding of Π, then:
  - If $\ell < i$, $(\sigma, i - \ell)$ is a backward encoding of $\mathit{succ}^\ell(\Pi)$;
  - if $\ell \geq i$, $(\sigma, \ell - i)$ is a forward encoding of $\mathit{succ}^\ell(\Pi)$;

– $\mathit{pred}^{\ell}(\Pi) \neq \mathtt{und}$ iff $(\sigma, i+\ell) \models \neg\mathsf{Halt}_{\leftarrow}$. Moreover, if $\mathit{pred}^{\ell}(\Pi) \neq \mathtt{und}$, then $(\sigma, i+\ell)$ is a backward encoding of $\mathit{pred}^{\ell}(\Pi)$.

Reduction to QPTL satisfiability. We define by structural induction a mapping $\mathsf{T} : \{\leftarrow, \rightarrow\} \times \text{SHyperLTL}^{\emptyset}_{\mathsf{S+C}} \rightarrow \text{QPTL}$ associating to each pair (dir, ϕ) consisting of a direction $\mathit{dir} \in \{\leftarrow, \rightarrow\}$ and a SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ formula ϕ over V and VAR a QPTL formula $\mathsf{T}(\mathit{dir}, \phi)$ over V'. Define $V'_x := V_x \cup S_x \cup \{\#_{\overleftarrow{x}}, \#_{\overrightarrow{x}}\}$.

- $\mathsf{T}(\mathit{dir}, \top) = \top$;
- $\mathsf{T}(\leftarrow, p[x]) = p_{\overleftarrow{x}}$ for all $p \in V$;
- $\mathsf{T}(\rightarrow, p[x]) = p_{\overrightarrow{x}}$ for all $p \in V$;
- $\mathsf{T}(\mathit{dir}, \langle x \rangle \psi[x]) = \mathsf{T}_x(\mathit{dir}, \psi[x])$, where $\mathsf{T}_x(\mathit{dir}, \psi[x])$ is obtained from $\mathsf{T}(\mathit{dir}, \psi[x])$ by replacing each occurrence of $\mathsf{Halt}_{\leftarrow}$ (resp., $\mathsf{Halt}_{\rightarrow}$) with $\#_{\overleftarrow{x}}$ (resp., $\#_{\overrightarrow{x}}$);
- $\mathsf{T}(\mathit{dir}, \neg\phi) = \neg\mathsf{T}(\mathit{dir}, \phi)$;
- $\mathsf{T}(\mathit{dir}, \phi_1 \vee \phi_2) = \mathsf{T}(\mathit{dir}, \phi_1) \vee \mathsf{T}(\mathit{dir}, \phi_2)$;
- $\mathsf{T}(\leftarrow, \mathbf{X}\phi) = \mathbf{Y}(\mathsf{T}(\leftarrow, \phi) \wedge \mathbf{Y}\top) \vee \mathbf{Y}(\mathsf{T}(\rightarrow, \phi) \wedge \neg\mathbf{Y}\top)$;
- $\mathsf{T}(\rightarrow, \mathbf{X}\phi) = \mathbf{X}\,\mathsf{T}(\rightarrow, \phi)$;
- $\mathsf{T}(\leftarrow, \mathbf{Y}\phi) = \mathbf{X}(\mathsf{T}(\leftarrow, \ \phi) \wedge \neg\mathsf{Halt}_{\leftarrow})$;
- $\mathsf{T}(\rightarrow, \mathbf{Y}\phi) = \mathbf{Y}(\mathsf{T}(\rightarrow, \ \phi) \wedge \neg\mathsf{Halt}_{\rightarrow}) \vee [\neg\mathbf{Y}\top \wedge \mathbf{X}(\mathsf{T}(\leftarrow, \ \phi) \wedge \neg\mathsf{Halt}_{\leftarrow})]$;
- $\mathsf{T}(\leftarrow, \phi_1 \,\mathbf{U}\, \phi_2) = \mathsf{T}(\leftarrow, \phi_1)\,\mathbf{S}\,(\mathbf{Y}\top \wedge \mathsf{T}(\leftarrow, \phi_2)) \vee [\mathbf{H}(\mathbf{Y}\top \rightarrow \mathsf{T}(\leftarrow, \phi_1)) \wedge \mathbf{O}(\neg\mathbf{Y}\top \wedge \mathsf{T}(\rightarrow, \phi_1)\,\mathbf{U}\,\mathsf{T}(\rightarrow, \phi_2))]$;
- $\mathsf{T}(\rightarrow, \phi_1 \,\mathbf{U}\, \phi_2) = \mathsf{T}(\rightarrow, \phi_1)\,\mathbf{U}\,\mathsf{T}(\rightarrow, \phi_2)$;
- $\mathsf{T}(\leftarrow, \phi_1 \,\mathbf{S}\, \phi_2) = \mathsf{T}(\leftarrow, \phi_1)\,\mathbf{U}\,(\mathsf{T}(\leftarrow, \phi_2) \wedge \neg\mathsf{Halt}_{\leftarrow})$;
- $\mathsf{T}(\rightarrow, \phi_1 \,\mathbf{S}\, \phi_2) = \mathsf{T}(\rightarrow, \phi_1)\,\mathbf{S}\,(\mathsf{T}(\rightarrow, \phi_2) \wedge \neg\mathsf{Halt}_{\rightarrow})) \vee [\mathbf{H}\mathsf{T}(\rightarrow, \phi_1) \wedge \mathbf{O}\big(\neg\mathbf{Y}\top \wedge \mathbf{X}(\mathsf{T}(\leftarrow, \phi_1)\,\mathbf{U}\,(\mathsf{T}(\leftarrow, \phi_2) \wedge \neg\mathsf{Halt}_{\leftarrow}))\big)]$;
- $\mathsf{T}(\leftarrow, \exists x.\, \phi) = \exists\, V'_x.\, (\theta(x, \leftarrow) \wedge \mathsf{T}(\leftarrow, \phi) \wedge \neg\#_{\overleftarrow{x}} \wedge \mathbf{X}\#_{\overleftarrow{x}})$;
- $\mathsf{T}(\rightarrow, \exists x.\, \phi) = \exists\, V'_x.\, (\theta(x, \rightarrow) \wedge \mathsf{T}(\rightarrow, \phi) \wedge \neg\#_{\overrightarrow{x}} \wedge (\mathbf{Y}\top \rightarrow \mathbf{Y}\#_{\overrightarrow{x}}))$;
- $\mathsf{T}(\leftarrow, \exists^{\mathsf{P}} x.\, \phi) = \exists\, V'_x.\, (\theta(x, \leftarrow) \wedge \mathsf{T}(\leftarrow, \phi) \wedge \neg\#_{\overleftarrow{x}})$;

- $\mathsf{T}(\rightarrow, \exists^{\mathsf{P}} x.\, \phi) = \exists\, V'_x.\, (\mathsf{T}(\rightarrow, \phi) \wedge [\theta(x, \leftarrow) \vee (\theta(x, \rightarrow) \wedge \neg \#_{\overrightarrow{x}})])$;

where $\theta(x, \leftarrow)$ and $\theta(x, \rightarrow)$ are the PLTL formulae of Claim 1. By construction $\mathsf{T}(\mathit{dir}, \phi)$ has size linear in ϕ and has the same strong alternation depth as ϕ. Moreover, $\mathsf{T}(\mathit{dir}, \phi)$ is a QPTL sentence if ϕ is a SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ sentence. Now, we prove that the construction is correct. Given a SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ formula ϕ and a $(\mathcal{K}, F)$-assignment Π such that $\mathit{Dom}(\Pi)$ consists of all and only the trace variables occurring free in ϕ, we write $\Pi \models \phi$ to mean that $(\Pi', \mathsf{VAR}) \models_{T(\mathcal{K},F)} \phi$, where Π' is the trace assignment over $T(\mathcal{K}, F)$ obtained from Π in the obvious way. For a trace σ over V', a variable $x \in \mathsf{VAR}$, and a trace σ_x over V'_x, we denote by $\sigma[x \mapsto \sigma_x]$ the trace σ' over V' such that $\sigma' =_{V' \setminus V'_x} \sigma$ and the projection of σ' over V'_x is σ_x. For a SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ sentence φ, we show that $\Pi_{\emptyset} \models \varphi \Leftrightarrow (\emptyset^{\omega}, 0) \models \mathsf{T}(\rightarrow, \varphi)$ where $\mathit{Dom}(\Pi_{\emptyset}) = \emptyset$. The result directly follows from the following claim.

Claim 3: let $\mathit{dir} \in \{\leftarrow, \rightarrow\}$, ϕ be a SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ formula, and Π be a $(\mathcal{K}, F)$-assignment such that $\mathit{Dom}(\Pi)$ contains all the trace variables occurring free in ϕ and $\mathit{dir} = \rightarrow$ if $\mathit{Dom}(\Pi) = \emptyset$. Then, for all encodings (σ, i) of Π such that (σ, i) is a forward encoding if $\mathit{dir} = \rightarrow$, and a backward encoding otherwise, the following holds: $\Pi \models \phi \Leftrightarrow (\sigma, i) \models \mathsf{T}(\mathit{dir}, \phi)$.

The proof of Claim 3 is by structural induction on ϕ. When ϕ is a trace-relativized atomic proposition, the result directly follows from the definition of the map T and the notion of an encoding (σ, i) of Π. The cases for the boolean connectives directly follow from the induction hypothesis, while the cases relative to the temporal modalities easily follow from the induction hypothesis, Claim 2, and the definition of the map T. For the other cases, the ones relative to the hyper quantifiers, we proceed as follows:

- $\phi = \exists x.\, \phi'$: we focus on the case, where $\mathit{dir} = \leftarrow$ (the other case being similar). By hypothesis, (σ, i) is a backward encoding of Π and $\mathit{Dom}(\Pi) \neq \emptyset$. Hence, it holds that $i \geq 1$. For the implication $\Pi \models \phi \Rightarrow (\sigma, i) \models \mathsf{T}(\leftarrow, \phi)$, assume that $\Pi \models \phi$. Hence, there exists a F-fair path π of $\mathcal{K}$ such that $\Pi[x \mapsto (\pi, 0)] \models \phi'$. Since $i \geq 1$, by construction, the trace σ_x over V'_x given by $(\overleftarrow{\sigma}(x, \pi, i), i)$ is the backward x-encoding of $(\pi, 0)$ where i is in backward mode in the encoding. Hence, the trace σ' given by $\sigma[x \mapsto \sigma_x]$ is a backward encoding of $\Pi[x \mapsto (\pi, 0)]$. By the induction hypothesis, $(\sigma', i) \models \mathsf{T}(\leftarrow, \phi')$. Moreover, by construction and Claim 1, $(\sigma_x, i) \models \theta(x, \leftarrow)$, $\#_{\overleftarrow{x}} \notin \sigma_x(i)$ and $\#_{\overleftarrow{x}} \in \sigma_x(i+1)$. Hence, by definition of $\mathsf{T}(\leftarrow, \exists x.\, \phi')$, we obtain that $(\sigma, i) \models \mathsf{T}(\leftarrow, \phi)$.

  For the converse implication, assume that $(\sigma, i) \models \mathsf{T}(\leftarrow, \phi)$. By definition of

$\mathsf{T}(\leftarrow, \exists x.\, \phi')$ and Claim 1, there exists a trace σ_x over V'_x such that σ_x is a backward x-encoding $\overleftarrow{\sigma}(x, \pi, k)$ of some F-fair path π of $\mathcal{K}$ for some offset $k > 0$. Moreover, $(\sigma', i) \models \mathsf{T}(\leftarrow, \phi')$, where σ' is given by $\sigma[x \mapsto \sigma_x]$, $\#_{\overleftarrow{x}} \notin \sigma_x(i)$ and $\#_{\overleftarrow{x}} \notin \sigma_x(i{+}1)$. This means that the offset k is exactly i. Hence, $(\overleftarrow{\sigma}(x, \pi, i), i)$ is a backward x-encoding of $(\pi, 0)$ where i is in backward mode in the encoding, and (σ', i) is a backward encoding of $\Pi[x \mapsto (\pi, 0)]$. Thus, by the induction hypothesis, the result directly follows.

- $\phi = \exists^{\mathsf{P}} x.\, \phi'$: we focus on the case, where $dir = \rightarrow$ (the other case being similar). By hypothesis, (σ, i) is a forward encoding of Π. For the implication $\Pi \models \phi \Rightarrow (\sigma, i) \models \mathsf{T}(\rightarrow, \phi)$, assume that $\Pi \models \phi$. Hence, there exists a F-fair path π of $\mathcal{K}$ and a position $\ell \geq 0$ such that $\Pi[x \mapsto (\pi, \ell)] \models \phi'$. Let (σ_x, i) be the trace over V'_x defined as follows:

  - if $\ell \leq i$: we set (σ_x, i) to $(\overrightarrow{\sigma}(x, \pi, i - \ell), i)$ which by construction is a forward x-encoding of (π, ℓ). By construction and Claim 1, $(\sigma_x, i) \models \theta(x, \rightarrow) \wedge \neg \#_{\overrightarrow{x}}$;
  - if $\ell > i$: we set (σ_x, i) to $(\overleftarrow{\sigma}(x, \pi, \ell - i), i)$ which by construction is a backward x-encoding of (π, ℓ) where i is in forward mode in the encoding. By construction and Claim 1, $(\sigma_x, i) \models \theta(x, \leftarrow)$.

  Hence, the trace σ' given by $\sigma[x \mapsto \sigma_x]$ is a forward encoding of $\Pi[x \mapsto (\pi, \ell)]$. By the induction hypothesis, $(\sigma', i) \models \mathsf{T}(\leftarrow, \phi')$. Hence, by definition of $\mathsf{T}(\rightarrow, \exists^{\mathsf{P}} x.\, \phi')$, we obtain that $(\sigma, i) \models \mathsf{T}(\rightarrow, \phi)$.

  For the converse implication, assume that $(\sigma, i) \models \mathsf{T}(\rightarrow, \phi)$. By definition of $\mathsf{T}(\rightarrow, \exists x.\, \phi')$ and Claim 1, there exists a F-fair path π of $\mathcal{K}$ and a trace σ_x over V'_x such that $(\sigma', i) \models \mathsf{T}(\rightarrow, \phi')$, where σ' is given by $\sigma[x \mapsto \sigma_x]$ and one of the following holds:

  - σ_x is the forward x-encoding $\overrightarrow{\sigma}(x, \pi, k)$ of π for some offset $k \geq 0$ and $\#_{\overrightarrow{x}} \notin \sigma_x(i)$. It follows that $k \leq i$ and (σ_x, i) is the forward x-encoding of the pointed path $(\pi, i - k)$.
  - σ_x is the backward x-encoding $\overleftarrow{\sigma}(x, \pi, k)$ of π for some offset $k > 0$. Hence, (σ_x, i) is the backward x-encoding of the pointed path $(\pi, i + k)$ where i is in forward mode in the encoding.

  Hence, for some $\ell \geq 0$, (σ', i) is a forward encoding of $\Pi[x \mapsto (\pi, \ell)]$. Thus, being $(\sigma', i) \models \mathsf{T}(\rightarrow, \phi')$, by the induction hypothesis, the result directly follows.

This concludes the proof of Claim 3 and Theorem 8.3 too. □

By an adaptation of the known reduction of satisfiability of QPTL without past to model checking of HyperCTL* [121], we obtain the following result.

Theorem 8.4 *Given a QPTL sentence ψ over V, one can build in linear time an STS M_V (depending only on V) and a singleton-free SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ sentence φ having the same strong alternation depth as ψ such that ψ is satisfiable* iff $\mathcal{L}(M_V) \models \varphi$.

Proof: Without loss of generality, we only consider *well-named* QPTL formulae, i.e. QPTL formulae where each quantifier introduces a different proposition. Moreover, we can assume that V is the set of all and only the propositions occurring in the given QPTL sentence. Let $V' = V \cup \{tag, in\}$, where *tag* and *in* are fresh propositions, and fix an ordering $\{p_1, \ldots, p_n\}$ of the propositions in V. First, we encode a trace σ over V by a trace $en(\sigma)$ over V' defined as follows: $en(\sigma) := \sigma_0 \cdot \sigma_1 \cdot \ldots$, where for each $i \geq 0$, σ_i (the encoding of the i^{th} symbol of σ) is the finite word over $2^{V'}$ of length $n+1$ given by $P_0 P_1, \ldots P_n$, where (i) for all $k \in [1, n]$, $P_k = \{p_k\}$ if $p_k \in \sigma(i)$, and $P_k = \emptyset$ otherwise, and (ii) $P_0 = \{tag\}$ if $i > 0$ and $P_0 = \{tag, in\}$ otherwise. Note that proposition *in* marks the first position of $en(\sigma)$. Then, the finite Kripke structure $\mathcal{K}_V = \langle S, S_0, E, Lab \rangle$ over V' has size linear in $|V|$ and it is constructed in such a way that its set of traces $T(\mathcal{K}_V)$ is the set of the encodings of the traces over V. Formally, $\mathcal{K}_V$ is defined as follows:

- $S = \{p_h, \overline{p}_h \mid h \in \{1, \ldots, n\}\} \cup \{tag, in\}$ and $S_0 = \{in\}$;
- E consists of the edges (p_k, p_{k+1}), $(p_k, \overline{p}_{k+1})$, $(\overline{p}_k, p_{k+1})$ and $(\overline{p}_k, \overline{p}_{k+1})$ for all $k \in [1, n-1]$, and the edges (s, p_1), $(s, \overline{p}_1)$, (p_n, tag), and $(\overline{p}_n, tag)$ where $s \in \{tag, in\}$.
- $Lab(tag) = \{tag\}$, $Lab(in) = \{in, tag\}$, and $Lab(p_k) = \{p_k\}$ and $Lab(\overline{p}_k) = \emptyset$ for all $k \in [1, n]$.

Let Λ be the set of pairs (h, ψ) consisting of a natural number $h \in [0, n]$ and a well-named QPTL formula ψ over V such that there is no quantifier in ψ binding proposition p_h if $h \neq 0$, and ψ is a sentence iff $h = 0$. We inductively define a mapping assigning to each pair $(h, \psi) \in \Lambda$ a singleton-free SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ formula $\mathsf{T}(h, \psi)$ over V' and $\mathsf{VAR} = \{x_1, \ldots, x_n\}$ (intuitively, if $h \neq 0$, then p_h represents the currently quantified proposition):

- $\mathsf{T}(h, \top) = \top$;

- $\mathsf{T}(h, p_i) = \mathbf{X}^i\, p_i[x_h]$ for all $p_i \in V$;
- $\mathsf{T}(h, \neg\psi) = \neg\mathsf{T}(h, \psi)$;
- $\mathsf{T}(h, \psi_1 \vee \psi_2) = \mathsf{T}(h, \psi_1) \vee \mathsf{T}(h, \psi_2)$;
- $\mathsf{T}(h, \mathbf{X}\psi) = \mathbf{X}^{n+1}\mathsf{T}(h, \psi)$;
- $\mathsf{T}(h, \mathbf{Y}\psi) = \mathbf{Y}^{n+1}\mathsf{T}(h, \psi)$;
- $\mathsf{T}(h, \psi_1\, \mathbf{U}\, \psi_2) = (tag[x_h] \rightarrow \mathsf{T}(h, \psi_1))\, \mathbf{U}\, (\mathsf{T}(h, \psi_2) \wedge tag[x_h])$;
- $\mathsf{T}(h, \psi_1\, \mathbf{S}\, \psi_2) = (tag[x_h] \rightarrow \mathsf{T}(h, \psi_1))\, \mathbf{S}\, (\mathsf{T}(h, \psi_2) \wedge tag[x_h])$;
- $$\mathsf{T}(h, \exists p_k.\psi) = \begin{cases} \exists^{\mathsf{P}}\, x_k.\, \Big(\mathsf{T}(k, \psi) \wedge \mathbf{O}\big(in[x_h] \wedge in[x_k] \wedge \\ \qquad\qquad \mathbf{G} \bigwedge_{j \in [1,n]\setminus\{k\}} (p_j[x_h] \leftrightarrow p_j[x_k])\big)\Big) & \text{if } h \neq 0 \\ \exists\, x_k.\, \mathsf{T}(k, \psi) & \text{otherwise} \end{cases}$$

By construction, $\mathsf{T}(h, \psi)$ has size linear in ψ and has the same strong alternation depth as ψ. Moreover, $\mathsf{T}(h, \psi)$ is a SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ sentence *iff* ψ is a QPTL sentence. Now we show that the construction is correct. A $T(\mathcal{K}_V)$-*assignment* is a mapping $\Pi : \mathsf{VAR} \to T(\mathcal{K}_V)$. For all $i \geq 0$, $T(\mathcal{K}_V)$-assignments Π, and SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ formulae φ over V', we write $(\Pi, i) \models \varphi$ to mean that $(\Pi_i, \mathsf{VAR}) \models_{T(\mathcal{K}_V)} \varphi$, where Π_i is the (pointed) trace assignment over $T(\mathcal{K}_V)$ assigning to each trace variable $x \in \mathsf{VAR}$, the pointed trace $(\Pi(x), i)$. Note that for each QPTL sentence ψ over V, ψ is satisfiable if for all traces σ, $(\sigma, 0) \models \psi$. Thus, correctness of the construction directly follows from the following claim, where for each $i \geq 0$, $\wp(i) := i \cdot (n+1)$. Intuitively, $\wp(i)$ is the *tag*-position associated with the V'-encoding of the position i of a trace over V.

Claim. Let $(h, \psi) \in \Lambda$. Then, for all pointed traces (σ, i) over V and $T(\mathcal{K}_V)$-assignments Π such that $\Pi(x_h) = en(\sigma)$ if $h \neq 0$, and $i = 0$ if $h = 0$, the following holds:

$$(\sigma, i) \models \psi \Leftrightarrow (\Pi, \wp(i)) \models \mathsf{T}(h, \psi)$$

The claim is proved by structural induction on ψ. The cases for the boolean connectives easily follow from the induction hypothesis. For the other cases, we proceed as follows:

- $\psi = p_j$ for some $p_j \in V$: in this case $h \neq 0$ (recall that $(h, \psi) \in \Lambda$). Then, we have that $(\sigma, i) \models p_j \Leftrightarrow p_j \in \sigma(i) \Leftrightarrow p_j \in en(\sigma)(\wp(i) + j) \Leftrightarrow p_j \in \Pi(x_h)(\wp(i) + j)$ $\Leftrightarrow (\Pi, \wp(i)) \models \mathbf{X}^j\, p_j[x_h] \Leftrightarrow (\Pi, \wp(i)) \models \mathsf{T}(h, p_j)$. Hence, the result follows.

- $\psi = \mathbf{X}\psi'$: hence, $h \neq 0$. We have that $(\sigma, i) \models \mathbf{X}\psi' \Leftrightarrow (\sigma, i+1) \models \psi' \Leftrightarrow$ (by the induction hypothesis) $(\Pi, \wp(i+1)) \models \mathsf{T}(h, \psi') \Leftrightarrow$ (since $\wp(i+1) = \wp(i) + n + 1$) $(\Pi, \wp(i)) \models \mathbf{X}^{n+1}\mathsf{T}(h, \psi') \Leftrightarrow (\Pi, \wp(i)) \models \mathsf{T}(h, \mathbf{X}\psi')$. Hence, the result follows.

- $\psi = \mathbf{Y}\psi'$: similar to the previous case.

- $\psi = \psi_1 \,\mathbf{U}\, \psi_2$: hence, $h \neq 0$. We have that $(\sigma, i) \models \psi_1 \,\mathbf{U}\, \psi_2 \Leftrightarrow$ there is $j \geq i$ such that $(\sigma, j) \models \psi_2$ and $(\sigma, \ell) \models \psi_1$ for all $i \leq \ell < j \Leftrightarrow$ (by the induction hypothesis) there is $j \geq i$ such that $(\Pi, \wp(j)) \models \mathsf{T}(h, \psi_2)$ and $(\Pi, \wp(\ell)) \models \mathsf{T}(h, \psi_1)$ for all $i \leq \ell < j \Leftrightarrow$ there is $j' \geq \wp(i)$ such that $(\Pi, j') \models \mathsf{T}(h, \psi_2)$ and $tag \in \Pi(x_h)(j')$, and for all $\wp(i) \leq \ell' < j'$ such that $tag \in \Pi(x_h)(\ell')$, $(\Pi, \ell') \models \mathsf{T}(h, \psi_1)$ $\Leftrightarrow (\Pi, \wp(i)) \models \mathsf{T}(h, \psi_1 \,\mathbf{U}\, \psi_2)$. Hence, the result follows.

- $\psi = \psi_1 \,\mathbf{S}\, \psi_2$: similar to the previous case.

- $\psi = \exists p_k.\, \psi'$ and $h \neq 0$: by hypothesis, $k \neq h$. For the implication, $(\Pi, \wp(i)) \models \mathsf{T}(h, \psi) \Rightarrow (\sigma, i) \models \psi$, assume that $(\Pi, \wp(i)) \models \mathsf{T}(h, \psi)$. By definition of $\mathsf{T}(h, \exists p_k.\, \psi')$ and since *in* marks the first position of the encoding of a trace over V, it easily follows that there exists a pointed trace over V of the form (σ', i) such that $\sigma' =_{V \setminus \{p_k\}} \sigma$ and $(\Pi[x_k \leftarrow en(\sigma')], \wp(i)) \models \mathsf{T}(k, \psi')$. Since $(k, \psi') \in \Lambda$, by the induction hypothesis, it follows that $(\sigma', i) \models \psi'$. Thus, being $\sigma' =_{V \setminus \{p_k\}} \sigma$, we obtain that $(\sigma, i) \models \psi$.

  The converse implication $(\sigma, i) \models \psi \Rightarrow (\Pi, \wp(i)) \models \mathsf{T}(h, \psi)$ is similar, and we omit the details here.

- $\psi = \exists p_k.\, \psi'$ and $h = 0$: hence, ψ and $\mathsf{T}(0, \psi)$ are sentences. We have that $(\sigma, 0) \models \exists p_k.\, \psi' \Leftrightarrow (\psi$ is a QPTL sentence) for some trace σ' over V, $(\sigma', 0) \models \psi' \Leftrightarrow$ (by the induction hypothesis) $(\Pi[x_k \to en(\sigma')], \wp(0)) \models \mathsf{T}(k, \psi') \Leftrightarrow$ (by definition of $\mathsf{T}(0, \exists p_k.\, \psi')$) $(\Pi, \wp(0)) \models \mathsf{T}(0, \exists p_k.\, \psi')$.

This concludes the proof of Theorem 8.4. □

8.2.2 Reduction to fair model checking against $\text{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$.

We solve the (fair) model checking problem for simple $\text{GHyperLTL}_{\mathsf{S+C}}$ by a reduction to fair model checking against the fragment $\text{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$. Our reduction is exponential in the size of the given sentence and is an adaptation of the reduction from model checking simple $\text{HyperLTL}_{\mathsf{S}}$ to model checking HyperLTL shown in [72]. As a preliminary

step, we first show, by an easy adaptation of the standard automata-theoretic approach for PLTL [287], that the problem for a simple GHyperLTL$_{\mathsf{S+C}}$ sentence φ can be reduced in exponential time to the fair model checking problem against a singleton-free sentence in the fragment SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ for some set Γ of *atomic propositions* depending on φ.

Proposition 8.2 *Given a simple GHyperLTL$_{\mathsf{S+C}}$ sentence φ and a fair STS M over V, one can build in single exponential time in the size of φ, an STS M' over an extension V' of V and a singleton-free SHyperLTL$^{\Gamma'}_{\mathsf{S+C}}$ sentence φ' for some $\Gamma' \subseteq V'$ such that $M' \models \varphi'$ if and only if $\mathcal{L}(M) \models \varphi$. Moreover, φ' has the same strong alternation depth as φ, $|\varphi'| = O(|\varphi|)$, and $|\mathcal{K}'| = O(|\mathcal{K}| * 2^{O(|\varphi|)})$.*

In order to prove Proposition 8.2, we need some preliminary results. Recall that a Nondeterministic Büchi Automaton over words (NBA for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, Acc \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $Acc \subseteq Q$ is the set of *accepting* states. Given a infinite word w over Σ, a run of $\mathcal{A}$ over w is an infinite sequence of states $q_0, q_1, \ldots$ such that $q_0 \in Q_0$ and for all $i \geq 0$, $(q_i, w(i), q_{i+1}) \in \Delta$. The run is accepting if for infinitely many i, $q_i \in Acc$. The language $T(\mathcal{A})$ accepted by $\mathcal{A}$ consists of the infinite words w over Σ such that there is an accepting run over w.

Fix a non-empty set Γ of PLTL formulae over V. The closure $cl(\Gamma)$ of Γ is the set of PLTL formulae consisting of the formulae $\top$, $\mathbf{Y}\top$, the sub-formulae of the formulae $\theta \in \Gamma$, and the negations of such formulae (we identify $\neg\neg\theta$ with θ). Note that $\Gamma \subseteq cl(\Gamma)$. Without loss of generality, we can assume that $V \subseteq \Gamma$. Precisely, V can be taken as the set of propositions occurring in the given simple GHyperLTL$_{\mathsf{S+C}}$ sentence and $cl(\Gamma)$ contains all the propositions in V and their negations. For each formula $\theta \in cl(\Gamma) \setminus V$, we introduce a fresh atomic proposition not in V, denoted by $at(\theta)$. Moreover, for allowing a uniform notation, for each $p \in V$, we write $at(p)$ to mean p itself. Let V_Γ be the set V extended with these new propositions. By a straightforward adaptation of the well-known translation of PLTL formulae into equivalent NBA [287], we obtain the following result, where for a trace σ_Γ over V_Γ, $(\sigma_\Gamma)_V$ denotes the projection of σ_Γ over V.

Proposition 8.3 *Given a finite set Γ of PLTL formulae over V, one can construct in single exponential time an NBA $\mathcal{A}_\Gamma$ over 2^{V_Γ} with $2^{O(|V_\Gamma|)}$ states satisfying the following:*

1. *let $\sigma_\Gamma \in \mathcal{L}(\mathcal{A}_\Gamma)$: then for all $i \geq 0$ and $\theta \in cl(\Gamma)$, $at(\theta) \in \sigma_\Gamma(i)$ iff $((\sigma_\Gamma)_V, i) \models \theta$.*
2. *for each trace σ over V, there exists $\sigma_\Gamma \in \mathcal{L}(\mathcal{A}_\Gamma)$ such that $\sigma = (\sigma_\Gamma)_V$.*

Proof: Here, we construct a *generalized* NBA $\mathcal{A}_\Gamma$ satisfying Properties (1) and (2) of Proposition 8.3, which can be converted in linear time into an equivalent NBA. Recall that a generalized NBA is defined as an NBA but the acceptance condition is given by a family $\mathcal{F} = \{Acc_1, \ldots, Acc_k\}$ of sets of accepting states. In this case, a run is accepting if for each accepting component $Acc_i \in \mathcal{F}$, the run visits infinitely often states in Acc_i.

The generalized NBA $\mathcal{A}_\Gamma = \langle 2^{V_\Gamma}, Q, Q_0, \Delta, \mathcal{F}\rangle$ is defined as follows. Q is the set of *atoms* of Γ consisting of the maximal propositionally consistent subsets A of $cl(\Gamma)$. Formally, an atom A of Γ is a subset of $cl(\Gamma)$ satisfying the following:

- $\top \in A$
- for each $\theta \in cl(\Gamma)$, $\theta \in A$ iff $\neg\theta \notin A$;
- for each $\theta_1 \vee \theta_2 \in cl(\Gamma)$, $\theta_1 \vee \theta_2 \in A$ iff $\{\theta_1, \theta_2\} \cap A \neq \emptyset$.

The set Q_0 of initial states consists of the atoms A of Γ such that $\neg\mathbf{Y}\top \in A$. For an atom A, $at(A)$ denotes the subset of propositions in V_Γ associated with the formulae in A, i.e. $at(A) \doteq \{at(\theta) \mid \theta \in A\}$. The transition relation Δ captures the semantics of the next and previous modalities and the local fixpoint characterization of the until and since modalities. Formally, Δ consists of the transitions of the form $(A, at(A), A')$ such that:

- for each $\mathbf{X}\theta \in cl(\Gamma)$, $\mathbf{X}\theta \in A$ iff $\theta \in A'$;
- for each $\mathbf{Y}\theta \in cl(\Gamma)$, $\mathbf{Y}\theta \in A'$ iff $\theta \in A$;
- for each $\theta_1 \,\mathbf{U}\, \theta_2 \in cl(\Gamma)$, $\theta_1 \,\mathbf{U}\, \theta_2 \in A$ iff either $\theta_2 \in A$, or $\theta_1 \in A$ and $\theta_1 \,\mathbf{U}\, \theta_2 \in A'$;
- for each $\theta_1 \,\mathbf{S}\, \theta_2 \in cl(\Gamma)$, $\theta_1 \,\mathbf{S}\, \theta_2 \in A'$ iff either $\theta_2 \in A'$, or $\theta_1 \in A'$ and $\theta_1 \,\mathbf{S}\, \theta_2 \in A$.

Finally, the generalized Büchi acceptance condition is used for ensuring the fulfillment of the liveness requirements θ_2 in the until sub-formulae $\theta_1 \,\mathbf{U}\, \theta_2$ in Γ. Formally, for each $\theta_1 \,\mathbf{U}\, \theta_2 \in cl(\Gamma)$, $\mathcal{F}$ has a component consisting of the atoms A such that either $\neg(\theta_1 \,\mathbf{U}\, \theta_2) \in A$ or $\theta_2 \in A$.

Let $\sigma_\Gamma \in \mathcal{L}(\mathcal{A}_\Gamma)$. By construction, there is an accepting infinite sequence of atoms $\rho = A_0 A_1 \ldots$ such that for all $i \geq 0$, $\sigma_\Gamma(i) = at(A_i)$. Let σ be the projection of σ_Γ over V (note that $A_i \cap V = \sigma(i)$ for all $i \geq 0$). By standard arguments (see [287]), the following holds: for all $i \geq 0$ and $\theta \in cl(\Gamma)$, $\theta \in A_i$ (hence, $at(\theta) \in \sigma_\Gamma(i)$) if and only if $(\sigma, i) \models \theta$. Hence, Property (1) of Proposition 8.3 follows.

For Property (2), let σ be a trace over V and let $\rho = A_0A_1\ldots$ be the infinite sequence of atoms defined as follows for all $i \geq 0$: $A_i = \{\theta \in cl(\Gamma) \mid (\sigma, i) \models \theta\}$. By construction and the semantics of PLTL, ρ is an accepting run of $\mathcal{A}_\Gamma$ over the word $\sigma_\Gamma = at(A_0)at(A_1)\ldots$. Moreover, σ coincides with the projection of σ_Γ over V. Hence, the result follows. □

Let $\mathcal{K} = \langle S, S_0, E, Lab\rangle$ be a finite Kripke structure over V and $F \subseteq S$. Next, we consider the synchronous product of the fair Kripke structure $(\mathcal{K}, F)$ with the NBA $\mathcal{A}_\Gamma = \langle 2^{V_\Gamma}, Q, Q_0, \Delta, Acc\rangle$ over 2^{V_Γ} of Proposition 8.3 associated with Γ. More specifically, we construct a Kripke structure $\mathcal{K}_\Gamma$ over V_Γ and a subset F_Γ of $\mathcal{K}_\Gamma$-states such that $T(\mathcal{K}_\Gamma, F_\Gamma)$ is the set of traces $\sigma_\Gamma \in T(\mathcal{A}_\Gamma)$ whose projections over V are in $T(\mathcal{K}, F)$. Formally, the Γ-*extension of* $(\mathcal{K}, F)$ is the fair Kripke structure $(\mathcal{K}_\Gamma, F_\Gamma)$ where $\mathcal{K}_\Gamma = \langle S_\Gamma, S_{0,\Gamma}, E_\Gamma, Lab_\Gamma\rangle$ and F_Γ are defined as follows:

- S_Γ is the set of tuples $(s, B, q, \ell) \in S \times 2^{V_\Gamma} \times Q \times \{1, 2\}$ such that $Lab(s) = B \cap V$;
- $S_{0,\Gamma} = S_\Gamma \cap (S_0 \times 2^{V_\Gamma} \times Q_0 \times \{1\})$;
- E_Γ consists of the following transitions:
  - $((s, B, q, 1), (s', B', q', \ell))$ such that $(s, s') \in E$, $(q, B, q') \in \Delta$, and $\ell = 2$ if $s \in F$ and $\ell = 1$ otherwise;
  - $((s, B, q, 2), (s', B', q', \ell))$ such that $(s, s') \in E$, $(q, B, q') \in \Delta$, and $\ell = 1$ if $q \in Acc$ and $\ell = 2$ otherwise.
- for each $(s, B, q, \ell) \in S_\Gamma$, $Lab_\Gamma((s, B, q, \ell)) = B$;
- $F_\Gamma = \{(s, B, q, 2) \in S_\Gamma \mid q \in Acc\}$.

By construction and Proposition 8.3(2), we easily obtain the following result.

Proposition 8.4 *For each trace σ_Γ over V_Γ, $\sigma_\Gamma \in T(\mathcal{K}_\Gamma, F_\Gamma)$ if and only if $\sigma_\Gamma \in T(\mathcal{A}_\Gamma)$ and $(\sigma_\Gamma)_V \in T(\mathcal{K}, F)$. Moreover, for each $\sigma \in T(\mathcal{K}, F)$, there exists $\sigma_\Gamma \in T(\mathcal{K}_\Gamma, F_\Gamma)$ such that $(\sigma_\Gamma)_V = \sigma$.*

We can now provide a proof of Proposition 8.2.

Proposition 8.2 *Given a simple GHyperLTL$_{\mathsf{S+C}}$ sentence φ and a fair STS M over V, one can build in single exponential time in the size of φ, an STS M' over an extension V' of V and a singleton-free SHyperLTL$^{\Gamma'}_{\mathsf{S+C}}$ sentence φ' for some $\Gamma' \subseteq V'$ such that $M' \models \varphi'$ if and only if $\mathcal{L}(M) \models \varphi$. Moreover, φ' has the same strong alternation depth as φ, $|\varphi'| = O(|\varphi|)$, and $|\mathcal{K}'| = O(|\mathcal{K}| * 2^{O(|\varphi|)})$.*

Proof: Let φ be a GHyperLTL$_{\mathsf{S+C}}$ sentence over V and $(\mathcal{K}, F)$ be a fair finite Kripke structure $(\mathcal{K}, F)$ over V. Then, there is a finite set Γ_0 of PLTL formulae such that φ is in the fragment SHyperLTL$^{\Gamma_0}_{\mathsf{S+C}}$. Let Γ be the set of PLTL formulae consisting of the formulae in Γ_0 and the PLTL formulae ψ such that $\langle x \rangle \psi[x]$ is a sub-formula of φ for some variable x. Define V', $\Gamma' \subseteq V'$, $(\mathcal{K}', F')$, and φ' as follows:

- $V' \doteq V_\Gamma$ (recall that $V_\Gamma = V \cup \{at(\theta) \mid \theta \in \Gamma\}$) and Γ' is the V_Γ-counterpart of Γ_0, i.e. $\Gamma' \doteq \{at(\theta) \mid \theta \in \Gamma_0\}$;
- $(\mathcal{K}', F') \doteq (\mathcal{K}_\Gamma, F_\Gamma)$, where $(\mathcal{K}_\Gamma, F_\Gamma)$ is the Γ-extension of $(\mathcal{K}, F)$;
- $\varphi' \doteq \mathsf{T}(\varphi)$ where the mapping T replaces (i) each sub-formula $\langle x \rangle \psi[x]$ of φ with its propositional x-version $at(\psi)[x]$, and (ii) each Γ_0-relativized temporal modality with its Γ'-relativized version. Note that $\mathsf{T}(\varphi)$ is a singleton-free SHyperLTL$^{\Gamma'}_{\mathsf{S+C}}$ formula where Γ' is propositional (in particular, $\Gamma' \subseteq V'$) and has the same strong alternation depth as φ.

It remains to show that the construction is correct, i.e. $T(\mathcal{K}_\Gamma, F_\Gamma) \models \mathsf{T}(\varphi)$ *iff* $T(\mathcal{K}, F) \models \varphi$. Let Λ be the set of formulae ϕ in the fragment SHyperLTL$^{\Gamma_0}_{\mathsf{S+C}}$ such that for each sub-formula $\langle x \rangle \psi[x]$ of ϕ, it holds that $\psi \in \Gamma$. Moreover, for a trace assignment Π_Γ over $T(\mathcal{K}_\Gamma, F_\Gamma)$, the *$V$-projection of* Π_Γ, written $(\Pi_\Gamma)_V$, is the trace assignment with domain $Dom(\Pi)$ obtained from Π by replacing each pointed trace $\Pi(x)$, where $x \in Dom(\Pi)$ and $\Pi(x)$ is of the form (σ_Γ, i), with $((\sigma_\Gamma)_V, i)$. Note that by Proposition 8.4, $(\Pi_\Gamma)_V$ is a trace assignment over $T(\mathcal{K}, F)$. Correctness of the construction directly follows from the following claim.

Claim. Let $\phi \in \Lambda$ and Π_Γ be a trace assignment over $T(\mathcal{K}_\Gamma, F_\Gamma)$. Then:

$$(\Pi_\Gamma, \mathsf{VAR}) \models_{T(\mathcal{K}_\Gamma, F_\Gamma)} \mathsf{T}(\phi) \text{ iff } ((\Pi_\Gamma)_V, \mathsf{VAR}) \models_{T(\mathcal{K}, F)} \phi$$

The claim is proved by structural induction on $\phi \in \Lambda$. The cases where the root modality of ϕ is a Boolean connective directly follow from the induction hypothesis. For the other cases, we proceed as follows.

- $\phi = \langle x \rangle \psi[x]$, where $\psi \in \Gamma$. Hence, $\mathsf{T}(\phi) = at(\psi)[x]$. Let $\Pi_\Gamma(x) = (\sigma_\Gamma, i)$. We have that $(\Pi_\Gamma)_V(x) = ((\sigma_\Gamma)_V, i)$. By Propositions 8.3 and 8.4, $at(\psi) \in \sigma_\Gamma(i)$ *iff* $((\sigma_\Gamma)_V, i) \models \psi$. Hence, the result follows.
- The root modality of ϕ is a Γ_0-relativized temporal modality. Assume that $\phi = \phi_1 \, \mathbf{U}_{\Gamma_0} \, \phi_2$ (the other cases being similar). Then, $\mathsf{T}(\phi) = \mathsf{T}(\phi_1) \, \mathbf{U}_{\Gamma'} \, \mathsf{T}(\phi_2)$ (recall

that $\Gamma' = \{at(\theta) \mid \theta \in \Gamma_0\}$ and $\Gamma_0 \subseteq \Gamma$). By Propositions 8.3 and 8.4, for each pointed trace (σ_Γ, i) over $T(\mathcal{K}_\Gamma, F_\Gamma)$ and $\theta \in \Gamma_0$, it holds that $at(\theta) \in \sigma_\Gamma(i)$ *iff* $((\sigma_\Gamma)_V, i) \models \theta$. By construction, it follows that $succ_{(\Gamma_0,\mathsf{VAR})}((\Pi_\Gamma)_V)$ is the V-projection of $succ_{(\Gamma',\mathsf{VAR})}(\Pi_\Gamma)$. Hence, by the semantics of GHyperLTL$_{\mathsf{S+C}}$ and the induction hypothesis, the result follows.

- $\phi = \exists x.\, \phi'$. Hence, $\mathsf{T}(\phi) = \exists x.\, \mathsf{T}(\phi')$. By Proposition 8.4, for each trace $\sigma \in T(\mathcal{K}, F)$, there exists $\sigma_\Gamma \in T(\mathcal{K}_\Gamma, F_\Gamma)$ such that $(\sigma_\Gamma)_V = \sigma$. Hence, the result easily follows from the induction hypothesis.

- $\phi = \exists^{\mathsf{P}} x.\, \phi'$: this case is similar to the previous one.

□

Let us fix a non-empty finite set $\Gamma \subseteq V$ of atomic propositions. We now show that fair model checking of the singleton-free fragment of SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ can be reduced in polynomial time to fair model checking of SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$. We observe that in the singleton-free fragment of SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$, when a pointed trace (σ, i) is selected by a pointed quantifier $\exists^{\mathsf{P}} x$, the positions of σ which are visited during the evaluation of the temporal modalities are all in the (Γ, ω)-stutter factorization of σ with the possible exception of the position i chosen by $\exists^{\mathsf{P}} x$. Thus, given a set T of traces and a special proposition $\# \notin V$, we define an extension $stfr^{\#}_{\Gamma}(T)$ of the set $stfr_{\Gamma}(T) = \{stfr_{\Gamma}(\sigma) \mid \sigma \in T\}$ as follows. Intuitively, we consider for each trace $\sigma \in T$, its Γ-stutter trace $stfr_{\Gamma}(\sigma)$ and the extensions of $stfr_{\Gamma}(\sigma)$ which are obtained by adding an extra position marked by proposition $\#$ (this extra position does not belong to the (Γ, ω)-stutter factorization of σ). Formally, $stfr^{\#}_{\Gamma}(T)$ is the smallest set containing $stfr_{\Gamma}(T)$ and satisfying the following condition:

- for each trace $\sigma \in T$ with (Γ, ω)-stutter factorization $\{\ell_k\}_{k\geq 0}$ and position $i \in (\ell_k, \ell_{k+1})$ for some $k \geq 0$, the trace $\sigma(\ell_0) \ldots \sigma(\ell_k)\, (\sigma(i) \cup \{\#\})\, \sigma(\ell_{k+1})\, \sigma(\ell_{k+2}) \ldots \in stfr^{\#}_{\Gamma}(T)$.

Given a singleton-free formula φ in SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$, we denote by $\mathsf{T}_{\#}(\varphi)$ the formula in SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ obtained from φ by applying inductively the following transformations:

- the Γ-relativized temporal modalities are replaced with their $\emptyset$-relativized counterparts;

- each formula $\exists^{\mathsf{P}} x.\, \phi$ is replaced with $\exists^{\mathsf{P}} x.\, \big(\mathsf{T}_{\#}(\phi) \wedge \langle x \rangle (\mathbf{XG}\neg\#[x] \wedge (\mathbf{Y}\top \rightarrow \mathbf{YH}\neg\#[x]))\big)$.

Intuitively, the formula $\mathsf{T}_{\#}(\exists^{\mathsf{P}} x.\, \phi)$ states that for the pointed trace (σ, i) selected by the pointed quantifier, at most position i may be marked by the special proposition $\#$. By the semantics of the logics considered, the following holds.

Remark 8.1 *Given a singleton-free sentence φ of SHyperLTL$^{\Gamma}_{\mathsf{S}+\mathsf{C}}$ and a set T of traces, it holds that $T \models \varphi$ if and only if $stfr^{\#}_{\Gamma}(T) \models \mathsf{T}_{\#}(\varphi)$.*

Let us fix now a fair finite Kripke structure $(\mathcal{K}, F)$. We first show that one can build in polynomial time a finite Kripke structure $(\mathcal{K}_\Gamma, F_\Gamma)$ and a LTL formula θ_Γ such that $stfr^{\#}_{\Gamma}(T(\mathcal{K}, F))$ coincides with the traces of $T(\mathcal{K}_\Gamma, F_\Gamma)$ satisfying θ_Γ.

Proposition 8.5 *Given $\emptyset \neq \Gamma \subseteq V$ and a fair finite Kripke structure $(\mathcal{K}, F)$ over V, one can construct in polynomial time a fair finite Kripke structure $(\mathcal{K}_\Gamma, F_\Gamma)$ and a LTL formula θ_Γ such that $stfr^{\#}_{\Gamma}(T(\mathcal{K}, F))$ is the set of traces $\sigma \in T(\mathcal{K}_\Gamma, F_\Gamma)$ so that $\sigma \models \theta_\Gamma$.*

Proof: Let $\mathcal{K} = \langle S, S_0, E, Lab \rangle$. Intuitively, the Kripke structure $\mathcal{K}_\Gamma$ is obtained from $\mathcal{K}$ by adding edges which are *summaries* of finite paths π of $\mathcal{K}$ where *either* the propositional valuation in Γ changes only at the final state of π, or the propositional valuation in Γ does not change along π. Formally, let $R_\Gamma(\mathcal{K})$ and $R_\Gamma(\mathcal{K}, F)$ be the sets of state pairs in $\mathcal{K}$ defined as follows:

- $R_\Gamma(\mathcal{K})$ consists of the pairs $(s, s') \in S \times S$ such that $Lab(s) \cap \Gamma \neq Lab(s') \cap \Gamma$ and there is a finite path of $\mathcal{K}$ of the form $s \cdot \rho \cdot s'$ such that $Lab(s) \cap \Gamma = Lab(\rho(i)) \cap \Gamma$ for all $0 \leq i < |\rho|$.
- $R_\Gamma(\mathcal{K}, F)$ is defined similarly but, additionally, we require that the finite path $s \cdot \rho \cdot s'$ visits some state in F.

Intuitively, $R_\Gamma(\mathcal{K})$ keeps track of the initial and final states of the finite paths of $\mathcal{K}$ with length at least 2 where the propositional valuation in Γ changes only at the final state of the finite path. Additionally, $R_\Gamma(\mathcal{K}, F)$ considers only those finite paths which visit some state in F. Moreover, let $R^{\#}_\Gamma(\mathcal{K})$ and $R^{\#}_\Gamma(\mathcal{K}, F)$ be the sets of state pairs in $\mathcal{K}$ defined as follows:

- $R^{\#}_\Gamma(\mathcal{K})$ consists of the pairs $(s, s') \in S \times S$ such that $Lab(s) \cap \Gamma = Lab(s') \cap \Gamma$ and there is a finite path of $\mathcal{K}$ of the form $s \cdot \rho \cdot s'$ such that $Lab(s) \cap \Gamma = Lab(\rho(i)) \cap \Gamma$ for all $0 \leq i < |\rho|$.

- $E^{\#}_{\Gamma}(\mathcal{K}, F)$ is defined similarly but, additionally, we require that the finite path $s \cdot \rho \cdot s'$ visits some accepting state in F.

Thus, $R^{\#}_{\Gamma}(\mathcal{K})$ keeps track of the initial and final states of the finite paths of $\mathcal{K}$ with length at least 2 where the propositional valuation in Γ does not change. Additionally, $R^{\#}_{\Gamma}(\mathcal{K}, F)$ considers only those finite paths which visit some state in F. The finite sets $R_{\Gamma}(\mathcal{K})$, $R_{\Gamma}(\mathcal{K}, F)$, $R^{\#}_{\Gamma}(\mathcal{K})$, $R^{\#}_{\Gamma}(\mathcal{K}, F)$ can be easily computed in polynomial time by standard closure algorithms. By exploiting these finite sets, we define the finite Kripke structure $\mathcal{K}_{\Gamma} = \langle S_{\Gamma}, S_{\Gamma,0}, E_{\Gamma}, Lab_{\Gamma} \rangle$ and the set $F_{\Gamma} \subseteq S_{\Gamma}$ as follows.

- S_{Γ} is given by $S \times 2^{\{acc,\#\}}$ and $S_{\Gamma,0}$ is the set of states of the form $(s, \emptyset)$ for some $s \in S_0$.
- E_{Γ} consists of the edges $((s,T),(s',T'))$ such that one of the following holds:
  - $(s,s') \in E \cup R_{\Gamma}(\mathcal{K})$, $\# \notin T'$, and $(acc \in T'$ iff $s' \in F)$;
  - $(s,s') \in R_{\Gamma}(\mathcal{K}, F)$, $\# \notin T'$, and $acc \in T'$;
  - $(s,s') \in R^{\#}_{\Gamma}(\mathcal{K})$, $\# \notin T$, $\# \in T'$, and $(acc \in T'$ iff $s' \in F)$;
  - $(s,s') \in R^{\#}_{\Gamma}(\mathcal{K}, F)$, $\# \notin T$, $\# \in T'$, and $acc \in T'$.
- $Lab_{\Gamma}(s,T) = Lab(s) \cup \{\#\}$ if $\# \in T$, and $Lab_{\Gamma}(s,T) = Lab(s)$ otherwise;
- F_{Γ} is the set of $\mathcal{K}_{\Gamma}$-states (s,T) such that $acc \in T$.

Intuitively, proposition $\#$ marks only the $\mathcal{K}$-states which are targets of pairs in $R^{\#}_{\Gamma}(\mathcal{K}) \cup R^{\#}_{\Gamma}(\mathcal{K}, F)$, while the flag acc marks either the states in F which are targets of $\mathcal{K}$-edges, or the $\mathcal{K}$-states which are targets of pairs in $R_{\Gamma}(\mathcal{K}, F) \cup R^{\#}_{\Gamma}(\mathcal{K}, F)$. We say that a trace σ over $V \cup \{\#\}$ is *well-formed* if one of the following conditions holds:

- *or* σ is a Γ-stutter free trace over V (i.e. $stfr_{\Gamma}(\sigma) = \sigma$);
- *or* there is a position $i \geq 0$ such that $i+1$ is the unique position where $\#$ holds and $stfr_{\Gamma}(\sigma) = \sigma(0) \ldots \sigma(i) \cdot \sigma^{i+2}$.

By construction, it easily follows that $stfr^{\#}_{\Gamma}(T(\mathcal{K}, F))$ is the set of *well-formed* traces in $T(\mathcal{K}_{\Gamma}, F_{\Gamma})$. Then, the LTL formula θ_{Γ} captures the well-formed requirement and is defined as follows.

$$\neg\# \wedge \mathbf{G}(\# \rightarrow \mathbf{XG}\neg\#) \wedge \mathbf{G}\Big(\bigvee_{p \in \Gamma}\big((p \leftrightarrow \neg\mathbf{X}p) \wedge \neg\# \wedge \mathbf{X}\neg\#\big) \vee \bigwedge_{p \in \Gamma}\mathbf{G}\big((p \leftrightarrow \mathbf{X}p) \wedge \neg\#\big) \vee \big(\mathbf{X}\# \wedge \bigwedge_{p \in \Gamma}(p \leftrightarrow \mathbf{X}p) \wedge \bigvee_{p \in \Gamma}(p \leftrightarrow \neg\mathbf{X}^2\neg p)\big)\Big)$$

This concludes the proof of Proposition 8.5. □

Fix now a singleton-free sentence φ of SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$. For the given fair finite Kripke structure $(\mathcal{K}, F)$ over V, let $(\mathcal{K}_\Gamma, F_\Gamma)$ and θ_Γ as in the statement of Proposition 8.5. We consider the SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$ sentence $\mathsf{T}(\varphi)$ obtained from $\mathsf{T}_{\#}(\varphi)$ by inductively replacing each subformula $\exists^{\mathsf{P}} x.\, \phi$ of $\mathsf{T}_{\#}(\varphi)$ with $\exists^{\mathsf{P}} x.\, (\mathsf{T}(\phi) \wedge \langle x \rangle \mathbf{O}(\neg \mathbf{Y} \top \wedge \theta_\Gamma[x]))$. In other terms, we ensure that in $\mathsf{T}_{\#}(\varphi)$ the hyper quantification ranges over traces which satisfy the LTL formula θ_Γ. By Remark 8.1 and Proposition 8.5, we obtain that $T(\mathcal{K}, F) \models \varphi$ if and only if $T(\mathcal{K}_\Gamma, F_\Gamma) \models \mathsf{T}(\varphi)$. Thus, together with Proposition 8.2, we obtain the following result.

Theorem 8.5 *The (fair) model checking problem against simple GHyperLTL$_{\mathsf{S+C}}$ can be reduced in singly exponential time to fair model checking against SHyperLTL$^{\emptyset}_{\mathsf{S+C}}$.*

8.3 SC-HyperLTL: Asynchronous HyperLTL over finite traces

In the previous section, we introduced a general logic for asynchronous hyperproperties over infinite traces. In this section, we focus on a significant prenex fragment of Simple GHyperLTL$_{\mathsf{S+C}}$, designed for reasoning over finite traces. This fragment simplifies the logic by disallowing trace quantification within formulae, as well as quantifiers of the form $\exists^{\mathsf{P}} x$ and $\forall^{\mathsf{P}} x$. Due to these restrictions, the logic becomes simpler than the original Simple SHyperLTL$_{\mathsf{S+C}}$. We therefore introduce a lighter syntax and semantics tailored specifically to this fragment.

We formalize its finite-trace semantics, illustrate its expressiveness through representative examples, and present two model checking procedures for verifying properties specified in this logic.

In the remainder of this section, we refer to a slightly different definition of finite languages of STS by treating F as the final states represented symbolically i.e. each finite σ of $\mathcal{L}_{fin}(M)$ satisfies each $f \in F$ at point $|\sigma| - 1$.

The syntax of SC-HyperLTL is:

$$\alpha := \forall x.\alpha \mid \exists x.\alpha \mid \Gamma.\varphi$$
$$\varphi := \langle \{x\} \rangle \beta[x] \mid a[x] \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\, \mathbf{U}\, \varphi \mid \mathbf{Y}\varphi \mid \varphi\, \mathbf{S}\, \varphi$$

where $a \in V$, Γ is a set of PLTL formulae over V, β is a PLTL formula over V and $\beta[x]$ is defined by replacing each variable $a \in V$ occurring in β with $a[x]$.

To define the semantics, we adapt the PLTL stutter factorization of Definition 8.1 to finite traces. To be symmetric, we include both the initial position and the final position in the factorization.

Definition 8.3 (LTL_f-stutter factorization) *Let Γ be a finite set of PLTL formulae and σ a finite trace. The Γ-stutter factorization of σ is the unique increasing sequence of positions $\{i_0 \dots i_k\}$ such that $i_0 = 0$, $i_k = |\sigma| - 1$ and:*

- *For each $\theta \in \Gamma$ and j, $0 \leq j < k$, the truth value of θ along $[i_j, i_{j+1})$ does not change, i.e., for all $h, l \in [i_j, i_{j+1})$, $(\sigma, h) \models \theta$ if and only if $(\sigma, l) \models \theta$.*

- *The truth value of some formula in Γ changes along adjacent segments (except at i_k), i.e., for each $0 \leq j < k - 1$, for some $\theta \in \Gamma$, $(\sigma, i_j) \models \theta$ if and only if $(\sigma, i_{j+1}) \not\models \theta$.*

In order to define the *relevant positions*, we define a finite version of the successor relation. We use the special symbol `und` (meaning undefined) to denote the successor of the last point in the trace and the predecessor of the first point in the trace. Intuitively, the new modification renders the definition of successor (*succ*) and predecessor (*pred*) specular.

Definition 8.4 (Relativized successor with finite semantics) *Let Γ be a finite set of PLTL formulae and σ a finite trace. The relativized successor and relativized predecessor are defined as follows:*

$$succ_\Gamma(\sigma, i) := \begin{cases} \texttt{und} & \text{If } i \geq |\sigma| - 1 \\ (\sigma, i_{j+1}) & \text{If } i \in [i_j, i_{j+1}) \text{ for some } j \in [0, k) \end{cases}$$

We extend the previous definitions to the trace assignment Π as follows; $succ_\Gamma(\texttt{und}) = \texttt{und}$ if there is a trace variable $x \in \text{VAR}$ such that the successor $succ_\Gamma(\Pi)(x) = \texttt{und}$. Otherwise, $succ_\Gamma(\Pi)(x) = succ_\Gamma(\Pi(x))$ for each $x \in \text{VAR}$.

We define the semantics of SC-HyperLTL as follows:

$$
\begin{array}{rll}
\Pi & \models_T \exists x.\alpha & \Leftrightarrow \text{for some } \sigma \in T,\ \Pi[x \mapsto (\sigma, 0)] \models_T \alpha \\
\Pi & \models_T \forall x.\alpha & \Leftrightarrow \text{for all } \sigma \in T,\ \Pi[x \mapsto (\sigma, 0)] \models_T \alpha \\
\Pi & \models_T \Gamma.\varphi & \Leftrightarrow (\Pi, \Gamma) \models \varphi \\
(\Pi, \Gamma) & \models a[x] & \Leftrightarrow a \in \sigma(p),\ \text{where } (\sigma, p) = \Pi(x) \\
(\Pi, \Gamma) & \models \varphi_1 \,\mathbf{U}\, \varphi_2 & \Leftrightarrow \text{for some } j \geq 0\ \ succ_\Gamma^j(\Pi) \neq \texttt{und}, (succ_\Gamma^j(\Pi), \Gamma) \models \varphi_2, \\
 & & \quad \text{and for all } i < j, (succ_\Gamma^i(\Pi), \Gamma) \models \varphi_1 \\
(\Pi, \Gamma) & \models \mathbf{X}\varphi & \Leftrightarrow succ_\Gamma(\Pi) \neq \texttt{und} \text{ and } (succ_\Gamma(\Pi), \Gamma) \models \varphi \\
(\Pi, \Gamma) & \models \varphi_1 \,\mathbf{S}\, \varphi_2 & \Leftrightarrow \text{for some } j \geq 0\ \ pred_\Gamma^j(\Pi) \neq \texttt{und}, (pred_\Gamma^j(\Pi), \Gamma) \models \varphi_2, \\
 & & \quad \text{and for all } i < j, (pred_\Gamma^i(\Pi), \Gamma) \models \varphi_1 \\
(\Pi, \Gamma) & \models \mathbf{Y}\varphi & \Leftrightarrow pred_\Gamma(\Pi) \neq \texttt{und} \text{ and } (pred_\Gamma(\Pi), \Gamma) \models \varphi \\
(\Pi, \Gamma) & \models \langle\{x\}\rangle\beta[x] & \Leftrightarrow (\sigma, i) \models \beta \text{ with } (\sigma, i) = \Pi(x)
\end{array}
$$

For an SC-HyperLTL sentence α and a set of *finite* traces T, T is a *model* of φ, written $T \models \alpha$, if $(\Pi_0, \text{VAR}) \models_T \alpha$. Finally, given an STS M$= \langle V, I, T\rangle$, we say that $M \models_f \alpha$ iff $\mathcal{L}_{fin}(M) \models \alpha$.

Properties expressible in SC-HyperLTL

In Section 8.1.4, we presented various properties expressible in the more expressive SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ fragment, showcasing how our temporal logics can capture rich system behaviours. Although SC-HyperLTL is strictly less expressive, it remains well-suited for specifying meaningful and practical properties that are essential for verifying many classes of systems. In particular, we explore how these properties—and the examples that illustrate them—can be interpreted not only over infinite traces, as in the original logic, but also in settings where executions are finite. This finite-trace perspective is especially relevant in practice, where systems often have bounded or terminating behaviours such as transactional systems or terminating programs in general.

We now revisit key properties—some of which have already been introduced over SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$—and show how they can also be captured within SC-HyperLTL, with attention to their finite-trace semantics. We further extend this discussion by presenting additional structured examples from the literature, as well as a new example model, to illustrate the expressiveness and practical relevance of SC-HyperLTL in finite-trace verification tasks.

Noninference [243]: Non-inference states that the observations of public information do not change when the secret information is removed. We can define this property for the asynchronous setting with the following formula:

$$\forall x.\exists y.\{lo\}.\langle\{x\}\rangle \mathbf{G}\lambda[x] \wedge \mathbf{G}(lo[x] \leftrightarrow lo[y]),$$

where lo represents the public information and λ represents the absence of secret inputs.

Observational Determinism[298]: We previously introduced a specification of observational determinism in Section 8.1.4 using the more expressive $\text{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ fragment. Here we provide an equivalent formulation in SC-HyperLTL

$$\forall x.\forall y.\{LO\}.\bigwedge_{v\in LI}(v[x] \leftrightarrow v[y]) \rightarrow \bigwedge_{v\in LO}\mathbf{G}(v[x] \leftrightarrow v[y]),$$

where LI and LO denote the sets of low inputs and low outputs, respectively. This formulation enforces that executions with identical low inputs produce indistinguishable low outputs throughout their finite executions. Being interpreted over finite traces, the specification considers any couple of traces (even with different lengths) to be evaluated up to the end of the shortest trace.

Bounded Recovery. Consider the high level model shown in Fig. 8.1, which describes a battery sensor. To deal with hardware faults in the sensor, a second sensor is used as a backup, and a Selector component chooses which sensor output (sensed1/sensed2) shall be considered. The selector component is connected with the Recovery component which detects faults like abnormal oscillations and triggers a recovery action by sending a "recovery" signal (dashed blue line). The Selector component will then select Sensor2 as the main sensor. We assume that faults are permanent and that sensors run asynchronously.

For any execution with a fault ($\mathit{fault} \doteq \mathit{fault}_1 \vee \mathit{fault}_2$), all observationally equivalent executions (from the Recovery component perspective) trigger a recovery command within a bounded amount of steps, where an observation occurs when *sensed* changes:

$$\varphi_{rec} := \forall x.\forall y.\{\mathit{sensed}\}.\mathbf{G}(\mathit{obseq} \wedge \langle\{x\}\rangle \mathit{faulted}[x] \rightarrow \langle\{y\}\rangle \mathbf{F}^{\leq d} \mathit{rcv}[y])$$

where $\mathbf{F}^{\leq d}\phi$ is the bounded future operator defined as $\phi \vee \mathbf{X}(\mathbf{F}^{\leq d-1}\phi)$ with $\mathbf{F}^{\leq 0} := \phi$, $\mathit{obseq} := \mathbf{H}(\mathit{sensed}[x] \leftrightarrow \mathit{sensed}[y])$, $\mathit{faulted} := (\neg \mathit{chd(sensed)}\ \mathbf{S}\ \mathit{fault})$ and *chd(sensed)* determines whether the sensed value changed.

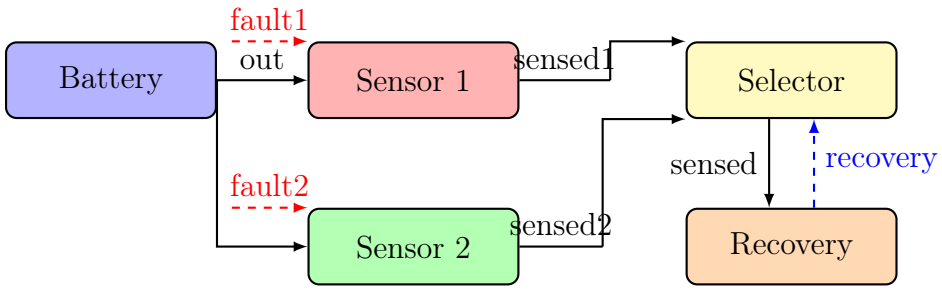


Figure 8.1: Redundant Sensor System with Fault Detection, Isolation, and Recovery component schema. The coloured rectangles represent the internal components, the black arrow represents the data-port connection between components, the red dashed arrows represent fault events and, the blue dashed arrow represent the recovery signal.

The set $\{sensed\}$ is used to consider only the relevant points in which the sensed value changes. The formula *obseq* ensures that the two traces have the same sequences of sensed values. Then, $faulted[x]$ states that a fault occurred between the last observation point and the current one. Finally, the consequence is that the recovery command is activated within d steps in the related trace y.

Finite programs pre-post conditions Due to our finite-trace semantics, we can also express $\forall\exists$ pre-post conditions over different executions of the same program, even when these executions have different and unbounded lengths. In addition, we can enforce local properties over individual executions by using singleton contexts, allowing us to specify standard temporal properties alongside hyperproperties within a unified framework.

$$\Phi_{pp} \doteq \forall x.\forall y\exists z.\emptyset.\langle\{x\}\rangle\beta_1[x] \wedge \Phi_{Pre}(x,y) \rightarrow \bigwedge_{v\in V}(v[y] = v[z]) \wedge \langle\{z\}\rangle\beta_2[z] \wedge \mathbf{X}\Phi_{Post}(x,z)$$

The first singleton-context formula assess a local property β_1 over the universally quantified execution (trace variable x). The precondition is expressed over an additional universally quantified trace variable y. The intuition is that we are restricting the universal traces – represented by x – to those that has another trace such that conjoined they satisfy the precondition. The right-side of the implication first ensures that (indirectly) the existential trace also satisfy the precondition (with x); then, it expresses its local property β_2 and – at the end of both traces i.e. x, z – the post-condition is satisfied.

8.3.1 Decidability of SC-HyperLTL

In the following, we show a reduction from SC-HyperLTL to $\text{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ via a rewriting function for the formula and a modified version of the STS to self-loop in accepting states. From this reduction we can then conclude that model checking of SC-HyperLTL over finite traces is *decidable*.

Definition 8.5 (Padding of STS) *Let M be an STS such that $end \notin V$. We denote the STS extending M with self-loops over accepting states as*

$$\mathcal{E}nd(M) \doteq \langle V \cup \{end\}, I \wedge \neg end, \mathcal{E}nd(T), F \cup \{end\}\rangle$$

where the transition relation is represented by the following formula:

$$\mathcal{E}nd(T) \doteq (end \rightarrow end') \wedge (\neg end' \rightarrow T) \wedge \left(\neg end' \rightarrow \bigwedge_{v \in V} v' = v\right)$$

Definition 8.6 (Finite Traces SC-HyperLTL to Infinite Traces $\text{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$) *Let φ be a SC-HyperLTL formula, we inductively define ζ^f as follows:*

$$\begin{array}{rclcrcl}
\zeta^f(\Gamma, v[x]) & := & v[x] & & \zeta^f(\Gamma, \varphi_1 \vee \varphi_2) & := & \varphi_1^f \vee \varphi_2^f \\
\zeta^f(\Gamma, \neg\varphi) & := & \neg\varphi^f & & \zeta^f(\Gamma, \langle\{x\}\rangle\beta[x]) & := & \langle\{x\}\rangle\zeta^f(\emptyset, \beta[x]) \\
\zeta^f(\Gamma, \boldsymbol{X}\varphi) & := & \boldsymbol{X}_{\Gamma^f}(\theta^f_{Dom(\Pi)} \wedge \varphi^f) & & \zeta^f(\Gamma, \varphi_1 \,\mathbf{U}\, \varphi_2) & := & \varphi_1^f \,\mathbf{U}_{\Gamma^f}\, (\theta^f_C \wedge \varphi_2^f) \\
\zeta^f(\Gamma, \boldsymbol{Y}\varphi) & := & \boldsymbol{Y}_{\Gamma^f}\varphi^f & & \zeta^f(\Gamma, \varphi_1 \,\mathbf{S}\, \varphi_2) & := & \varphi_1^f \,\mathbf{S}_{\Gamma^f}\, \varphi_2^f
\end{array}$$

where $\theta^f_C := \bigwedge_{x \in C}(\neg \boldsymbol{X}_{\emptyset} end[x])$ and $\varphi^f := \zeta^f(C, \varphi)$.

Finally, we denote the rewriting for quantified formulae ζ^f as

$$\begin{array}{rclcrcl}
\zeta^f(\forall x.\varphi) & := & \forall x.\zeta^f(\varphi) & & \zeta^f(\exists x.\varphi) & := & \exists x.\zeta^f(\varphi) \\
\zeta^f(\Gamma.\varphi) & := & \zeta^f(\Gamma, \varphi) & & & &
\end{array}$$

where $\Gamma^f := \Gamma \cup \{\boldsymbol{X} end\}$ if $\Gamma \neq \emptyset$, $\emptyset$ otherwise.

Proposition 8.6 *Let M be an STS and φ be a SC-HyperLTL formula, $M^{inf} := \mathcal{E}nd(M)$.*

$$M \models_f \varphi \text{ iff } M^{inf} \models \zeta^f(\varphi)$$

Proof: To prove the proposition, we consider the following intermediate claim over the satisfiability of quantifier free formulae with complete trace assignments (i.e. trace assignments with domain equal to VAR).

Let Π be a complete trace assignment with finite traces of M with all traces pointing to the initial position. Let $\mathcal{E}nd(\Pi)$ be defined as the complete trace assignment with infinite traces over $\mathcal{E}nd(M)$, in which each trace is obtained padding the end of each trace of Π.

For each i, if $succ^i_\Gamma(\Pi) \neq \texttt{und}$: $(succ^i_\Gamma(\Pi), \Gamma) \models_f \varphi \Leftrightarrow succ^i_{(\Gamma,\text{VAR})}(\Pi) \models \zeta^f(\Gamma, \varphi)$

Before starting the proof, we observe that (*)if $succ_\Gamma(\Pi) \neq \texttt{und}$, $succ_{(\Gamma^f,\text{VAR})}(\mathcal{E}nd(\Pi))$ is the trace assignment augmented with end of $succ_\Gamma$. Moreover, (**)if $\Pi \neq \texttt{und}$: $pred_\Gamma(\Pi) = pred_{\Gamma^f}(\Pi^{end})$ Finally, (***)$succ_\Gamma(\Pi) = \texttt{und} \Leftrightarrow succ_{\Gamma^f} \not\models \theta^f_{\text{VAR}}$.

Base case: Trivially follows from the fact that the traces have the same assignments to the variables.

Inductive case: We skip all the trivial cases, we also skip past because is symmetric to future:

- $\langle\{x\}\rangle\alpha[x]$: This is proved by the reduction from finite to infinite of LTL_f described in the literature (e.g. [183]). $\zeta^f(\emptyset, \alpha[x])$ produces a $\text{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ without contexts, which, since the evaluation context is VAR is satisfied iff α is satisfied with finite trace semantics by the pointed trace of $\Pi(x)$.

- $\mathbf{X}\psi$: $(\Pi, \Gamma) \models_f \mathbf{X}\psi \Leftrightarrow succ_\Gamma(\Pi) \neq \texttt{und}$ and $succ_\Gamma(\Pi) \models_f \psi$. $succ_\Gamma(\Pi) = \texttt{und}$ iff exists $x \in Dom(\Pi)$ .s.t. $(\sigma, |\sigma|-1) = \Pi(x) \Leftrightarrow$ exists $x \in Dom(\Pi)$ s.t. $\sigma, j \models \mathbf{X}end$ with $(\sigma, j) = \Pi^{end}(x) \Leftrightarrow \Pi^{end} \not\models \theta^f_{Dom(\Pi)}$. Finally, by (*): $\Leftrightarrow \Pi^{end} \models \theta^f_C$ and $succ_{\Gamma^f}(\Pi^f) \models \zeta^f(\psi) \Leftrightarrow (\Pi^{end}, \Gamma) \models \zeta^f(\mathbf{X}, Dom(\Pi))$

- $\psi_1 \mathbf{U} \psi_2$: $(\Pi, \Gamma) \models_f \psi_1 \mathbf{U} \psi_2 \Leftrightarrow$ there exists $k \geq 0$ s.t. $(succ^k_\Gamma(\Pi) \neq \texttt{und}$, $(succ^k_\Gamma, \Gamma) \models_f \psi$ and for all $0 \leq j < k : (succ^j_\Gamma(\Pi), \Gamma) \models_f \psi_1$. By (*) and (***): $\Leftrightarrow$ there exists $k \geq 0$ s.t. $(succ^k_{\Gamma^f}(\Pi^f), \Gamma^f) \models \theta^f_{Dom(\Pi)}$ and $(succ^k_{\Gamma^f}(\Pi^f), \Gamma^f) \models \zeta^f(\psi_2, Dom(\Pi))$ and for all $0 \leq j < k : succ^j_{\Gamma^f}$.

Finally, from the inductive claim it is straightforward to generalize the induction with quantifiers Γ operator. □

Theorem 8.6 *Model Checking of SC-HyperLTL is decidable.*

Proof: The decidability results trivially from Proposition 8.6 since $\zeta^f(\varphi)$ is still in the decidable fragment of $\text{GHyperLTL}_{\mathsf{S+C}}$. □

8.3.2 Reversibility property of SC-HyperLTL

We now introduce an interesting property of our logic on finite traces. One very nice property of LTL_f has is that past and future are symmetric. In practice if we flip a trace and replace future operators with their corresponding past operators and vice-versa, we obtain the same satisfiability result. Moreover, given an STS $M = \langle V, I, T, F\rangle$, we can construct the STS outputting the "reversed" finite language as $M^{rev} \doteq \langle V, \bigwedge_{f\in F} f, T^{rev}, \{I\}\rangle$ where T^{rev} is obtained by replacing each variable of V with its primed version (of V') and vice-versa and initial condition (I) and final (F) are flipped. This is not true in the general definition of STS language given in 2.4; however, to avoid conflict with the alternative finite semantics introduced elsewhere, we assume an adjusted interpretation in which each $f \in F$ must hold at the end of the trace.[1] Formally, reversibility of LTL_f is expressed by the following Proposition.

Proposition 8.7 (Reverse of LTL_f) *Let $M = \langle V, I, T, F\rangle$ be an STS*

$$M \models_f \varphi \text{ iff } M^{rev} \models_f rev(\varphi)$$

where $rev(\varphi)$ replaces each future operator with its part counterpart and vice-versa. Intuitively, the past counterpart of $\boldsymbol{X}$ is $\boldsymbol{Y}$ and for $\mathbf{U}$ is $\mathbf{S}$.

Proof: It is easy to see that M^{rev} produce the reversed trace of M. We can then find a one to one matching between a trace σ and its reversed counterpart denoted as σ^{-1}. The trace σ^{-1} has the following characteristics:

- The reverse trace has the same length as the original trace: $|\sigma^{-1}| = |\sigma|$
- $\sigma(i) = \sigma^{-1}(\bar{i})$ with $\bar{i} \doteq |\sigma| - i - 1$

Finally, we can prove that for each i $\sigma, i \models_f \varphi$ iff $\sigma^{-1}, \bar{i} \models_f rev(\varphi)$. This result follows from induction on the structure of the formula and trivially each case follows from the semantics of each operator and its past/future counterpart. □

We can lift this results from LTL_f to SC-HyperLTL. The intuition is that the rewriting of *singleton-context* operators is the same of LTL_f, while for each temporal operator interpreted under Γ, we also need to "reverse" Γ. The idea in this case is that we

[1] This special interpretation is used solely for this section, to avoid introducing a second, distinct notion of finite language that would otherwise be unnecessary and potentially confusing in the broader context of the thesis.

consider interesting point comparing the previous point of the trace with the current one. Therefore, when we reverse the trace, we need to evaluate that Γ change in the future (see Γ_R in the following definition).

Definition 8.7 (Reverse SC-HyperLTL formula) *Let φ be a SC-HyperLTL formula. We define $rev(\varphi)$ as follows (here, $\Gamma_R := \{\boldsymbol{X} rev(\theta) | \theta \in \Gamma\}$):*

$$
\begin{array}{rcl@{\qquad}rcl}
rev(\forall x.\varphi) & := & \forall x.rev(\varphi) & rev(\boldsymbol{X}\psi) & := & \boldsymbol{Y} rev(\psi) \\
rev(\exists x.\varphi) & := & \exists x.rev(\varphi) & rev(\boldsymbol{Y}\psi) & := & \boldsymbol{X} rev(\psi) \\
rev(\Gamma.\psi) & := & \Gamma_R.rev(\psi) & rev(\langle\{x\}\rangle\psi) & := & \langle\{x\}\rangle rev(\psi) \\
rev(v[x]) & := & v[x] & rev(\psi_1 \mathbf{U} \psi_2) & := & rev(\psi_1) \mathbf{S} rev(\psi_2) \\
rev(\psi_1 \vee \psi_2) & := & rev(\psi_1) \vee rev(\psi_2) & rev(\psi_1 \mathbf{S} \psi_2) & := & rev(\psi_1) \mathbf{U} rev(\psi_2) \\
rev(\neg\psi) & := & \neg rev(\psi) & & &
\end{array}
$$

The reverse of a formula considers the trace in the opposite time direction, which is enabled by having future and past operators and traces having a beginning and an end. Since Γ points are evaluated considering the points with different past evaluation, we need to reverse Γ as well. Given a trace σ we use σ^{-1} for the trace such that $\sigma^{-1}(i) = \sigma(N-i)$ where $N = |\sigma| - 1$. We use Π^{-1} for the trace assignment that assigns σ^{-1} to x when Π assigns σ to x.

To prove reversability, we first introduce an intermediate lemma *highlighting* the correspondence between the reverse of the relativized Γ-successor of a trace assignment Π with the relativized Γ_R-predecessor of the reverse of Π. Formally

Lemma 8.1 (Successor-Predecessor Equivalence) *Let Π be a trace assignment over finite traces and Γ be a set of PLTL formulae:*

$[succ_\Gamma(\Pi)]^{-1} = pred_{\Gamma_R}(\Pi^{-1})$ *and* $[pred_\Gamma(\Pi)]^{-1} = succ_{\Gamma_R}(\Pi^{-1})$

Proof: To prove the Lemma, we consider an intermediate claim. Let $i_0, \ldots, i_k$ be the Γ-stutter factorization of a finite trace σ. The sequence $\overline{i_k}, \ldots, \overline{i_{k-1}}, \ldots \overline{i_0}$ is the Γ_R-stutter factorization of the finite trace σ^{-1}.

We prove the claim each by showing that each characteristic of Definition 8.3 is satisfied by $\overline{i_k}, \ldots, \overline{i_{k-1}}, \ldots \overline{i_0}$.

- It is easy to see that $\overline{i_k} = 0$ and $\overline{i_0} = |\sigma^{-1}| - 1$.
- Consider the following statement of the stutter factorization definition: For all $h, l \in [i_j, i_{j+1})$, $(\sigma, h) \models \theta$ if and only if $(\sigma, l) \models \theta$. Since σ^{-1} is the reverse

of σ and by LTL_f reversability, then for all $\overline{h}, \overline{l} \in (\overline{i_{j+1}}, \overline{i_j}]$, $(\sigma^{-1}, \overline{h}) \models_f rev(\theta)$ iff $(\sigma^{-1}, \overline{l}) \models_f rev(\theta)$. Since neither h nor l are 0 or $|\sigma| - 1$, we can derive the following result over $\mathbf{X}rev(\theta)$.

For all $\overline{h}, \overline{l} \in (\overline{i_{j+1}}, \overline{i_j}]$ $(\sigma^{-1}, \overline{h} - 1) \models_f \mathbf{X}rev(\theta)$ iff $(\sigma^{-1}, \overline{l} - 1) \models_f \mathbf{X}rev(\theta)$ from which we can simplify -1 fixing the interval as: For all $\overline{h}, \overline{l} \in [\overline{i_{j+1}}, \overline{i_j})$ $(\sigma^{-1}, \overline{h}) \models_f \mathbf{X}rev(\theta)$ iff $(\sigma^{-1}, \overline{l}) \models_f \mathbf{X}rev(\theta)$. Since Γ_R renders each $\theta \in \Gamma$ into $\mathbf{X}rev(\Gamma)$, we proved this condition for the new sequence.

- For each $0 \leq j < k - 1$, for some $\theta \in \Gamma$, $(\sigma, i_j) \models_f \theta$ if and only if $(\sigma, i_{j+1}) \not\models_f \theta$. From which we derive that $(\sigma^{-1}, \overline{i_j} \models_f rev(\theta)$ if and only if $(\sigma^{-1}, \overline{i_{j+1}}) \not\models_f rev(\theta)$ and consequently that $(\sigma^{-1}, \overline{i_j - 1} \models_f \mathbf{X}rev(\theta)$ iff $(\sigma^{-1}, \overline{i_{j+1}} - 1) \not\models_f \mathbf{X}rev(\theta)$. We can now apply the property proved in the previous bullet (i.e. stuttering of positions) and the following result.

  For $0 \leq j < k - 1$ $(\sigma^{-1}, \overline{i_{j+1}} \models_f \mathbf{X}rev(\theta)$ iff $(\sigma^{-1}, \overline{i_{j+2}}) \not\models_f \mathbf{X}rev(\theta)$. From which we easily obtain that for $0 < j < k$ $(\sigma^{-1}, \overline{i_j} \models_f \mathbf{X}rev(\theta)$ iff $(\sigma^{-1}, \overline{i_{j+1}}) \not\models_f \mathbf{X}rev(\theta)$.

  Let's consider the reversed sequence $\overline{i_k} \ldots \overline{i_0}$ as $i'_0, \ldots i'_k$: Then we obtain that for $0 \leq j < k - 1$ $(\sigma^{-1}, i'_j) \models_f \mathbf{X}rev(\theta)$ iff $(\sigma^{-1}, i'_{j+1}) \not\models_f rev(\theta)$.

Given the intermediate claim, the Lemma trivially follows. □

Finally, with the intermediate results we can derive the following theorem on the reversability of SC-HyperLTL

Theorem 8.7 (Reversibility of SC-HyperLTL) *Let φ be a SC-HyperLTL formula, let Π be a trace assignment,*

$$(\Pi, \Gamma) \models_f \varphi \Leftrightarrow (\Pi^{-1}, \Gamma_R) \models_f rev(\varphi)$$

Moreover, $rev(rev(\varphi))$ and φ are equisatisfiable. Finally, if the traces of M have the same amount of observation points, we show that model checking has the reversability property. Formally, if $M \models_f \forall x_A \forall x_B . \Gamma . \boldsymbol{G}(\langle\{x\}\rangle \boldsymbol{N}\bot[x] \leftrightarrow \langle\{y\}\rangle \boldsymbol{N}\bot)$, then

$$M \models_f \varphi \text{ iff } M^{rev} \models_f \boldsymbol{G}(\boldsymbol{N}\bot \rightarrow rev(\varphi))$$

Proof: We prove the first part of the theorem inductively on the structure of the formula.

The base case trivially holds because the current positions are the same assignments.

We consider each inductive case separately. We omit trivially inductive cases and past cases (which are symmetric w.r.t. future cases).

- $\langle\{x\}\rangle$: The singleton case follows from Proposition 8.6.

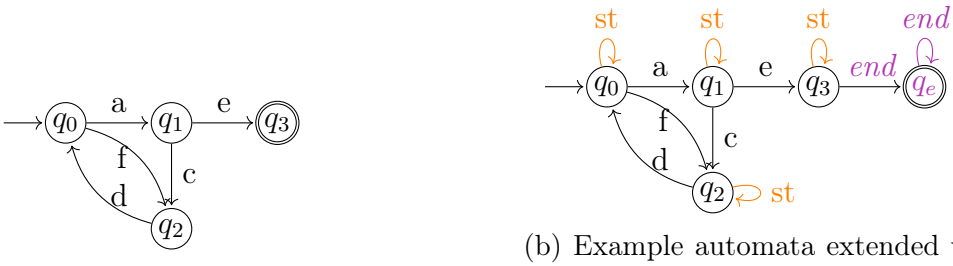
- $\mathbf{X}$: $(\Pi, \Gamma) \models_f \mathbf{X}\varphi$ iff $succ_\Gamma(\Pi) \neq \texttt{und}$ and $(succ_\Gamma(\Pi), \Gamma) \models_f \varphi$. By induction we get that it holds iff $[succ_\Gamma(\Pi)]^{-1} \neq \texttt{und}$ and $([succ_\Gamma(\Pi)]^{-1}, \Gamma_R) \models_f rev(\varphi)$ By Lemma 8.1 it is true iff $pred_{\Gamma_R}(\Pi^{-1}) \neq \texttt{und}$ and $(pred_{\Gamma_R}(\Pi^{-1}), \Gamma_R) \models_f rev(\varphi)$ iff $(\Pi^{-1}, \Gamma_R) \models_f \mathbf{Y} rev(\varphi)$ iff $(\Pi^{-1}, \Gamma_R) \models_f rev(\mathbf{X}\varphi)$

- $\mathbf{U}$: $(\Pi, \Gamma) \models_f \varphi_1 \mathbf{U} \varphi_2$ iff there is a $k \geq 0$ s.t. $succ^k_\Gamma(\Pi) \neq \texttt{und}$, $(succ^k_\Gamma(\Pi), \Gamma) \models_f \varphi_2$ and for each $0 \leq j < k$: $(succ^j_\Gamma(\Pi), \Gamma) \models_f \varphi_1$. By induction we get that it holds iff there is a $k \geq 0$ s.t. $[succ^k_\Gamma(\Pi)]^{-1} \neq \texttt{und}$, $([succ^k_\Gamma(\Pi)]^{-1}, \Gamma_R) \models_f rev(\varphi_2)$ and for each $0 \leq j < k$: $([succ^j_\Gamma(\Pi)]^{-1}, \Gamma_R) \models_f rev(\varphi_1)$. By Lemma 8.1 it is true iff there is a $k \geq 0$ s.t. $pred^k_{\Gamma_R}(\Pi^{-1}) \neq \texttt{und}$, $(pred^k_{\Gamma_R}(\Pi^{-1}), \Gamma_R) \models_f rev(\varphi_2)$ and for each $0 \leq j < k$: $(pred^j_{\Gamma_R}(\Pi^{-1}), \Gamma_R) \models_f rev(\varphi_1)$ iff $(\Pi^{-1}, \Gamma_R) \models_f rev(\varphi_1)\, \mathbf{S}\, rev(\varphi_2)$ iff $(\Pi^{-1}, \Gamma_R) \models_f rev(\varphi_1\, \mathbf{U}\, \varphi_2)$.

By applying two consecutive times the first part of the theorem we trivially derive the equivalence between φ and $rev(rev(\varphi))$.

Let Π_0^{-1} be the reversed trace assignment where each reversed trace points to 0.

$(\Pi_0^{-1}, \Gamma_R) \models_f \mathbf{G}(\mathbf{N}\bot \rightarrow rev(\varphi))$ iff for each $j \geq 0$: such that $succ_j \Gamma_R \neq \texttt{und}$: $(succ^j_{\Gamma_R}(\Pi_0^{-1}), \Gamma_R) \models_f \mathbf{N}\bot \rightarrow rev(\varphi)$ iff for each $j \geq 0$: such that $succ^j_{\Gamma_R} \neq \texttt{und}$ and $succ^{j+1}_{\Gamma_R} = \texttt{und}$: $(succ^j_{\Gamma_R}(\Pi_0^{-1}), \Gamma_R) \models_f \mathbf{N}\bot \rightarrow rev(\varphi)$. There is only one possible position j being the last defined successor. Moreover, since each trace has the same amount of observations, from the finite semantics we obtain that for each $x_i \in \text{VAR}$: $succ^j_{\Gamma_R}(\Pi_0^{-1})(x_i) = (\sigma_i^{-1}, |\sigma_i^{-1}| - 1)$. From which we obtain that such successor is indeed Π^{-1}. Finally, we can apply the previous part of the theorem and related Π_0^{-1} with Π satisfaction. Then applying trivially inductive reasoning on quantifiers and Γ-operator, we obtain the equivalence of the two model checking queries (assuming same amount of observations). $\square$



(a) Example automata

(b) Example automata extended with stuttering transitions (in orange)

Figure 8.2: Graphical intuitive view of stuttering extension of the automata. Note that variables and symbols labelling the transitions are slightly different of the one actually used.

8.4 Model Checking algorithms for SC-HyperLTL

8.4.1 Stuttering Encoding

The first algorithm we present adapts the approach presented in Chapter 5 of rewriting of LTL formulae and asynchronous composition of STS. The technique is efficient because it allows for a direct reduction to LTL_f model checking (previously presented in Chapter 4). However, it is limited to the quantifier-alternation free fragment of SC-HyperLTL. The high-level intuition of the approach is the following:

1. We apply the asynchronous composition γ_S introduced in Section 5.1.4 (redefined as self-composition) to the STS M for each trace variable. An intuitive view of the extension of the single automaton with stuttering (then composed with the other automaton copies) is depicted in Figure 8.2.

2. We introduce a synchronization constraint that aligns observation points of each trace variable exploiting the *asynchronous* composition. Intuitively, two traces are aligned adding *stuttering* transition to one of the two traces. Figure 8.3 provides an intuitive view of the approach.

3. We combine the synchronization constraint with a variation of the rewriting $\mathcal{R}^{\mathcal{H}}$ introduced for the compositional verification of $\text{LTL}(\mathcal{T})$ in Section 5.2. The reader can refer to Figure 5.5 and Figure 5.4 for an intuitive view of the approach[2].

Although omitted in this section, these results can be straightforwardly extended to the infinite-trace semantics; moreover, for the finite case semantics one can also reduce to standard HyperLTL model checking instead of LTL_f considering simple rewritings

[2]Please note that while the rewriting is almost identical, here the setting is completely different.

$$\mathbf{G}(\neg end[x] \wedge \neg end[y] \rightarrow (\Phi_\Gamma[x] \leftrightarrow \Phi_\Gamma[y]))$$

Figure 8.3: This figure shows two traces of the original STS on the left, and the extended version with the corresponding extended traces on the right, after asynchronous composition. The traces on the right are *aligned* and satisfy the alignment constraint.

from the finite semantics to the infinite semantics and extending the STS with stuttering instead of directly applying asynchronous composition – as instead shown in the following definition.

The model transformation is as follows.

Definition 8.8 (Finite Asynchronous Self Composition) *Let $M = \langle V, I, T, F\rangle$ be an STS, let $x_1, \ldots, x_k \in VAR$ be k trace variables. The Finite Asynchronous K-Self Composition of M with trace variables $x_1, \ldots, x_k$ is defined as the STS $M^k_{\mathcal{A}}$ defined as*

$$M^k_{\mathcal{A}} := \bigotimes_{x_i \in \{x_1,\ldots,x_k\}} M[x_i]$$

where $M[x_i]$ is defined replacing each occurrence of variables of V with $V[x_i]$ in I and T, and substituting V with $V[x_i]$ and $\bigotimes$ is taken from Definition 5.3[3].

Essentially, $M^k_{\mathcal{A}}$ represents the k-self-composition in which each copy of the STS can either stutter or run. Each copy of M can also construct traces with arbitrary length, the composed language would consist of a trace that at some point reaches a halting state and then stutters forever.

Definition 8.9 (Alignment under stuttering) *Let Γ be a finite set of PLTL formulae, $x_1, \ldots, x_n$ be a collection of trace variables. We define θ^{st}_Γ as follows:*

$$\theta^{st}_\Gamma := \boldsymbol{G}(\neg end^{st} \rightarrow \bigwedge_{1<i\leq n} (\theta_\Gamma[x_i] \leftrightarrow \theta_\Gamma[x_1])), \qquad \textit{where}$$

- $\theta_\Gamma := first^{st} \vee last^{st} \vee \bigvee_{\theta\in\Gamma}(\boldsymbol{Y}\theta \leftrightarrow \neg\theta)$ *represents the observation points,*

[3]We recall the fact that STS is a special case of ITS. Moreover, since all the "fairness" (recall that final states are treated like fairness) constraints are unconditional, also the resulting composed STS can have unconditional "fairness".

- $end^{st} := \bigvee_{1 \leq i \leq n} end[x_i]$ *models the termination of some trace,*
- $first^{st} := \boldsymbol{Z}\bot$ *captures the first state of a trace (the same for all traces), and*
- $last^{st} := \neg end \wedge \boldsymbol{X} end$ *represents the last state.*

Def. 8.9 introduces the alignment constraints of the traces at the observation points. The alignment is preserved up to the "end" of the "shortest" stuttered trace. The formula θ_Γ captures observation points, which include the initial state $first^{st}$ and the final state of the trace $last^{st}$. The antecedent of the implication holds whenever one of the traces has not surpassed the final state yet, while the consequent is the same truth value of observations of each trace. This constraint synchronously evaluates the observation points by padding the non-aligned observations using stuttering expansion.

Definition 8.10 (Stuttering Compositional Trace) *Let* Π *be a trace assignment over* M *with* $Dom(\Pi) = \{x_1, \ldots, x_n\}$*, we denote the trace* $\sigma^{\otimes}_{\mathcal{A}}$ *over the n-asynchronous self-composition of* M *as a stuttering compositional trace of* Π *if the following conditions hold:*

1. *The asynchronous compositional trace is constructed composing local traces of* Π*: for each* $i \in \{x_1, \ldots, x_n\}$*:* $Pr_{M[x_i]}(\sigma^{\otimes}_{\mathcal{A}}) = \Pi(x_i)\lceil V/V[x_i]\rfloor$.
2. *All traces are padded with end: for each* i*:* $\sigma^{\otimes}_{\mathcal{A}}, |\sigma^{\otimes}_{\mathcal{A}}| - 1 \models_f end[x_i]$.
3. *The stuttered traces are aligned over observations (changes in* Γ*) up to the last observation of the trace with fewer observations:* $\sigma^{\otimes}_{\mathcal{A}} \models_f \theta^{st}_{\Gamma}$.

Lemma 8.2 (Existence of Stuttering Compositional Trace) *Let* Π *be a trace assignment over STS* M *with* $Dom(\Pi) = \{x_1, \ldots, x_n\}$*. There exists a trace* $\sigma^{\otimes}_{\mathcal{A}}$ *of* $M^n_{\mathcal{A}}$ *that is a Stuttering Compositional Trace.*

Proof: The first and the second point trivially follows from the definition of the composition and from the fact that self-composition preserves local-traces (i.e. if M has a trace σ in its language, the trace is also part of the composition).

It is easy to see that, each trace $\sigma_i = \Pi(x_i)$ can be expanded in any arbitrary position using stuttering transitions (with $run = \bot$). Since each copy also copies all the symbols, there is no "interference" between the local traces. Therefore, we can simply add an arbitrary amount of stuttering transition to each trace to provide such alignment. □

We now introduce a variation of rewriting of formulae of Section 5.2 adapted for SC-HyperLTL. This rewriting $\mathcal{R}^{\mathcal{H}}$ takes two quantifier-free and Γ-free SC-HyperLTL formulae ψ and η and produces a quantifier-free HyperLTL formula:

Definition 8.11 (Stuttering Rewriting of SC-HyperLTL) *Let φ be a quantifier free SC-HyperLTL formula without Γ operator and η a q.f. HyperLTL formula. We define the rewriting $\mathcal{R}^{\mathcal{H}}_{\eta}(\psi)$ as follows:*

$$\begin{aligned}
\mathcal{R}^{\mathcal{H}}_{\eta}(v[x]) &:= v[x] \\
\mathcal{R}^{\mathcal{H}}_{\eta}(\neg\psi) &:= \neg\mathcal{R}^{\mathcal{H}}_{\eta}(\psi) \\
\mathcal{R}^{\mathcal{H}}_{\eta}(\psi_1 \vee \psi_2) &:= \mathcal{R}^{\mathcal{H}}_{\eta}(\psi_1) \vee \mathcal{R}^{\mathcal{H}}_{\eta}(\psi_2) \\
\mathcal{R}^{\mathcal{H}}_{\eta}(\langle\{x\}\rangle\psi) &:= \mathcal{R}^{\mathcal{H}}_{state[x]}(\psi) \\
\mathcal{R}^{\mathcal{H}}_{\eta}(\boldsymbol{X}\psi) &:= \boldsymbol{X}(\neg\eta\ \mathbf{U}\ (\eta \wedge \mathcal{R}^{\mathcal{H}}_{\eta}(\psi))) \\
\mathcal{R}^{\mathcal{H}}_{\eta}(\boldsymbol{Y}\psi) &:= \boldsymbol{Y}(\eta\ \mathbf{S}\ (\neg\eta \wedge \mathcal{R}^{\mathcal{H}}_{\eta}(\psi))) \\
\mathcal{R}^{\mathcal{H}}_{\eta}(\psi_1\ \mathbf{U}\ \psi_2) &:= (\eta \vee \mathcal{R}^{\mathcal{H}}_{\eta}(\psi_1))\ \mathbf{U}\ (\neg\eta \wedge \mathcal{R}^{\mathcal{H}}_{\eta}(\psi_2)) \\
\mathcal{R}^{\mathcal{H}}_{\eta}(\psi_1\ \mathbf{S}\ \psi_2) &:= (\eta \vee \mathcal{R}^{\mathcal{H}}_{\eta}(\psi_1))\ \mathbf{S}\ (\neg\eta \wedge \mathcal{R}^{\mathcal{H}}_{\eta}(\psi_2))
\end{aligned}$$

where $state[x]$ (with state introduced in Definition 5.6) represents the point of the local trace that neither stutter nor loop after termination.

Finally, let $\varphi \doteq \Gamma.\psi$ and let x_1 be a trace variable occurring in ψ. We denote $\mathcal{R}^{\mathcal{H}}(\varphi) := \mathcal{R}^{\mathcal{H}}_{\theta_\Gamma[x_1]\wedge\neg end^{st}}$.

Temporal operator formulae are evaluated at positions where η is false (that is, aligned at non-stuttering positions). For example, $\mathcal{R}^{\mathcal{H}}_{\eta}(\mathbf{X}p[x])$ would be true under the HyperLTL semantics only if at the next occurrence of $\neg\eta$ the variable $p[x]$ is true. A detailed description of the LTL part of the rewriting can be found in Section 5.2.

For singleton-context formulae, η is replaced for the subformula with $state[x]$ that represents points of the local traces that neither stutter neither loop after termination. The intuition is that, with singleton context, we want to only evaluate the points that are relevant for the local trace. For non-singleton formulae η is set to $\theta_\Gamma[x_1]$ with $x_1 \in \text{VAR}$ being one of the trace variables of the overall formula. Therefore, each temporal operator is evaluated only at the observation points of x_1 in which no trace has terminated yet. Provided the alignment of Definition 8.9, the observation points that need to be evaluated with finite semantics are the ones in which $\theta_\Gamma[x_1] \wedge \neg end^{st}$ is satisfied.

Proposition 8.8 *Let $\langle\{x\}\rangle\alpha[x]$ be a formula, let Π be a trace assignment and let $\sigma^{\otimes}_{\mathcal{A}}$ be a stuttering compositional trace of Π.*

$$(\Pi, \Gamma) \models_f \langle\{x\}\rangle\alpha[x] \text{ iff } \sigma^{\otimes}_{\mathcal{A}} \models_f \mathcal{R}^{\mathcal{H}}_{state[x]}(\alpha[x])$$

Proof: We can inductively prove the property over the structure of the formula. We omit the proof which adapts the one of Lemma 5.1 of Chapter 5 to our setting (with LTL_f). □

Proposition 8.9 *Let ψ be a SC-HyperLTL q.f. formula without Γ (corresponding to the syntactic category ϕ in the grammar defining the syntax of SC-HyperLTL in Section 8.3), let Π be a trace assignment mapping each trace to the j-th position of the Γ-stutter factorization and let $\sigma^{\otimes}_{\mathcal{A}}$ be a stuttering compositional trace of Π.*

$$(\Pi, \Gamma) \models_f \psi \Leftrightarrow \sigma^{\otimes}_{\mathcal{A}}, j_\Gamma \models_f \mathcal{R}^{\mathcal{H}}(\psi)$$

where j_Γ is the j-th position in which one of the PLTL Γ formula changes.

Proof: To prove the Lemma, we apply induction over the size of the formula.

Base case holds because the assignment and the self-composition trace point to the same assignments for the variables.

Inductive case:

- $(\vee, \neg)$ Trivially follows from induction and operator semantics
- $(\mathbf{X})$: We can split the proof in two cases: $succ_\Gamma(\Pi) = \texttt{und}$ or $succ_\Gamma(\Pi) \neq \texttt{und}$.

  If the successor is undefined, then there is one trace variable x s.t. $(\sigma, |\sigma| - 1) = \Pi(x)$. Thus, the extended asynchronous composed trace $\sigma^{\otimes}_{\mathcal{A}} \models_f \mathbf{X}end[x]$. Since in $M^k_{\mathcal{A}}$ each local symbol of each copy of the automaton after ends become, end is true forever and each variable stutters forever. Therefore, $\mathcal{R}^{\mathcal{H}}(\mathbf{X}\psi)$ is violated by $\sigma^{\otimes}_{\mathcal{A}}$ because the right side of until is never satisfied.

  If $succ_\Gamma(\Pi) \neq \texttt{und}$, then by definition $succ_\Gamma(\Pi)$ points each trace variable to the $j+1$-th point in which one of each $\theta \in \Gamma$ changes. By induction: $(succ_\Gamma(\Pi), \Gamma) \models_f \psi$ iff $\sigma^{\otimes}_{\mathcal{A}}, [j+1]_\Gamma \models_f \mathcal{R}^{\mathcal{H}}(\psi)$. We can apply the same approach used to prove Lemma 5.1.
- $(\mathbf{U})$: As for next, you can prove it inductively as done in Lemma 5.1.
- $(\langle\{x\}\rangle\alpha[x])$: Follows from Proposition 8.8.

- (Past operators): identical to future.

□

Theorem 8.8 *Let M be an STS and ψ be a quantifier-free SC-HyperLTL formula. Then,*

$$M \models_f \forall x_n \dots \forall x_1.\Gamma.\psi \quad \textit{iff} \quad \bigotimes_{x_i \in VAR} (M[x_i]) \models_f \bigwedge_{x_i \in VAR} (\boldsymbol{F}end[x_i]) \wedge \theta_\Gamma^{st} \rightarrow \mathcal{R}^{\mathcal{H}}(\psi)$$

$$M \models_f \exists x_n \dots \exists x_1.\Gamma.\psi \quad \textit{iff} \quad \bigotimes_{x_i \in VAR} (M[x_i]) \not\models_f \bigwedge_{x_i \in VAR} (\boldsymbol{F}end[x_i]) \wedge \theta_\Gamma^{st} \rightarrow \mathcal{R}^{\mathcal{H}}(\neg\psi)$$

Proof: From Proposition 8.9 we have a correspondence between a trace assignment Π and the composition. Since the initial state is an *observation* point, we easily derive that $\Pi \models_f \Gamma.\psi$ iff $\sigma_{\mathcal{A}}^{\otimes} \models_f \mathcal{R}^{\mathcal{H}}(\psi)$. Observing that each traces of the self-composition has a corresponding trace assignment, we can inductively prove over the trace variables that the theorem holds. □

8.4.2 Model Checking with Bounded Observations

We now introduce an alternative algorithm to deal with asynchrony for formulae with arbitrary quantifier alternations. This algorithm introduces k history variables $\{v_1, \dots v_k\}$ for each variable v occurring inside the formula. The intended meaning of v_j is to mimic the value that v has at the j-th position of the stuttering factorization of the trace. Then, the variables are used in a BMC-like approach [47] to check whether the property holds on a horizon with at most k observations. If k observations are sufficient to prove or disprove the property then the outcome is informative. Note that the bound k is on the number of observations and not on the length of the trace, which is unbounded. Moreover, in principle with this approach is possible to check for formulae in which more than one set of LTL property Γ is present. The idea is that with bounded-observations asynchrony can always be removed. The overall process is depicted in Figure 8.4

In this section, we introduce the following notations that we are going to use to prove the correctness of our approach.

Let M be an STS, $\alpha := \mathcal{Q}_n x_n \dots \mathcal{Q}_1 x_1.\Gamma.\psi$ be a SC-HyperLTL formula where Γ is a set of LTL formulae over $\overline{V} \subseteq V$.

Let $\tilde{\sigma}_1, \dots \tilde{\sigma}_n \in \mathcal{L}_{fin}(M)$ be n traces of M. Let $\tilde{\Pi}$ be the trace assignment s.t. for all $x_j \in$ VAR: $\tilde{\Pi}(x_j) = (\tilde{\sigma}_j, 0)$. Finally, we denote $\tilde{\Pi}_i := succ_\Gamma^i(\tilde{\Pi})$.

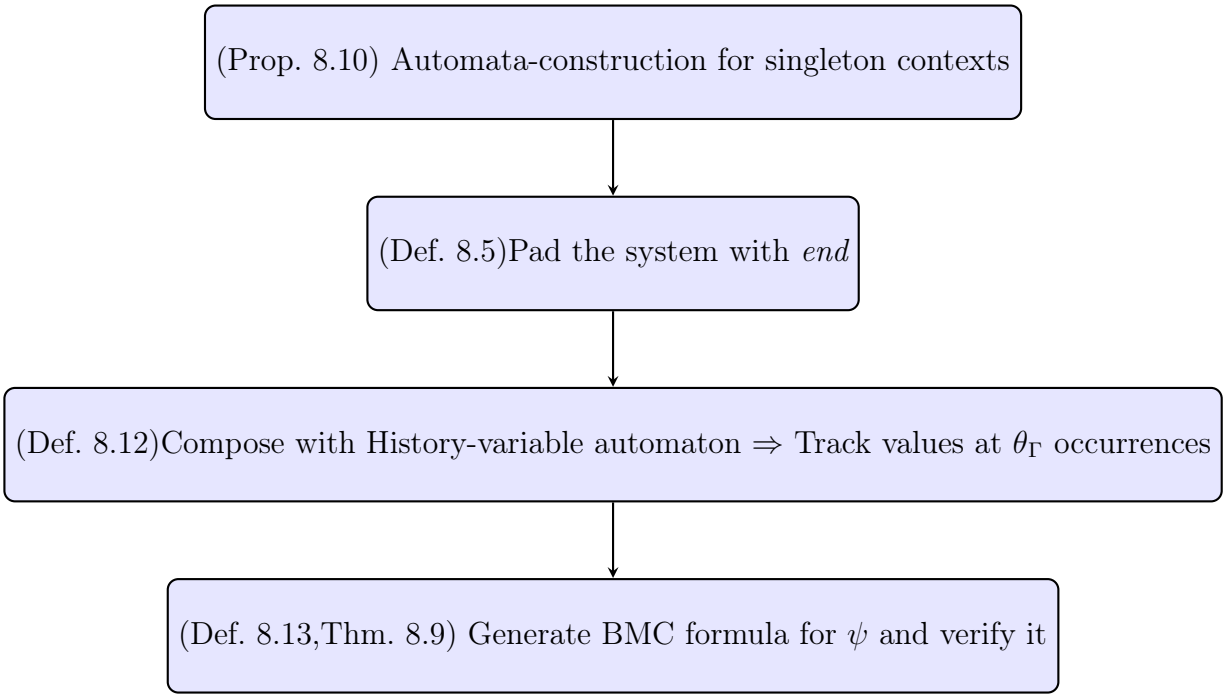


Figure 8.4: High-level step-by-step view of the k-bound transformation approach.

The first step of our approach consists in removing singleton expressions from the formula. Every single context sub-formula can be translated into automata with standard techniques (refer to Chapter 4 for details), and the sub-formula can be replaced by a fresh variable as shown in Prop. 8.10 below. Given an LTL_f formula β over variables $\hat{V}$, we recall the notation from Chapter 4 M_β as the STS with alphabet $V \cup \{v_\beta\}$ such that for each (finite) pointed trace (σ, i), $(\sigma, i) \models \beta \Leftrightarrow (\sigma, i) \models_f v_\beta$. We use $Sub(\varphi)$ for the set of all the sub-formulae of φ.

Proposition 8.10 (Removing singleton context) *Let M be an STS and φ be a SC-HyperLTL formula. $M \models_f \varphi$ if and only if $(M \times \bigtimes_{\langle\{x_i\}\rangle\beta[x_i] \in Sub(\varphi)} M_\beta) \models_f \varphi'$, where φ' is obtained by replacing each occurrence of $\langle\{x_i\}\rangle\beta[x_i]$ with $v_\beta[x_i]$.*

Proof: The proposition trivially follows from the standard results on the automata-based construction of LTL_f. The semantics in SC-HyperLTL of singleton operators reflects the exact finite semantics of the logics. □

Definition 8.12 (History-variable STS) *Let M be an STS, $\overline{V} \subseteq V$ be a set of variables, $\overline{V}^k := \{v_i \mid v \in \overline{V}, 0 \leq i < k\}$ be the set containing k copies of the variables of $\overline{V}$ and pos be an enumerative variable ranging from 0 to $k+1$.*

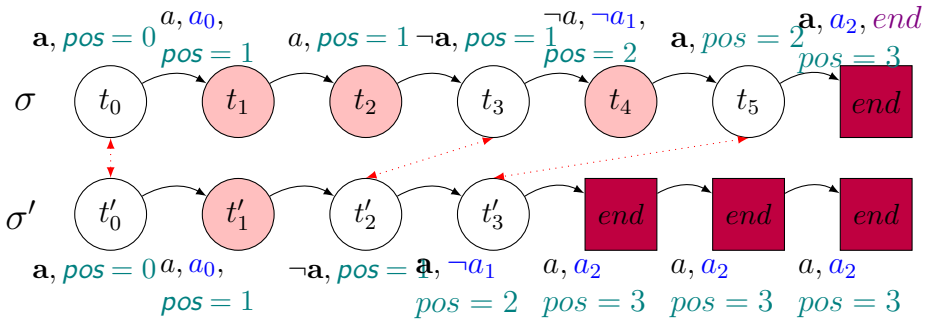


Figure 8.5: Traces σ, σ' with the construction in Algorithm 2 for $k = 3$ and $\Gamma = \{a\}$. History variables are shown in blue; the end of the local traces are encoded by *end* variables, shown in purple; *pos* variables are shown in teal. Red dotted lines map the Γ-points of σ with the corresponding Γ-points of σ'. Pink circles represent the non-interesting trace points and purple boxes the end of the traces. The trace σ' is shorter than σ, so σ' is extended to be aligned with σ in the very last position according to Def. 8.5.

We define the STS $P^k_{\theta_\Gamma,\overline{V}} := \langle V \cup \overline{V}^k \cup \{pos, v_{\theta_\Gamma}, end\}, pos = 0, T^k \rangle$ *with*

$$T^k := \begin{pmatrix} (obs_\Gamma \rightarrow pos' = pos + 1) \wedge (\neg obs_\Gamma \rightarrow pos' = pos) \wedge \\ \bigwedge_{v_i \in \overline{V}^k} ((pos = i \wedge obs_\Gamma \rightarrow v_i' = v) \wedge (pos \neq i \vee \neg obs_\Gamma \rightarrow v_i' = v_i)) \end{pmatrix}$$

where v_{θ_Γ} *represents the points in which elements of* Γ *changed,* $obs_\Gamma := \neg end \wedge (v_{\theta_\Gamma} \vee end' \vee pos = 0)$ *represents the observation points in the trace i.e. either the points in which elements of* Γ *change, the initial state or the last point of the trace.*

Def. 8.12 introduces the behaviour of history variables. Variable *pos* counts the observations seen so far, initialized to 0, and increased each time an observation is encountered. If more than k observation are encountered, *pos* takes the value $k + 1$ (encoding that more than k observations have happened). Each variable v_i mimics the value of v at the i-th observation. When $pos = i$, it takes the current value of v ($v_i' = v$), otherwise it stutters ($v_i' = v_i$). It should be noted that, before the i-th observation, v_i can take any value. Variable v_{θ_Γ} represents the points of observations, where one of the $\theta \in \Gamma$ changes. Figure 8.5 shows two traces of a system, with $\Gamma = \{a\}$ and $k = 3$. Proposition 8.11 shows the relationship, already depicted in Figure 8.5, between the traces of M and $M^k_{\mathcal{B}}$.

Proposition 8.11 (History variables monitoring) *For each* $\tilde{\sigma} \in \mathcal{L}_{fin}(M)$*, for each* $\sigma \in \mathcal{L}_{fin}(M^k_{\mathcal{B}})$ *such that* $\sigma(V) = \tilde{\sigma}$ *(recall that traces of* M *are preserved in* $M^k_{\mathcal{B}}$*), for*

each $v \in \overline{V}$ *and for each* $0 \leq i < k$:

$$\textit{If } \sigma, |\sigma| - 1 \models_f pos > i, \textit{then } \begin{pmatrix} \sigma, |\sigma| - 1 \models_f v_i \Leftrightarrow \\ succ^i_\Gamma(\tilde{\sigma}, 0) \models_f v \end{pmatrix}$$

Moreover, $\sigma, |\sigma| - 1 \models_f pos \leq i : succ^i_\Gamma(\tilde{\sigma}, 0) = \textbf{\textit{und}}$.

Proof: We prove the proposition by considering the following claims that combined entail it. Let

1. For all $0 \leq j < |\sigma| : \sigma, j \models_f pos = j \wedge obs_\Gamma \Leftrightarrow (\tilde{\sigma}, j) = succ^i_\Gamma(\tilde{\sigma}, 0)$
2. Let $(\tilde{\sigma}, j) = succ^i_\Gamma(\tilde{\sigma}, 0)$:
   (a) $(\sigma, j) \models_f v \Leftrightarrow (\sigma, j+1) \models_f v_i$
   (b) For all $j' > j : (\sigma, j) \models_f v_i \Leftrightarrow (\sigma, j') \models_f v_i$

Claim 1 can be easily proved by induction on i:

$(i = 0)$: $succ^0_\Gamma(\tilde{\sigma}, 0) = (\tilde{\sigma}, 0)$, Since $pos = 0$ at the initial state (due to $P^k_{\theta_\Gamma, \overline{V}}$) and obs_Γ is true if $pos = 0$ then one direction is proved. The other direction follows because *pos* is increased when obs_Γ holds and never decreases; that means that *pos* is 0 initially, then it becomes 1 and then it remains weakly monotonic.

$(i - 1 \Rightarrow i)$: We know that $succ^i_\Gamma(\tilde{\sigma}, 0) := succ_\Gamma(succ^{i-1}_\Gamma(\tilde{\sigma}, 0))$. By induction, we know that either $succ^{i-1}_\Gamma(\tilde{\sigma}, 0) = \texttt{und}$ or there is a unique point j s.t. $(\sigma, j) = succ^{i-1}_\Gamma(\tilde{\sigma}, 0)$ and $\sigma, j \models_f obs_\Gamma \wedge pos = i - 1$.

If $succ^{i-1}_\Gamma(\tilde{\sigma}, 0) = \texttt{und}$, then $succ^i_\Gamma(\tilde{\sigma}, 0) = \texttt{und}$ as well; since *pos* changes only by increasing by one exactly in the points in which obs_Γ is true, then if there is no point s.t. $obs_\Gamma \wedge pos = i - 1$ is true, there is no point s.t. $obs_\Gamma \wedge pos = i$ as well.

If $succ^{i-1}_\Gamma(\tilde{\sigma}, 0) = (\sigma, j)$, then $succ_\Gamma(\tilde{\sigma}, j) = succ^i_\Gamma(\tilde{\sigma}, 0)$. Since obs_Γ captures the semantics of observation points (initial state, tail states or points in which previously $\theta \in \Gamma$ changed) then either $succ^i_\Gamma(\tilde{\sigma}, 0) = \texttt{und}$ or $succ^i_\Gamma(\tilde{\sigma}, 0) = (\sigma, j')$ s.t. $j' > j$ and for all $j < j'' < j' : \sigma, j'' \not\models_f obs_\Gamma$. Since *pos* changes only if obs_Γ is true, then either $pos = i \wedge obs_\Gamma$ is never true in the trace or it will be true exactly at point j' with $(\sigma, j') = succ^i_\Gamma(\sigma, 0)$.

Claim 2 instead follows by the definition of $P^k_{\theta_\Gamma, \overline{V}}$. It constraints v_i to take the value of v at the i-th observation and then makes v_i stutters forever (because $pos = i \wedge obs_\Gamma$ is true at most once).

Given the two claims, the first part of the proposition follows from Claim 2 and part b of claim 2. The second part of the proposition follows from Claim 1. □

Definition 8.13 (K-bound unrolling) *We define the bounded unrolling of quantifier free SC-HyperLTL formulae up to k. The complete encoding assumes formulae in negation normal form i.e. in which negations only occur over predicates.*

${}^{\rho}[\![\varphi]\!]^k_k := \rho$. *For all* $i < k$:

$$
\begin{array}{rcl@{\qquad}rcl}
{}^{\rho}[\![v[x]]\!]^k_i & := & v_i[x] & {}^{\rho}[\![\neg v[x]]\!]^k_i & := & \neg v_i[x] \\
{}^{\rho}[\![\psi_1 \vee \psi_2]\!]^k_i & := & {}^{\rho}[\![\psi_1]\!]^k_i \vee {}^{\rho}[\![\psi_2]\!]^k_i & {}^{\rho}[\![\psi_1 \wedge \psi_2]\!]^k_i & := & {}^{\rho}[\![\psi_1]\!]^k_i \wedge {}^{\rho}[\![\psi_2]\!]^k_i \\
{}^{\rho}[\![\boldsymbol{X}_\Gamma \psi]\!]^k_i & := & \neg end^{i+1}_\Gamma \wedge {}^{\rho}[\![\psi]\!]^k_{i+1} & {}^{\rho}[\![\boldsymbol{N}_\Gamma \psi]\!]^k_i & := & end^{i+1}_\Gamma \vee {}^{\rho}[\![\psi]\!]^k_{i+1} \\
{}^{\rho}[\![\psi_1 \,\mathbf{U}_\Gamma\, \psi_2]\!]^k_i & := & {}^{\rho}[\![\psi_2]\!]^k_i \vee ({}^{\rho}[\![\psi_1]\!]^k_i \wedge {}^{\rho}[\![\boldsymbol{X}\top]\!]^k_{i+1} \wedge {}^{\rho}[\![\psi_1 \,\mathbf{U}_\Gamma\, \psi_2]\!]^k_{i+1}) & & & \\
{}^{\rho}[\![\psi_1 \,\mathbf{R}_\Gamma\, \psi_2]\!]^k_i & := & {}^{\rho}[\![\psi_2]\!]^k_i \wedge ({}^{\rho}[\![\psi_1]\!]^k_i \vee {}^{\rho}[\![\boldsymbol{N}\bot]\!]^k_{i+1} \vee {}^{\rho}[\![\psi_1 \,\mathbf{R}_\Gamma\, \psi_2]\!]^k_{i+1}) & & & \\
{}^{\rho}[\![\boldsymbol{Y}_\Gamma \psi]\!]^k_0 & := & \bot \quad {}^{\rho}[\![\boldsymbol{Z}_\Gamma \psi]\!]^k_0 := \top \quad {}^{\rho}[\![\boldsymbol{Y}_\Gamma \psi]\!]^k_i := {}^{\rho}[\![\psi]\!]^k_{i-1} \quad {}^{\rho}[\![\boldsymbol{Z}_\Gamma \psi]\!]^k_i := {}^{\rho}[\![\psi]\!]^k_{i-1} & & & \\
{}^{\rho}[\![\psi_1 \,\mathbf{S}_\Gamma\, \psi_2]\!]^k_i & := & {}^{\rho}[\![\psi_2]\!]^k_i \vee ({}^{\rho}[\![\psi_1]\!]^k_i \wedge {}^{\rho}[\![\boldsymbol{Y}\top]\!]^k_i \wedge {}^{\rho}[\![\psi_1 \,\mathbf{S}_\Gamma\, \psi_2]\!]^k_{i-1}) & & & \\
{}^{\rho}[\![\psi_1 \,\mathbf{T}_\Gamma\, \psi_2]\!]^k_i & := & {}^{\rho}[\![\psi_2]\!]^k_i \wedge ({}^{\rho}[\![\psi_1]\!]^k_i \vee ({}^{\rho}[\![\boldsymbol{Z}\bot]\!]^k_i \vee {}^{\rho}[\![\psi_1 \,\mathbf{T}_\Gamma\, \psi_2]\!]^k_{i-1})) & & &
\end{array}
$$

where $end^i_\Gamma := \bigvee_{x \in VAR}(pos[x] \leq i)$, $\rho \in \{\bot, \top\}$.

The formula ${}^{\rho}[\![\psi]\!]^k_0$ introduces an unrolling of formulae following the finite semantics of the operators, using the history variables as if they were synchronous projections of each trace and then traverse the traces synchronously. Since each trace might have a different number of observations, the encoding considers the end of the projected trace as the position i such that at least one trace has $pos = i$ (end^i_Γ). Lemma 8.3 shows the satisfaction relationship between the unrolling and our original property.

Lemma 8.3 *Let* $\sigma_1, \ldots, \sigma_n$ *be* n *traces of* $M^k_{\mathcal{B}}$ *and* $\tilde{\sigma}_1, \ldots, \tilde{\sigma_n}$ *the corresponding trace of* M *(i.e. traces s.t. For all* $0 \leq i < |\tilde{\sigma}|$*:* $\sigma(i)(V) = \tilde{\sigma}(i)(V)$ *and* $\sigma, |\tilde{\sigma}| \models_f end$*), let* Π *be trace assignment over VAR s.t. for all* $x_i \in VAR : \Pi(x_i) := (\sigma_i, 0)$*, let* $\tilde{\Pi}$ *be the trace assignment over VAR s.t. for all* $x_i \in VAR : \Pi(x_i) := (\tilde{\sigma}, 0)$ *and let* ψ *be a q.f. SC-HyperLTL formula without singleton contexts.*

If there is a $\tilde{\sigma}_i$ *s.t.* θ_Γ *occurs at most* k *times:*

$$(\tilde{\Pi}, \Gamma) \models_f \psi \Leftrightarrow \Pi \models_f \boldsymbol{G}(\boldsymbol{N}\bot \rightarrow {}^{\rho}[\![\psi]\!]^k_0)$$

Proof: We prove the Lemma using the following statement: For all i: $\tilde{\Pi}_i \neq \texttt{und} \Rightarrow \tilde{\Pi}_i \models_f \psi \Leftrightarrow \Pi^e \models_f {}^{\rho}[\![\psi]\!]^k_i$ where Π^e points each σ to position $|\sigma| - 1$. The statement can be proved by induction on the structure of the formula.

Base case: $\tilde{\Pi}_i \models_f v[x] \Leftrightarrow succ^i_\Gamma(\tilde{\sigma}, 0) \models_f v$. By Proposition 8.11, if $\sigma, |\sigma|-1 \models_f pos > i$, then $\sigma, |\sigma| - 1 \models_f v_i \Leftrightarrow succ^i_\Gamma(\tilde{\sigma}, 0) \models_f v$. Therefore, we deduce that $\sigma, |\sigma - 1| \models_f v_i \Leftrightarrow succ^i_\Gamma(\tilde{\sigma}, 0) \models_f v$.

We can ignore the base case with $i = k$ since by assumption $\Pi_{k+1} = \texttt{und}$, thus ${}^\rho[\![\psi]\!]^k_k$ is never reached (to be reached *pos* must be $= k + 1$ in each trace).

Inductive case:

- $(\vee, \wedge)$: Trivial induction.
- $\mathbf{X}\psi$: $(\tilde{\Pi}_i, \Gamma) \models_f \mathbf{X}\psi \Leftrightarrow \tilde{\Pi}_{i+1} \neq \texttt{und}$ and $(\tilde{\Pi}_{i+1}, \Gamma) \models_f \psi$.

  By Proposition 8.11 and due to the semantics of the successor, we know that $\tilde{\Pi}_{i+1} \neq \texttt{und} \Leftrightarrow \Pi^e \models_f \neg end^{i+1}_\Gamma$. Moreover, by induction if $\tilde{\Pi}_{i+1} \neq \texttt{und}$: $(\tilde{\Pi}_{i+1}, \Gamma) \models_f \psi \Leftrightarrow \Pi^e \models_f {}^\rho[\![\psi]\!]^k_{i+1}$. Therefore, we can conclude that $(\tilde{\Pi}_i, \Gamma) \models_f \mathbf{X}\psi \Leftrightarrow \Pi^e \models_f \neg end^{i+1}_\Gamma \wedge {}^\rho[\![\psi]\!]^k_{i+1} \Leftrightarrow \Pi \models_f {}^\rho[\![\mathbf{X}\psi]\!]^k_i$.

- $\mathbf{N}_\Gamma\psi$: The proof is identical to the one of $\mathbf{X}$ with the minor difference that $\Pi_{i+1} \neq \texttt{und}$ and $\neg end^{i+1}_\Gamma$ appear with an implication instead of a conjunction.

- $\psi_1 \,\mathbf{U}\, \psi_2$: $(\tilde{\Pi}_i, \Gamma) \models_f \psi_1 \,\mathbf{U}\, \psi_2 \Leftrightarrow$ exists $l \geq i$ s.t. $\tilde{\Pi}_l \neq \texttt{und}$, $(\tilde{\Pi}_l, \Gamma) \models_f \psi_2$ and for all $i \leq j < l$: $(\tilde{\Pi}_j, \Gamma) \models_f \psi_1$. Combining Proposition 8.11, due to the successor semantics (same as $\mathbf{X}$) and using induction, we obtain $\cdots \Leftrightarrow \Pi^e \models_f \bigvee_{i \leq l \leq k} (\neg end^i_\Gamma \wedge {}^\rho[\![\psi_2]\!]^k_l \wedge \bigwedge_{i \leq j < l} ({}^\rho[\![\psi_1]\!]^k_j))$. We observe that the previous expression is equivalent to the unrolling: ${}^\rho[\![\psi_2]\!]^k_i \vee {}^\rho[\![\psi_1]\!]^k_i \wedge \neg end^{i+1}_\Gamma \wedge {}^\rho[\![\psi_1 \,\mathbf{U}\, \psi_2]\!]^k_{i+1} (\Leftrightarrow {}^\rho[\![\psi_1 \,\mathbf{U}\, \psi_2]\!]^k_i)$. Thus, we can conclude that $(\tilde{\Pi}_i, \Gamma) \models_f \psi_1 \,\mathbf{U}\, \psi_2 \Leftrightarrow \Pi^e \models_f {}^\rho[\![\psi_1 \,\mathbf{U}_\Gamma\, \psi_2]\!]^k_i$.

- $\psi_1 \,\mathbf{R}\, \psi_2$: The proof for release follows a similar procedure of until.

- $\mathbf{Y}\psi$: We split the two cases:

  1. $i = 0$: By definition $\tilde{\Pi}_i \not\models_f \mathbf{Y}\psi$. By ${}^\rho[\![\mathbf{Y}_\Gamma\psi]\!]^k_0$ definition, $\Pi^e \not\models_f {}^\rho[\![\mathbf{Y}_\Gamma\psi]\!]^k_0$.
  2. $0 < i < k$: $\tilde{\Pi}_i \models_f \mathbf{Y}_\Gamma\psi \Leftrightarrow \tilde{\Pi}_{i-1} \neq \texttt{und}$ and $\tilde{\Pi}_{i-1} \models_f \psi$. Since $i > 0$, $\tilde{\Pi}_{i-1} \neq \texttt{und}$. Therefore, we obtain $\tilde{\Pi}_i \models_f \mathbf{Y}\psi \Leftrightarrow \tilde{\Pi}_{i-1} \models_f \psi$. By induction, we obtain $\cdots \Leftrightarrow \Pi^e \models_f {}^\rho[\![\psi]\!]^k_{i-1}$ proving the equivalence.

- $\mathbf{Z}\psi$: Proof almost identical to yesterday (case 1 is true instead of false).

- $\psi_1 \,\mathbf{S}\, \psi_2$: $(\tilde{\Pi}_i, \Gamma) \models_f \psi_1 \,\mathbf{S}\, \psi_2 \Leftrightarrow$ exists $l \leq i$ s.t. $(\tilde{\Pi}_l, \Gamma) \neq \texttt{und}$, $\tilde{\Pi}_l \models_f \psi_2$ and for all $l < j \leq i$: $(\tilde{\Pi}_j, \Gamma) \models_f \psi_1 \Leftrightarrow \Pi^e \models_f \bigvee_{0 \leq l \leq k} ({}^\rho[\![\psi_2]\!]^k_l \vee \bigwedge_{l < j \leq i} {}^\rho[\![\psi_1]\!]^k_j)$. We observe

that the previous expression is equivalent to the unrolling: ${}^{\rho}[\![\psi_2]\!]^k_i \vee {}^{\rho}[\![\psi_1]\!]^k_i \wedge {}^{\rho}[\![\mathbf{Y}\top]\!]^k_{i-1} \wedge {}^{\rho}[\![\psi_1 \,\mathbf{S}\, \psi_2]\!]^k_{i-1}$.

- $\psi_1 \,\mathbf{T}\, \psi_2$: Proof almost identical to since

Finally, we need to prove that $\Pi^e \models_f {}^{\rho}[\![\psi]\!]^k_i \Leftrightarrow \Pi \models_f \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\rho}[\![\psi]\!]^k_i)$.

We recall that $\mathbf{N}\bot$ is satisfied by a trace assignment at the end of the shortest trace. Since all traces of $M^k_{\mathcal{B}}$ terminates in a state s.t. *end* is true and when *end* is true the variables remain unchanged ($(end' \rightarrow V' = V)$ and the fact that obs_Γ requires $\neg end$). □

The following theorem establishes the correctness of the reduction SC-HyperLTL to HyperLTL with finite semantics (as defined in Section 2.8.1) using Definition 8.5, Definition 8.12 and, Definition 8.13.

Theorem 8.9 *Let M be a STS, let $\varphi := \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\Gamma.\psi$ be a prenex SHyperLTL$^{\Gamma}_{\mathsf{S}+\mathsf{C}}$ formula, where each $\mathcal{Q}_i \in \{\forall, \exists\}$ is a quantifier, let $\theta_\Gamma := \bigvee_{\theta\in\Gamma}(\neg\theta \leftrightarrow \boldsymbol{Y}\theta)$. Consider $M^k_{\mathcal{B}} := M^{pad} \times M_{\theta_\Gamma} \times P^k_{\theta_\Gamma,\overline{V}}$ to be the STS constructed by the synchronous composition of $P^k_{\theta_\Gamma,\overline{V}}$ with M and M_{θ_Γ} extended with end transitions. If $M^k_{\mathcal{B}} \models_f \boldsymbol{G}(pos \leq k)$ holds, then $M \models_f \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\psi$ iff $M^k_{\mathcal{B}} \models_f \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\boldsymbol{G}(\boldsymbol{N}\bot \rightarrow {}^{\bot}[\![\psi]\!]^k_0)$. Moreover,*

- *If $M^k_{\mathcal{B}} \models_f \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\boldsymbol{G}(\boldsymbol{N}\bot \rightarrow {}^{\bot}[\![\psi]\!]^k_0)$ then $M \models_f \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\psi$.*
- *If $M^k_{\mathcal{B}} \not\models_f \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\boldsymbol{G}(\boldsymbol{N}\bot \rightarrow {}^{\top}[\![\psi]\!]^k_0)$ then $M \not\models_f \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\psi$*

Proof: (First part): Since $M^k_{\mathcal{B}} \models_f \mathbf{G}(pos \leq k)$ each trace of $M^k_{\mathcal{B}}$ has at most k observations.

We can prove the first part of the theorem by induction with quantifiers and Γ. The statement is the following: Let $\tilde{\Pi}$ be the partial trace assignment of $x_n, \ldots, x_{i+1}$ over M and Π be the partial trace assignment of $x_n, \ldots, x_i + 1$ over $M^k_{\mathcal{B}}$ s.t. each trace of $\tilde{\Pi}$ correspond with the extended trace with end of $M^k_{\mathcal{B}}$. Then, $\Pi \models_f \mathcal{Q}_i x_i \ldots \mathcal{Q}_1 x_1.\psi^i \Leftrightarrow \Pi' \models_f \mathcal{Q}_i x_i \ldots \mathcal{Q}_1 x_1.\psi$.

Base case(Γ): $\tilde{\Pi} \models_{fT} \Gamma.\varphi \Leftrightarrow (\tilde{\Pi}, \Gamma) \models_f \varphi$. By Lemma 8.3 we obtain $\tilde{\Pi} \models_{fT} \Gamma.\varphi \Leftrightarrow \Pi \models_f \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\rho}[\![\varphi]\!]^k_0$.

Inductive case: $\tilde{\Pi} \models_{fT} \forall x_i.\alpha$ if and only if for all $\tilde{\sigma}_i \in \mathcal{L}_{fin}(M)\tilde{\Pi}[x_i \mapsto \tilde{\sigma}_i] \models_{fT} \alpha$. By induction, we obtain $\Pi[x_i \mapsto \sigma_i \models_{fT} \mathcal{Q}_{-1} \ldots .\mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\rho}[\![\varphi]\!]^k_0)$. Since each trace $\tilde{\sigma} \in \mathcal{L}_{fin}(M)$ as a corresponding trace $\sigma \in \mathcal{L}_{fin}(M^k_{\mathcal{B}})$ and vice-versa, we obtain:

for all $\tilde{\sigma}_i \in \mathcal{L}_{fin}(M)\tilde{\Pi}[x_i \mapsto \tilde{\sigma}_i] \models_{fT} x_{i-1} \ldots . \Gamma.\varphi \Leftrightarrow$ for all $\sigma_i \in \mathcal{L}_{fin}(M^k_{\mathcal{B}})$ $\Pi[x_i \mapsto \sigma_i] \models_{fT} x_{i-1} \ldots . \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\rho}[\![\varphi]\!]^k_0) \Leftrightarrow \Pi \models_f \forall x_i.\mathcal{Q}_{i-1} \ldots . \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\rho}[\![\varphi]\!]^k_0)$. The proof for the existential case is identical.

We now prove the second part of Theorem 8.9:

(Second part): Let $\alpha := \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\Gamma.\varphi$ and let ${}_{\rho}\alpha^k_B := \mathcal{Q}_n x_n \ldots \mathcal{Q}_1 x_1.\mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\rho}[\![\varphi]\!]^k_0)$.

Suppose by w.o.c. that $M^k_{\mathcal{B}} \models_f {}^{-}\alpha^k_B$ and $M \not\models_f \alpha$.

We consider for simplicity trace assignments Π (for $M^k_{\mathcal{B}}$) and $\tilde{\Pi}$ (for M).

Since $(\tilde{\Pi}, \Gamma)$ violates φ, then there must be more than k' observations in $\tilde{\Pi}$ and Π s.t. $k' > k$ (note that k' is finite but unbounded).

Let us now considered the trace assignment Π' obtained from $\tilde{\Pi}$ to be part of $M^{k'}_{\mathcal{B}}$ (pos is determined deterministically while v_i are non-deterministically chosen until $pos = i \wedge obs_{\Gamma}$ is true).

We derive that $\Pi' \not\models_f \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\bot}[\![\varphi]\!]^{k'}_0)$. By the unrolling structure of ${}^{\rho}[\![\varphi]\!]^k_0$, we see that $\Pi' \models_f \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\bot}[\![\varphi]\!]^k_0) \Rightarrow \Pi' \models_f \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\bot}[\![\varphi]\!]^{k'}_0)$. This is true because the unrolling at k is false with ${}^{\bot}[\![\varphi]\!]^k_0$ and negation occur only in the leafs (φ is in negation normal form).

Finally, $\Pi' \not\models_f \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\bot}[\![\varphi]\!]^k_0)$ implies that $\Pi \not\models_f \mathbf{G}(\mathbf{N}\bot \rightarrow {}^{\bot}[\![\varphi]\!]^k_0)$. This is true because (i) both traces have the same assignments to V, the prophecies variables up to k, and pos (until it reaches $k+1$); (ii) the property does not depend on $v_{i+1}, \ldots v_k$ (for each $v \in \overline{V}$) or pos when it is assigned to values greater than $k+1$ (we just check it to be $> k$). Therefore, we encounter a contradiction.

The proof of the other direction is very similar. □

The theorem establishes a correspondence between the model-checking problem of SC-HyperLTL formula in the original STS M and in the synchronous composition of M^{pad}, M_{θ_Γ} and $P^k_{\theta_\Gamma,\overline{V}}$. The extended system $M^k_{\mathcal{B}}$ incorporates the evaluation of Γ points using M_{θ_Γ}. The extended system considers traces with possibly different lengths via M^{pad}, and incorporates $P^k_{\theta_\Gamma,\overline{V}}$ to keep track of the assignments of variables in the observational point. The first part of the theorem states that, if M has at most k observations, then the unrolling ${}^{\rho}[\![\psi]\!]^k_0$ is satisfied at the end of the traces (i.e. when $\mathbf{N}\bot$ is true) if and only if the original model-checking problem holds. The second part of the theorem considers a partial relation between the model checking results of both problems. If the unrolling provides a positive result then the original formula satisfies the property as well. If proving the property requires more than k steps, the rule ${}^{\rho}[\![\psi]\!]^k_k$ returns $\bot$ as result (with $\rho = \bot$). Conversely, with an "optimistic" view by setting

$\rho = \top$, a negative result (i.e. the property is not satisfied) entails that M violates the original formula. This mimics the pessimistic (resp. optimistic) semantics [199] for synchronous BMC with the difference that in our algorithm the actual traces have unbounded length.

Beyond k-observations.

It is possible to extend Algorithm 2 relaxing the global limit of k-observations into k-difference-in-observations, where observations are stored and evaluated modulo k. In Algorithm 2 the i-th observation is stored at point i (assuming $i < k$) and remains fixed for the rest of the trace. In the extended approach, the storage of observations would cycle (indexed by $i \mod k$), allowing properties to be checked dynamically over the trace. To support this, an additional constraint needs to keep track of which trace is the fastest, enabling evaluation relative to trace alignment. This modified algorithm can handle systems where the differences in observations is bounded by k.

8.4.3 Proof of Concept Implementation

We implemented the techniques described in this section in a proof-of-concept implementation using the nuXmv model checker [87]. The implementation applies the translations described above and outputs either a self-composed SMV model (if the property has no quantifier alternation) or an AutoHyper model (see [38]) with its corresponding HyperLTL property.

To validate our approach, we use three examples: (1) the *reactive monotonic* example presented at the beginning of Chapter 8, (2) a variation of the battery sensor from Section 8.3 and (3) the parallel composition, under interleaving semantics, of two programs: *P0*, that can take an arbitrary number of steps to terminate, and *P1*, which terminates after k steps. A scheduler that decides non-deterministically which process runs: *P0* (if $sched = 0$) or *P1* (if $sched = 1$). We intend to prove that the execution of *P1* is *deterministic* no matter what *P0* does, encoded as follows:

$$\psi_{det} := \forall x.\forall y.\{P1.var\}.\langle\{x\}\rangle FSchd[x] \wedge \langle\{y\}\rangle FSchd[y] \rightarrow \mathbf{G}(P1.var[x] \leftrightarrow P1.var[y])$$

where $FSchd := \mathbf{G}(\mathbf{NN}\bot \rightarrow sched = 1)$ states that the last scheduled process is *P1*. The property states that, if in the last transition of both traces *P1* is scheduled, then the two programs have always the same value of variable *P1.var*. We analyzed variations with quantifier alternations, incorrect versions (with manual bugs), and versions with

Table 8.1: Empirical evaluation, where $*$ means non-informative with k-bound due to too many observations (TO is 1h). Dashed line on tools mean that they cannot be solved with it. For instance the quantifier alternation model cannot be checked with nuXmv and models with past (Battery Sensor) cannot be checked with AutoHyper.

Name	**Alg.**	**K**	**Res.**	**nuXmv Time (s)**	**AutoHyper Time (s)**
Proc. $n = 2$	k-bound	3	$\top$	2.82	5.54
Proc. $n = 4$	k-bound	5	$\top$	4.03	TO
Proc. $n = 4$	Stuttering	–	$\top$	1.86	MO
Proc. $n = 6$	k-bound	7	$\top$	4.17	MO
Proc. $n = 6$	Stuttering	–	$\top$	2.65	MO
Proc. $n = 6$ (bug)	k-bound	7	$\bot$	3.33	MO
Proc. $n = 6$ (bug)	Stuttering	–	$\bot$	3.02	MO
Proc. q. alt. $n = 2$	k-bound	3	$\top$	–	34.93
Proc. q. alt. $n = 3$	k-bound	4	$\top$	–	TO
Motivating example	k-bound	7	$\top$	1245.95	TO
Motivating example	k-bound	3	$\bot^*$	34.64	TO
Motivating example	Stuttering	–	$\top$	20.88	6.19
Battery Sensor	k-bound	7	$\bot^*$	11.86	–
Battery Sensor	Stuttering	–	$\top$	6.03	–
Battery Sensor (bug)	Stuttering	–	$\top$	3.04	–

unbounded amount of observations and controlled scheduling for bounded observations. The experiments[4] were run on a cluster with Intel Xeon CPU 6226R nodes running at 2.9GHz with 32CPU, 12GB. The timeout for each run was one hour and the memory cap was set to 1GB with 1 CPU assigned for each instance. Table 8.1 reports the execution times of the approaches. We observe that nuXmv is able to solve quite rapidly all instances, but AutoHyper suffers a significant performance slowdown. One possible reason is that encoding the behaviour as part of the formula has a significant impact on explicit state model-checkers like AutoHyper. Another possibility is the additional variables introduced to deal with traces of different lengths. We leave the research of how to alleviate this problem as future work. More in general, we observe that Algorithm 1, that applies stuttering rewriting (for quantifier-alternation free instances) is more successful (when applicable).

[4] All the data can be found at `https://es-static.fbk.eu/people/bombardelli/papers/spin25/spin25.tar.gz`

8.5 Related Works

In Chapter 3, we studied the landscape of asynchronous hyperproperties (see Section 3.4. In [37], the asynchronous hyperlogic Observation HyperLTL is introduced—a decidable logic that shares several similarities with the logics defined in this work. However, there are key differences between our formalisms and Observation HyperLTL: (*i*) SC-HyperLTL and $\text{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ enable local reasoning on traces through the use of singleton contexts, (*ii*) Observation HyperLTL allows each trace to have distinct observation conditions, whereas our framework defines a uniform observation model by selecting a common set of LTL formulae applied identically across all traces, and (*iii*) $\text{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ supports non-prenex quantification, in contrast to the standard prenex form required by Observation HyperLTL.

Similarly, [27] presents *Asynchronous HyperLTL* (A-HLTL), which extends HyperLTL with the notion of trajectories, introduced in [64], to control the progress of the traces. Our first algorithm is similar to the stutter algorithm from [27], but our fragment is strictly richer, allowing synchronous bounded operators as well and directly supporting past operators.

Two other extensions of HyperLTL [72] are stuttering HyperLTL (HyperLTL_S), which describes with LTL formulae the interesting points of each trace, and context HyperLTL (HyperLTL_C), which restricts the progress of specific subsets of traces. This work is a direct extension of [72].

All of these logics are defined for infinite traces and most of them do not support past operators. Two exception for finite traces are [182] and [22]. For what regards [182], the logic introduced there is synchronous, lacks past operators, and provides no tool. On the other hand, in [22] the formalism considers both finite traces and asynchronous systems. However, the formalism is conceptually different and based on automata rather than temporal logics.

Several tools have been proposed for verifying synchronous hyperproperties: MCHyper [162] (which reduces alternating-free verification to LTL model checking), AutoHyper [38] (which implements explicit-state automata constructions), and HyperQB [199] (which follows a bounded-model approach by generating queries for a QBF solver). Recently, new works have focused on verifying asynchronous hyperlogics [198, 37, 27, 34, 129]. In [198], the BMC-based verification of HyperLTL has been adapted to handle A-HLTL. While similar in spirit to our second algorithm, the key difference lies in our algorithm's ability to reason over unbounded executions. Another notable difference between the two approaches is in their reduction techniques. Our method reduces the

problem to a synchronous HyperLTL model-checking problem, whereas [198] reduces it to a QBF verification problem. The remaining works employ very different techniques. In [37], the authors reduce Observation HyperLTL model checking to a game-based verification using predicate abstraction. In [129], symbolic execution is used to efficiently find counterexamples to asynchronous $\forall\exists$ hyperproperties. Finally, in [34], Hoare logic is used to reason over $\forall\exists$ temporal safety properties in asynchronous programs.

In [71], the authors compare the expressiveness of HyperCTL* with KCTL* and provide a logic subsuming both. Our work, extends that study in the asynchronous context.

Part V

Conclusions

Chapter 9

Conclusions and Future Work

In this thesis, we discussed various challenges and technical aspects of model checking across a heterogeneous set of applications. We presented modest contributions in different areas of the model checking domain, taking into account key challenges such as scalability, expressiveness, and the structural complexity of systems—particularly in the presence of asynchrony. While each contribution addressed distinct challenges–ranging from verification modulo theory and compositional reasoning to the verification of hyperproperties–they collectively advance the broader goal of making formal verification more tractable, expressive, and applicable to real-world systems.

Part II: Chapter 4 presented a framework for the verification of LTL modulo theory that reduces to respectively safety and liveness checking modulo theory. We provided direct relation between various finite semantics and the safety fragment of LTL, showing a common reduction to $\text{LTL}_f(\mathcal{T})$ model checking. As a second contribution, we exploited the notion of *relative safety* [194] to verify larger fragments of LTL modulo theory directly using safety properties techniques. First, we defined a set of conditions for which it is possible to verify a fragment of $\text{LTL}(\mathcal{T})$ exploiting relative safety and relative liveness with specific assumptions on the system. Second, when that is not directly possible, we extended our approach with a counterexample guided approach in which, when a spurious counterexample is encountered, we refine the system blocking it. Although the spurious check query is still a liveness query, it is performed to a very simple specification, and thus, it is usually very quick. To reduce the amount of the steps needed by the algorithm to converge, we considered an optimization that blocks counterexamples that cannot be extended for more that k steps. This approach was compared with state-of-the-art liveness checkers; it was proved that in a variety of

structures of benchmarks our approach was significantly more scalable than standard algorithms.

Part III: Chapter 5 present a framework for compositional verification of asynchronous systems. The framework is based on hierarchical decomposition of I/O component based system. This compositional approach was achieved with a local rewriting based approach. The rewriting mapped properties meant to be interpreted over local traces over the trace of the composition. The main challenge faced to apply this approach was the presence of input data ports that could change when the local component was not scheduled, de facto breaking known properties of stutter invariance. To fix the overhead of this rewriting in that context, the approach was optimized exploiting the structure of the formula. This technique was integrated in the contract based design tool OCRA [92] and the approach was evaluated with an extensive experimental evaluation.

Chapter 6 extended the asynchronous compositional framework to timed properties considering an additional challenge: clock skew. The chapter presented an extension of the compositional framework to timed transitions systems with possibly skewed clocks. When clocks are non resettable clocks, assuming that each local clock has the same rate, it is possible to locally check local properties using the standard semantics while checking only the composition in a distributed setting. On the other hand, with clock resets, we had to generalize the framework to deal with the non monotonicity of time introduced by skewed clock resets. We introduced the temporal logics MTLSK, which extends MTL dealing with distributed time using local clocks and define variations of the semantics to treat non monotonicity in the traces. We then constructed an encoding from this logic to LTL($\mathcal{T}$) to enable the use of it in a compositional framework.

Part IV: Chapter 7 introduces a novel model checking algorithms for $\forall\exists$ temporal safety HyperLTL formulae. The approach was build adapting the known concept of inductive invariant used in the context of trace properties. Here the setting is slightly different and thus, the proof obligations are adapted with quantification and discarded either by QBF solvers or by SMT solvers with theory capabilities e.g., Z3 [139]. The challenge in this quantified setting is that – in order to be inductive – the inductive proof obligation still need to universally quantify over the existential part of the path, resulting in practice with the impossibility to check directly with induction hyperproperties. To address this issue, we employed two approaches: one included generalized this flavour of induction considering k-inductive proof obligations, the other is to define

a generalization approach removing counterexamples to inductions. As for the original k-induction line of research, we introduced an incremental encoding of the proof obligations. However, the encoding remains only partially incremental, as the quantified structure of the obligations prevents full incrementality. Finally, the chapter presents a comparison with known tools employing different techniques for symbolic model checking where it is shown that our algorithm is competitive with the other approaches.

Chapter 8 presents contributes in the context of hyperproperties from a slightly different perspective. In presents our study in the context of asynchronous hyperproperties to provide expressible and yet tractable formalisms. The Chapter present a unified asynchronous hyperlogics GHyperLTL$_{\mathsf{S+C}}$ constructed combining Stuttering HyperLTL with Context HyperLTL allowing non prenex quantification. The resulting logic is not surprisingly undecidable; nevertheless, we found a very expressive and yet decidable fragment keeping significant aspects of both HyperLTL$_{\mathsf{S}}$,HyperLTL$_{\mathsf{C}}$ and still allowing some forms of inner quantification. One key aspect to guarantee decidability To prove decidability, we present a polynomial reduction from SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ to QPTL– a known extension of LTL that permits propositional quantification. On this unified framework, we also studied the expressiveness of the logics comparing it with HyperLTL$_{\mathsf{S}}$ and KLTL with a single agent under observation semantics. We have shown that SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ subsume both formalisms while still being decidable. In the second part of the Chapter, we studied a prenex fragment of SHyperLTL$^{\Gamma}_{\mathsf{S+C}}$ over finite traces that supported reasoning over trace with different lengths. Exploiting the asynchronous composition framework presented in Chapter 5[1] to deal with quantifier-alternation free properties. Moreover, we presented an alternative approach for properties with quantifiers alternation but with a bounded amount of observation. This second approach was designed combining the use of monitoring variables and BMC like encoding of formulae. This approach still lacks of scalability when quantifier-alternation properties are considered.

9.1 Future Work

While this thesis addresses several core challenges in model checking, it also opens up a number of promising directions for future research. The techniques and results presented here lay the groundwork for further exploration, both in extending theoretical frameworks and in improving the scalability and applicability of verification tools in

[1]We did not use the same tool system, we exploited variations of the very same technique.

practical settings.

Part II. The first part of this section focuses on building a modular and transparent framework for symbolic model checking of Linear Temporal Logic modulo theories (LTL($\mathcal{T}$)) and its variants over both finite and infinite traces. While this material primarily reorganizes known techniques [99] in a uniform and accessible way, it provides a solid base for further extensions. A natural continuation would be to extend the framework to support timed systems integrating known constructions such as [159] or [99],

In the second part of the chapter, we explored the concept of *relative safety*, where certain non-safety properties can be reduced to invariant checking under specific assumptions. One possible direction is to expand the class of supported formulae by leveraging the notion of relative safety more effectively. Currently, we focus on formulae of the form $\alpha \rightarrow \varphi$, where φ is a SafetyLTL($\mathcal{T}$) formula. A natural extension would be to consider cases where φ is not a safety property. This would require identifying a SafetyLTL($\mathcal{T}$) formula φ_S such that the implication $\alpha \rightarrow \varphi$ is equivalent to $\alpha \rightarrow \varphi_S$, thereby reducing the original verification problem to a purely safety-based one.

On the algorithmic side, future work could focus on refining the generalization strategy within the refinement loop. This could be done by analyzing inductive invariants derived from the k-liveness check proving the spuriousness of the counterexample, or by integrating abstract counterexamples, particularly in the context of infinite-state systems. Other generalization approaches might include interpolation techniques; for instance one might try to look for extendibility using BMC, and, if the counterexample is not extendible employ interpolation to learn a new set of unfeasible states.

Another challenge lies in the automata-theoretic encoding of assumptions: the current construction can introduce deadlocks that complicate the verification process. Exploring alternative automata constructions, such as those in [159, 115], may help mitigate this issue.

Finally, the implication structure of formulae in the relative safety setting makes this framework a natural candidate for compositional verification. Specifically, adapting it for verifying SafetyLTL($\mathcal{T}$) properties in a compositional setting could bridge the techniques developed here with those discussed in Part III.

Part III. A promising direction for future research is to extend the current asynchronous compositional framework toward contract-based reasoning, similar to the OCRA methodology. While some steps have been made in the context of infinite-state

systems with LTL($\mathcal{T}$), this has not yet been fully explored under truncated semantics. In this setting, it could be interesting to evaluate parts of the assumptions under a stronger semantic interpretation, especially for early violation detection. Another natural extension would involve enhancing the contract-based framework with safety assessment techniques inspired by [68], allowing one to reason about contracts that may fail dynamically during execution.

A potential future direction would involve considering ad-hoc optimizations tailored for specific forms communications between components and ad-hoc scheduling constraints. For instance, if two components are composed asynchronously and scheduled with a very different rate [91](e.g., one component is executed every time unit while the other every 1000 time units), verifying their composition would be very challenging without ad-hoc techniques.

A further avenue involves introducing explicit-state reasoning into the compositional approach. While our work addresses scalability through compositionality, integrating techniques such as ESST [108] may offer more flexibility in handling certain classes of systems.

In the timed setting presented later in this part, additional directions emerge. One is the integration of our compositional approach for Distributed Real-Time Systems, based on skewed clocks, into the OCRA [92] tool. Moreover, evaluating the applicability of these methods in real-world case studies is a crucial next step to demonstrate practical impact. From a theoretical perspective, a natural direction for future work is to investigate the decidability and complexity bounds of MTLSK. Although the current reduction relies on fragments of LTL($\mathcal{LRA}$) that are known to be undecidable, we have not yet established whether MTLSK itself is decidable or undecidable. Identifying a suitable decidable fragment remains an open and interesting challenge. One possible approach is to explore a reduction to Rectangular Linear Hybrid Automata, which may preserve decidability under certain restrictions. Such a reduction would likely require limiting the expressiveness of the logic—for instance, by constraining how clock rates vary or by restricting the conditions under which resets may occur. Finally, an intriguing research direction is the adaptation of relative safety techniques to Metric Temporal Logics, and in particular, to the MTLSK logic introduced in this thesis, potentially enabling more scalable analysis of expressive real-time specifications e.g. leveraging MTL to timed automata construction [159].

Part IV. We now turn to possible future directions in the context of hyperproperties, starting with our work on Hyperk-induction. A first natural extension is to explore

abstraction-based techniques to enhance scalability and precision. In particular, integrating predicate abstraction as explored in [37], or abstraction refinements like those in [187], may improve the efficiency and applicability of the approach.

Furthermore, recent advances in solving constrained Horn clauses (CHC) with alternating quantifiers, particularly for $\forall\exists$ fragments [284, 204], could offer new insights. Studying these techniques may both inspire new optimizations in our current method and highlight potential limitations or strengths.

A more ambitious direction would be to explore whether variants of algorithms such as IC3 [75] or IMC [246], originally developed for trace properties, can be adapted to hyperproperties. While this direction poses significant technical challenges, such an extension might enhance practical model checking capabilities. On a similar perspective, it would be nice to develop algorithms to deal with temporal liveness i.e. properties of the form $\forall^*\exists^*.\mathbf{FG}\phi$ and to define a general framework for $\forall^*\exists^*.\phi$ properties as done for trace properties in Chapter 4.

On the solver side, there are two concrete paths: integrating support for bounded Skolem functions (as some modern QBF solvers provide), and incorporating incremental QBF solving [232] to improve the efficiency of the iterative checks performed by our algorithms. A crucial but still underdeveloped direction is to study how to generalize hyperkinduction and its supporting framework to verification in asynchronous settings, where interleaving and temporal alignment pose additional complexity.

In the second chapter of Part IV, we introduced $\text{SHyperLTL}^{\Gamma}_{\mathsf{S}+\mathsf{C}}$, a general and expressive asynchronous hyperlogic. Going forward, a key goal is to develop practical verification algorithms for fragments of this logic. We conjecture that for specific fragments—especially those without quantifier alternation between traces –algorithmic techniques similar to the ribbon-shaped encodings used in [70] may be adapted to work efficiently.

Additionally, we plan to explore whether the stuttering-based semantics defined for finite traces can be reduced to known problems such as LTL_f model checking. Such a reduction could enable leveraging mature symbolic techniques for more scalable verification. Another important comparative step would be a systematic evaluation and integration study with the logic and tools developed for A-HLTL [27].

From the bounded-observation algorithm perspective, we aim to optimize the current encoding of the formulae. One direction is to relax the synchronization constraints on observations: instead of requiring fully aligned observations, we may consider traces where observations are only a bounded distance apart. This relaxation can significantly increase the range of interactions captured by the algorithm, even when using a rela-

tively small bound k. Another direction would be to try to simplify the overall encoding reducing the size of the generated formula.

Bibliography

[1] Ieee standard for property specification language (psl) - redline. *IEEE Std 1850-2010 (Revision of IEEE Std 1850-2005) - Redline*, pages 1–188, 2010.

[2] Ieee standard for systemverilog–unified hardware design, specification, and verification language. *IEEE Std 1800-2023 (Revision of IEEE Std 1800-2017)*, pages 1–1354, 2024.

[3] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Trans. Program. Lang. Syst.*, 17(3):507–535, May 1995.

[4] Karam Abd Elkader, Orna Grumberg, Corina S. Păsăreanu, and Sharon Shoham. Automated circular assume-guarantee reasoning. *Form. Asp. Comput.*, 30(5):571–595, September 2018.

[5] Erika Ábrahám and Borzoo Bonakdarpour. Hyperpctl: A temporal logic for probabilistic hyperproperties. In Annabelle McIver and Andras Horvath, editors, *Quantitative Evaluation of Systems*, pages 20–35, Cham, 2018. Springer International Publishing.

[6] R. Alur and T.A. Henzinger. Logics and Models of Real Time: A Survey. In *REX Workshop*, pages 74–106, 1991.

[7] R. Alur and T.A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.

[8] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(1):253–273, 1999.

[9] Rajeev Alur and Thomas A. Henzinger. A Really Temporal Logic. *J. ACM*, 41(1):181–204, 1994.

[10] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[11] Alessandro Artale, Luca Geatti, Nicola Gigante, Andrea Mazzullo, and Angelo Montanari. Complexity of safety and cosafety fragments of linear temporal logic. In *AAAI*, pages 6236–6244. AAAI Press, 2023.

[12] Thomas Arts, Michele Dorigatti, and Stefano Tonetta. Making implicit safety requirements explicit. In Andrea Bondavalli and Felicita Di Giandomenico, editors, *Computer Safety, Reliability, and Security*, pages 81–92, Cham, 2014. Springer International Publishing.

[13] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*, volume 26202649. 01 2008.

[14] Musard Balliu, Mads Dam, and Gurvan Le Guernic. Epistemic temporal logic for information flow security. In *ACM Workshop on Programming Languages and Analysis for Security*, 2011.

[15] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength smt solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442, Cham, 2022. Springer International Publishing.

[16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, 2021. `https://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2021-05-12.pdf`.

[17] Clark Barrett and Cesare Tinelli. *Satisfiability Modulo Theories*, pages 305–343. Springer International Publishing, Cham, 2018.

[18] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1267–1329. IOS Press, 2021.

[19] Damian Barsotti, Leonor Prensa Nieto, and Alwen Tiu. Verification of clock synchronization algorithms: Experiments on a combination of deductive tools.

Electronic Notes in Theoretical Computer Science, 145:63–78, 2006. Proceedings of the 5th International Workshop on Automated Verification of Critical Systems (AVoCS 2005).

[20] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. *Mathematical Structures in Computer Science*, 21:1207–1252, 2011.

[21] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. *Introduction to Runtime Verification*, pages 1–33. 02 2018.

[22] Ezio Bartocci, Thomas A. Henzinger, Dejan Nickovic, and Ana Oliveira da Costa. Hypernode Automata. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory (CONCUR 2023)*, volume 279 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[23] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*, pages 3–12, 2006.

[24] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20(4), sep 2011.

[25] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011.

[26] Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A Temporal Logic for Asynchronous Hyperproperties. In *CAV (1)*, volume 12759 of *Lecture Notes in Computer Science*, pages 694–717. Springer, 2021.

[27] Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A temporal logic for asynchronous hyperproperties. In *Proc. of the 33rd Int'l Conf. on Computer Aided Verification (CAV'21), Part I*, volume 12759 of *LNCS*, pages 694–717. Springer, 2021.

[28] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*, pages 125–126, 2006.

[29] Nikola Benes, Lubos Brim, Ivana Cerná, Jirí Sochor, Pavlína Vareková, and Barbora Buhnova. Partial order reduction for state/event ltl. In *IFM*, 2009.

[30] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. D-finder: A tool for compositional deadlock detection and verification. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 614–619, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[31] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *FMCO*, 2007.

[32] Albert Benveniste, Benoît Caillaud, and Roberto Passerone. A generic model of contracts for embedded systems. *ArXiv*, abs/0706.1456, 2007.

[33] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Vincentelli, Werner Damm, Thomas Henzinger, and Kim Larsen. Contracts for system design. *Foundations and Trends® in Electronic Design Automation*, 12, 11 2012.

[34] Raven Beutner. Automated software verification of hyperliveness. In *Tools and Algorithms for the Construction and Analysis of Systems: 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6–11, 2024, Proceedings, Part II*, page 196–216, Berlin, Heidelberg, 2024. Springer-Verlag.

[35] Raven Beutner and Bernd Finkbeiner. A Temporal Logic for Strategic Hyperproperties. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[36] Raven Beutner and Bernd Finkbeiner. Prophecy variables for hyperproperty verification. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*, pages 471–485, 2022.

[37] Raven Beutner and Bernd Finkbeiner. *Software Verification of Hyperproperties Beyond k-Safety*, page 341–362. Springer International Publishing, 2022.

[38] Raven Beutner and Bernd Finkbeiner. Autohyper: Explicit-state model checking for hyperltl. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 145–163, Cham, 2023. Springer Nature Switzerland.

[39] Raven Beutner and Bernd Finkbeiner. Hyperatl*: A logic for hyperproperties in multi-agent systems. *Logical Methods in Computer Science*, Volume 19, Issue 2, May 2023.

[40] Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Second-Order Hyperproperties. In *Proc. 35th CAV*, volume 13965 of *Lecture Notes in Computer Science*, pages 309–332. Springer, 2023.

[41] Dirk Beyer. Competition on software verification - (SV-COMP). In *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 504–524. Springer, 2012.

[42] Dirk Beyer and Andreas Podelski. *Software Model Checking: 20 Years and Beyond*, pages 554–582. Springer Nature Switzerland, Cham, 2022.

[43] Armin Biere. Picosat essentials. *JSAT*, 4:75–97, 05 2008.

[44] Armin Biere, Cyrille Artho, and Viktor Schuppan. Liveness checking as safety checking. In *International Workshop on Formal Methods for Industrial Critical Systems*, 2002.

[45] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In W. Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[46] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking, 2003.

[47] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proc. of the 5th Int'l Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[48] Armin Biere, Keijo Heljanko, and Siert Wieringa. AIGER 1.9 and beyond. Technical Report 11/2, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2011.

[49] Benjamin Bittner, Marco Bozzano, Alessandro Cimatti, Marco Gario, Stefano Tonetta, and Viktoria Vozárová. Diagnosability of fair transition systems. *Artif. Intell.*, 309:103725, 2022.

[50] Nikolaj Bjørner, Arie Gurfinkel, Ken McMillan, and Andrey Rybalchenko. *Horn Clause Solvers for Program Verification*, pages 24–51. Springer International Publishing, Cham, 2015.

[51] Simon Bliudze, Alessandro Cimatti, Mohamad Jaber, Sergio Mover, Marco Roveri, Wajeb Saab, and Qiang Wang. Formal Verification of Infinite-State BIP Models. In *ATVA*, volume 9364 of *LNCS*, pages 326–343. Springer, 2015.

[52] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan, and Richard Seeber. RATSY - A new requirements analysis tool with synthesis. In *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 425–429. Springer, 2010.

[53] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[54] Alberto Bombardelli, Marco Bozzano, Roberto Cavada, Alessandro Cimatti, Alberto Griggio, Massimo Nazaria, Edoardo Nicolodi, and Stefano Tonetta. Compasta: Extending taste with formal design and verification functionality. In *Model-Based Safety and Assessment: 8th International Symposium, IMBSA 2022, Munich, Germany, September 5–7, 2022, Proceedings*, page 21–27, Berlin, Heidelberg, 2022. Springer-Verlag.

[55] Alberto Bombardelli, Laura Bozzelli, César Sánchez, and Stefano Tonetta. Unifying asynchronous logics for hyperproperties. *CoRR*, abs/2404.16778, 2024.

[56] Alberto Bombardelli, Laura Bozzelli, César Sánchez, and Stefano Tonetta. (Asynchronous) temporal logics for hyperproperties on finite traces. In *SPIN*, 2025.

[57] Alberto Bombardelli, Alessandro Cimatti, Alberto Griggio, and Stefano Tonetta. *Another Look at LTL Modulo Theory over Finite and Infinite Traces*, pages 419–443. Springer Nature Switzerland, Cham, 2025.

[58] Alberto Bombardelli, Alessandro Cimatti, Stefano Tonetta, and Marco Zamboni. Symbolic model checking of relative safety ltl properties. In *IFM 2023: 18th International Conference, IFM 2023, Leiden, The Netherlands, November 13-15, 2023, Proceedings*, pages 302–320, Berlin, Heidelberg, 2023. Springer-Verlag.

[59] Alberto Bombardelli and Stefano Tonetta. Asynchronous composition of local interface LTL properties. In *Proc. of the 14th Int'l NASA Formal Methods Symposium (NFM'22)*, volume 13260 of *LNCS*, pages 508–526, 2022.

[60] Alberto Bombardelli and Stefano Tonetta. Metric temporal logic with resettable skewed clocks. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2023.

[61] Alberto Bombardelli and Stefano Tonetta. Reasoning with metric temporal logic and resettable skewed clocks. In Kristin Yvonne Rozier and Swarat Chaudhuri, editors, *NASA Formal Methods*, pages 174–190, Cham, 2023. Springer Nature Switzerland.

[62] Alberto Bombardelli and Stefano Tonetta. Asynchronous composition of ltl properties over infinite and finite traces (under review lmcs25), 2025.

[63] B. Bonakdarpour, C. Sánchez, and G. Schneider. Monitoring hyperproperties by combining static analysis and runtime verification. In *Proc. of the 8th Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'18)*, volume 11245 of *LNCS*, pages 8–27. Springer-Cham, 2018.

[64] Borzoo Bonakdarpour, Pavithra Prabhakar, and César Sánchez. Model checking timed hyperproperties in discrete-time systems. In *Proc. of the 12th NASA Formal Methods Symposium (NFM'2020)*, volume 12229 of *LNCS*, pages 311–328. Springer, 2020.

[65] Marco Bozzano, Alessandro Cimatti, Marco Gario, and Stefano Tonetta. Formal Design of Asynchronous Fault Detection and Identification Components using Temporal Epistemic Logic. *Log. Methods Comput. Sci.*, 11(4), 2015.

[66] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. The compass approach: Correctness, modelling and performability of aerospace systems. In Bettina Buth, Gerd Rabe, and Till Seyfarth, editors, *Computer Safety, Reliability, and Security*, pages 173–186, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[67] Marco Bozzano, Alessandro Cimatti, Cristian Mattarei, and Stefano Tonetta. Formal safety assessment via contract-based design. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 81–97, Cham, 2014. Springer International Publishing.

[68] Marco Bozzano, Alessandro Cimatti, Cristian Mattarei, and Stefano Tonetta. Formal safety assessment via contract-based design. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 81–97, Cham, 2014. Springer International Publishing.

[69] Marco Bozzano, Alessandro Cimatti, Anthony Fernandes Pires, David Jones, Greg Kimberly, T. Petri, R. Robinson, and Stefano Tonetta. Formal design and safety analysis of AIR6110 wheel brake system. In *CAV (1)*, volume 9206 of *Lecture Notes in Computer Science*, pages 518–535. Springer, 2015.

[70] Marco Bozzano, Alessandro Cimatti, Stefano Tonetta, and Viktoria Vozarova. Searching for ribbon-shaped paths in fair transition systems. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 543–560, Cham, 2022. Springer International Publishing.

[71] Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In *Proc. 18th FoSSaCS*, volume 9034 of *LNCS*, pages 167–182. Springer, 2015.

[72] Laura Bozzelli, Adriano Peron, and César Sánchez. Asynchronous extensions of HyperLTL. In *Proc. of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'21)*, pages 1–13. IEEE, 2021.

[73] Laura Bozzelli, Adriano Peron, and Cesar Sanchez. Expressiveness and Decidability of Temporal Logics for Asynchronous Hyperproperties, 2022.

[74] Laura Bozzelli, Adriano Peron, and César Sánchez. Expressiveness and Decidability of Temporal Logics for Asynchronous Hyperproperties. In *Proc. 33rd CONCUR*, volume 243 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[75] Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 70–87, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[76] Aaron R. Bradley, Fabio Somenzi, Zyad Hassan, and Yan Zhang. An incremental approach to model checking progress properties. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 144–153, 2011.

[77] Martin Brain, Saurabh Joshi, Daniel Kroening, and Peter Schrammel. Safety Verification and Refutation by k-Invariants and k-Induction. In *SAS*, volume 9291 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2015.

[78] Robert Brayton and Alan Mishchenko. Abc: an academic industrial-strength verification tool. In *Proceedings of the 22nd International Conference on Computer Aided Verification*, CAV'10, page 24–40, Berlin, Heidelberg, 2010. Springer-Verlag.

[79] Geoffrey Brown and Lee Pike. Easy Parameterized Verification of Biphase Mark and 8N1 Protocols. volume 3920, pages 58–72, 03 2006.

[80] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, aug 1986.

[81] Lei Bu, Alessandro Cimatti, Xuandong Li, Sergio Mover, and Stefano Tonetta. Model checking of hybrid systems using shallow synchronization. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, pages 155–169, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[82] Denis Bueno and Karem A. Sakallah. euforia: Complete software model checking with uninterpreted functions. In Constantin Enea and Ruzica Piskac, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 363–385, Cham, 2019. Springer International Publishing.

[83] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10ˆ20 states and beyond. In *LICS*, pages 428–439. IEEE Computer Society, 1990.

[84] Alberto Camacho, Eleni Triantafillou, Christian J. Muise, Jorge A. Baier, and Sheila A. McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*, pages 3716–3724. AAAI Press, 2017.

[85] Lin Cao, Shaolong Shu, Feng Lin, Qijun Chen, and Chengju Liu. Weak diagnosability of discrete-event systems. *IEEE Transactions on Control of Network Systems*, 9(1):184–196, 2022.

[86] Claudia Carapelle, Shiguang Feng, Oliver Fernandez Gil, and Karin Quaas. Satisfiability for MTL and TPTL over Non-monotonic Data Words. In *LATA*, volume 8370 of *LNCS*, pages 248–259, 2014.

[87] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. volume 8559, pages 334–342, 07 2014.

[88] Adrien Champion, Alain Mebsout, Christoph Sticksel, and Cesare Tinelli. The kind 2 model checker. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification*, pages 510–517, Cham, 2016. Springer International Publishing.

[89] Edward Y. Chang, Zohar Manna, and Amir Pnueli. Characterization of Temporal Property Classes. In *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 474–486. Springer, 1992.

[90] Alessandro Cimatti, Sara Corfini, Luca Cristoforetti, Marco Di Natale, Alberto Griggio, Stefano Puri, and Stefano Tonetta. A comprehensive framework for the analysis of automotive systems. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, MODELS '22, page 379–389, New York, NY, USA, 2022. Association for Computing Machinery.

[91] Alessandro Cimatti, Luca Cristoforetti, Alberto Griggio, Stefano Tonetta, Sara Corfini, Marco Di Natale, and Florian Barrau. Eva: a tool for the compositional verification of autosar models. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 3–10, Cham, 2023. Springer Nature Switzerland.

[92] Alessandro Cimatti, Michele Dorigatti, and Stefano Tonetta. Ocra: A tool for checking the refinement of temporal contracts. pages 702–705, 11 2013.

[93] Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. Reactive synthesis from extended bounded response ltl specifications. In *2020 Formal Methods in Computer Aided Design (FMCAD)*, pages 83–92, 2020.

[94] Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. Fairness, assumptions, and guarantees for extended bounded response LTL+P synthesis. *Softw. Syst. Model.*, 23(2):427–453, 2024.

[95] Alessandro Cimatti, Alberto Griggio, and Enrico Magnago. Automatic discovery of fair paths in infinite-state transition systems. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis*, pages 32–47, Cham, 2021. Springer International Publishing.

[96] Alessandro Cimatti, Alberto Griggio, and Enrico Magnago. Proving the existence of fair paths in infinite-state systems. In *International Conference on Verification, Model Checking and Abstract Interpretation*, 2021.

[97] Alessandro Cimatti, Alberto Griggio, and Enrico Magnago. Ltl falsification in infinite-state systems. *Information and Computation*, 289:104977, 2022.

[98] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, and Stefano Tonetta. Extending nuxmv with timed transition systems and timed temporal properties. In *International Conference on Computer Aided Verification*, 2019.

[99] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, and Stefano Tonetta. SMT-based satisfiability of first-order LTL with event freezing functions and metric operators. *Inf. Comput.*, 272:104502, 2020.

[100] Alessandro Cimatti, Alberto Griggio, Sergio Mover, Marco Roveri, and Stefano Tonetta. Verification modulo theories. *Formal Methods Syst. Des.*, 60(3):452–481, 2022.

[101] Alessandro Cimatti, Alberto Griggio, Sergio Mover, Marco Roveri, and Stefano Tonetta. Verification modulo theories. *Formal Methods in System Design*, 60:1–30, 09 2023.

[102] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 Modulo Theories via Implicit Predicate Abstraction. In *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2014.

[103] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Verifying LTL Properties of Hybrid Systems with K-Liveness. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 424–440, 2014.

[104] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT Solver. In Nir Piterman and Scott Smolka, editors, *Proceedings of TACAS*, volume 7795 of *LNCS*. Springer, 2013.

[105] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT 5 SMT Solver. 2012.

[106] Alessandro Cimatti, Alberto Griggio, and Stefano Tonetta. The VMT-LIB language and tools. In *SMT*, volume 3185 of *CEUR Workshop Proceedings*, pages 80–89. CEUR-WS.org, 2022.

[107] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. Hydi: A language for symbolic hybrid systems with discrete interaction. pages 275–278, 08 2011.

[108] Alessandro Cimatti, Iman Narasamdya, and Marco Roveri. Software model checking with explicit scheduler and symbolic threads. *Log. Methods Comput. Sci.*, 8, 2012.

[109] Alessandro Cimatti, Marco Roveri, Angelo Susi, and Stefano Tonetta. Validation of requirements for hybrid systems: A formal approach. *ACM Trans. Softw. Eng. Methodol.*, 21(4):22:1–22:34, 2012.

[110] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. Hreltl: A temporal logic for hybrid systems. *Information and Computation*, 245:54–71, 2015.

[111] Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-based Runtime Verification. *Formal Methods Syst. Des.*, 60(2):277–324, 2022.

[112] Alessandro Cimatti and Stefano Tonetta. Contracts-refinement proof system for component-based embedded systems. *Science of Computer Programming*, 97:333–348, 2015. Object-Oriented Programming and Systems (OOPS 2010) Modeling and Analysis of Compositional Software (papers from EUROMICRO SEAA 12).

[113] Alessandro Cimatti and Stefano Tonetta. Contracts-refinement proof system for component-based embedded systems. *Science of Computer Programming*, 97:333–348, 2015.

[114] Koen Claessen, Niklas Eén, and Baruch Sterin. A circuit approach to ltl model checking. *2013 Formal Methods in Computer-Aided Design*, pages 53–60, 2013.

[115] Koen Claessen, Niklas Een, and Baruch Sterin. A circuit approach to ltl model checking. In *2013 Formal Methods in Computer-Aided Design*, pages 53–60, 2013.

[116] Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. *2012 Formal Methods in Computer-Aided Design (FMCAD)*, pages 52–59, 2012.

[117] Edmund M. Clarke, Orna Grumberg, and Kiyoharu Hamaguchi. Another look at ltl model checking. *Formal Methods in System Design*, 10:47–71, 1994.

[118] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, Cham, 2018.

[119] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pages 353–362, 1989.

[120] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Proc. of the 3rd Conference on Principles of Security and Trust (POST 2014)*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

[121] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal Logics for Hyperproperties. In *Proc. 3rd POST*, LNCS 8414, pages 265–284. Springer, 2014.

[122] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[123] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. PĂsĂreanu. Learning assumptions for compositional verification. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[124] Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *Proc. 34th LICS*, pages 1–13. IEEE, 2019.

[125] Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In *Proc. of the 31st Int'l Conf. on Computer Aided Verification (CAV'19)*, volume 11561 of *LNCS*, pages 121–139. Springer, 2019.

[126] Darren Cofer, Andrew Gacek, Steven Miller, Michael Whalen, Brian LaValley, and Lui Sha. Compositional verification of architectural models. pages 126–140, 04 2012.

[127] Tiago Cogumbreiro, Julien Lange, Dennis Rong, and Hannah Zicarelli. *Checking Data-Race Freedom of GPU Kernels, Compositionally*, pages 403–426. 07 2021.

[128] Arthur Correnson and Bernd Finkbeiner. Coinductive proofs for temporal hyperliveness. *Proc. ACM Program. Lang.*, 9(POPL), January 2025.

[129] Arthur Correnson, Tobias Nießen, Bernd Finkbeiner, and Georg Weissenbacher. Finding ∀∃ Hyperbugs using Symbolic Execution. *Proc. ACM Program. Lang.*, 8(OOPSLA2), October 2024.

[130] Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper, and Mihalis Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods Syst. Des.*, 1(2/3):275–288, 1992.

[131] Andrei Damian, Cezara Drăgoi, Alexandru Militaru, and Josef Widder. Communication-closed asynchronous protocols. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 344–363, Cham, 2019. Springer International Publishing.

[132] Werner Damm, Hardi Hungar, Bernhard Josko, Thomas Peikenkamp, and Ingo Stierand. Using contract-based component specifications for virtual integration testing and architecture design. pages 1 – 6, 04 2011.

[133] Jakub Daniel, Alessandro Cimatti, Alberto Griggio, Stefano Tonetta, and Sergio Mover. Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification*, pages 271–291, Cham, 2016. Springer International Publishing.

[134] Thibault Dardinier and Peter Müller. Hyper hoare logic: (dis-)proving program hyperproperties. *Proc. ACM Program. Lang.*, 8(PLDI), June 2024.

[135] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120. ACM, 2001.

[136] Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on ltl on finite traces: insensitivity to infiniteness. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1027–1033. AAAI Press, 2014.

[137] Giuseppe De Giacomo and Moshe Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13. AAAI Press, 2013.

[138] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for ltl and ldl on finite traces. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 1558–1564. AAAI Press, 2015.

[139] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[140] Leonardo de Moura, Sam Owre, Harald Ruess, John Rushby, Natarajan Shankar, Maria Sorea, and Ashish Tiwari. Sal 2. volume 3114, pages 496–500, 07 2004.

[141] Barbara Di Giampaolo, Salvatore La Torre, and Margherita Napoli. Parametric metric interval temporal logic. In Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications*, pages 249–260, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[142] Isil Dillig, Thomas Dillig, Boyang Li, Kenneth L. McMillan, and Mooly Sagiv. Synthesis of circular compositional program proofs via abduction. *Int. J. Softw. Tools Technol. Transf.*, 19(5):535–547, 2017.

[143] R. Dimitrova, B. Finkbeiner, M. Kovács, M.N. Rabe, and H. Seidl. Model Checking Information Flow in Reactive Systems. In *Proc. 13th VMCAI*, LNCS 7148, pages 169–185. Springer, 2012.

[144] Tommaso Dreossi, Alexandre Donzé, and Sanjit Seshia. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, 63, 12 2019.

[145] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From spot 2.0 to spot 2.10: What's new? In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification*, pages 174–187, Cham, 2022. Springer International Publishing.

[146] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 737–744, Cham, 2014. Springer International Publishing.

[147] Matthew Dwyer, George Avrunin, and James Corbett. Patterns in property specifications for finite-state verification. *Proceedings - International Conference on Software Engineering*, 02 1970.

[148] Niklas Een, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 125–134, 2011.

[149] Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. In *BMC@CAV*, 2003.

[150] Cindy Eisner and Dana Fisman. *Temporal Logic Made Practical*. 01 2016.

[151] Cindy Eisner, Dana Fisman, and John Havlicek. The >;? approach for truncated semantics. 10 2022.

[152] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *Proc. of the 15th Int'l Conf. on Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 27–39. Springer, 2003.

[153] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *International Conference on Computer Aided Verification*, 2003.

[154] Cindy Eisner, Dana Fisman, John Havlicek, Anthony McIsaac, and David Van Campenhout. The Definition of a Temporal Clock Operator. In *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 857–870. Springer, 2003.

[155] J. Eker, J.W. Janneck, E.A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[156] Jenna Elwing, Laura Gamboa-Guzman, Jeremy Sorkin, Chiara Travesset, Zili Wang, and Kristin Yvonne Rozier. Mission-time ltl (mltl) formula validation via regular expressions. In Paula Herber and Anton Wijs, editors, *Integrated Formal Methods*, pages 279–301, Cham, 2024. Springer Nature Switzerland.

[157] Ronald Fagin, Yoram Moses, and Moshe Vardi. *Reasoning About Knowledge*. 01 2003.

[158] Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. Temporal logic motion planning for dynamic robots. *Autom.*, 45(2):343–352, 2009.

[159] Thomas Ferrère, Oded Maler, Dejan Ničković, and Amir Pnueli. From real-time logic to timed automata. *J. ACM*, 66(3), May 2019.

[160] Bernd Finkbeiner. Synthesis of reactive systems. In *Dependable Software Systems Engineering*, volume 45 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 72–98. IOS Press, 2016.

[161] Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *International Conference on Concurrency Theory*, 2016.

[162] Bernd Finkbeiner, Markus Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *In Proc. of the 27th Int'l Conf. on Computer Aided Verification (CAV'15)*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.

[163] Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. In *Proc. 34th STACS*, LIPIcs 66, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[164] Valeria Fionda and Gianluigi Greco. The complexity of ltl on finite traces: hard and easy fragments. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 971–977. AAAI Press, 2016.

[165] Valeria Fionda and Gianluigi Greco. The complexity of LTL on finite traces: Hard and easy fragments. In *AAAI*, pages 971–977. AAAI Press, 2016.

[166] Michael J. Fischer and Richard E. Ladner. Propositional Dynamic Logic of Regular Programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.

[167] Michael Fisher and Michael J. Wooldridge. Temporal reasoning in agent-based systems. In *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*, pages 469–495. Elsevier, 2005.

[168] Dana Fisman and Hillel Kugler. *Temporal Reasoning on Incomplete Paths: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part II*, pages 28–52. 11 2018.

[169] Marie Fortin, Louwe B. Kuijer, Patrick Totzke, and Martin Zimmermann. Hyperltl satisfiability is highly undecidable, hyperctl* is even harder. *Logical Methods in Computer Science*, Volume 21, Issue 1, Jan 2025.

[170] Carlo Furia. A compositional world a survey of recent works on compositionality in formal methods. 01 2005.

[171] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '80, page 163–173, New York, NY, USA, 1980. Association for Computing Machinery.

[172] Ritam Ganguly, Yingjie Xue, Aaron Jonckheere, Parker Ljungy, Benjamin Schornsteiny, Borzoo Bonakdarpour, and Maurice Herlihy. Distributed Runtime Verification of Metric Temporal Properties for Cross-Chain Protocols. *CoRR*, abs/2204.09796, 2022.

[173] Marco Gario. A formal foundation of fdi design via temporal epistemic logic. 2016.

[174] Marco Gario, Alessandro Cimatti, Cristian Mattarei, Stefano Tonetta, and Kristin Yvonne Rozier. Model checking at scale: Automated air traffic control design space exploration. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification*, pages 3–22, Cham, 2016. Springer International Publishing.

[175] Marco Gario and Andrea Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop 2015*, 2015.

[176] Luca Geatti, Alessandro Gianola, and Nicola Gigante. Linear temporal logic modulo theories over finite traces. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 2641–2647. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Main Track.

[177] Luca Geatti, Alessandro Gianola, and Nicola Gigante. First-order automata. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(14):14940–14948, Apr. 2025.

[178] Luca Geatti, Nicola Gigante, Angelo Montanari, and Gabriele Venturato. SAT meets tableaux for linear temporal logic satisfiability. *J. Autom. Reason.*, 68(2):6, 2024.

[179] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1995.

[180] Mihaela Gheorghiu Bobaru, Corina S. Păsăreanu, and Dimitra Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, pages 135–148, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[181] Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, and Daniele Zucchelli. Combination Methods for Satisfiability and Model-Checking of Infinite-State Systems. In *CADE*, volume 4603 of *Lecture Notes in Computer Science*, pages 362–378. Springer, 2007.

[182] Giuseppe De Giacomo, Paolo Felli, Marco Montali, and Giuseppe Perelli. HyperLDLf: a logic for checking properties of finite traces process logs. In *Proc. of the 30th Int'l Joint Conf. on Artificial Intelligence (IJCAI'21)*, pages 1859–1865. ijcai.org, 2021.

[183] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. of the 23rd Int'l Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 854–860. IJCAI/AAAI, 2013.

[184] Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, and Johann Schumann. Generation of formal requirements from structured natural language. In Nazim Madhavji, Liliana Pasquale, Alessio Ferrari, and Stefania Gnesi, editors, *Requirements Engineering: Foundation for Software Quality*, pages 19–35, Cham, 2020. Springer International Publishing.

[185] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified boolean formulas. *Frontiers in Artificial Intelligence and Applications*, 185, 01 2009.

[186] J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*, pages 11–11, 1982.

[187] Ohad Goudsmid, Orna Grumberg, and Sarai Sheinvald. Compositional model checking for multi-properties. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 55–80, Cham, 2021. Springer International Publishing.

[188] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Propositional dynamic logic for hyperproperties. In *Proc. 31st CONCUR*, LIPIcs 171, pages 50:1–50:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[189] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 4(POPL), 2021.

[190] Joseph Y. Halpern and Moshe Y. Vardi. The Complexity of Reasoning about Knowledge and Time: Extended Abstract. In *Proc. 18th STOC*, pages 304–315. ACM, 1986.

[191] Moritz Hammer, Alexander Knapp, and Stephan Merz. Truly on-the-fly ltl model checking. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 191–205, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[192] Anton Hampus and Mattias Nyberg. *Formally Verifying Decompositions of Stochastic Specifications*, pages 193–210. 09 2022.

[193] Jesko Hecking-Harbusch and Leander Tentrup. Solving QBF by abstraction. In Andrea Orlandini and Martin Zimmermann, editors, *Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26-28th September 2018*, volume 277 of *EPTCS*, pages 88–102, 2018.

[194] Thomas A. Henzinger. Sooner is safer than later. *Inf. Process. Lett.*, 43:135–141, 1992.

[195] Krystof Hoder and Nikolaj S. Bjørner. Generalized property directed reachability. In *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2012.

[196] Gerard J. Holzmann, Doron A. Peled, and Mihalis Yannakakis. On nested depth first search. In *The Spin Verification System*, volume 32 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 23–31. DIMACS/AMS, 1996.

[197] Hyoung Seok Hong, Insup Lee, Oleg Sokolsky, and Hasan Ural. A temporal logic based theory of test coverage and generation. In *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 327–341. Springer, 2002.

[198] Tzu-Han Hsu, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. Bounded model checking for asynchronous hyperproperties. In *Tools and Algorithms for the Construction and Analysis of Systems: 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22–27, 2023, Proceedings, Part I*, page 29–46, Berlin, Heidelberg, 2023. Springer-Verlag.

[199] Tzu-Han Hsu, César Sánchez, and Borzoo Bonakdarpour. Bounded model checking for hyperproperties. In *Proc. of the 27th Int'l Conf on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'21). Part I*, volume 12651 of *LNCS*, pages 94–112. Springer, 2021.

[200] Tzu-Han Hsu, César Sánchez, Sarai Sheinvald, and Borzoo Bonakdarpour. Efficient loop conditions for bounded model checking hyperproperties, 2023.

[201] Inigo Incer, Apurva Badithela, Josefine B. Graebener, Piergiuseppe Mallozzi, Ayush Pandey, Nicolas Rouquette, Sheng-Jung Yu, Albert Benveniste, Benoit Caillaud, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. Pacti: Assume-guarantee contracts for efficient compositional analysis and design. *ACM Trans. Cyber-Phys. Syst.*, 9(1), January 2025.

[202] Inigo Incer, Albert Benveniste, Alberto Vincentelli, and Sanjit Seshia. *Hypercontracts*, pages 674–692. 05 2022.

[203] Daisuke Ishii. A hypergraph-based formalization of hierarchical reactive modules and a compositional verification method, 2024.

[204] Shachar Itzhaky, Sharon Shoham, and Yakir Vizel. Hyperproperty verification as chc satisfiability. In Stephanie Weirich, editor, *Programming Languages and Systems*, pages 212–241, Cham, 2024. Springer Nature Switzerland.

[205] Alexander Ivrii and Arie Gurfinkel. Pushing to the top. In *Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design*, FMCAD '15, page 65–72, Austin, Texas, 2015. FMCAD Inc.

[206] Alexander Ivrii, Ziv Nevo, and Jason Baumgartner. k-fair = k-liveness + fair revisiting sat-based liveness algorithms. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–5, 2018.

[207] Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The first reactive synthesis competition (SYNTCOMP 2014). *Int. J. Softw. Tools Technol. Transf.*, 19(3):367–390, 2017.

[208] B. Jonsson and Yih-Kuen Tsay. Assumption/guarantee specifications in linear-time temporal logic. *Theor. Comput. Sci.*, 167:47–72, 1996.

[209] Bengt Jonsson and Tsay Yih-Kuen. Assumption/guarantee specifications in linear-time temporal logic. *Theoretical Computer Science*, 167(1):47–72, 1996.

[210] Dejan Jovanovic and Bruno Dutertre. Property-directed k-induction. In *FMCAD*, pages 85–92. IEEE, 2016.

[211] Temesghen Kahsai and Cesare Tinelli. Pkind: A parallel k-induction based model checker. In *PDMC*, volume 72 of *EPTCS*, pages 55–62, 2011.

[212] Yonit Kesten, Amir Pnueli, and Li-on Raviv. Algorithmic Verification of Linear Temporal Logic Specifications. In *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1998.

[213] Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki. SMT-based model checking for recursive programs. *Formal Methods Syst. Des.*, 48(3):175–205, 2016.

[214] Roman Kontchakov, Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. Temporalising tableaux. *Stud Logica*, 76(1):91–134, 2004.

[215] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4):255–299, oct 1990.

[216] Andreas Krebs, Arne Meier, Jonni Virtema, and Martin Zimmermann. Team Semantics for the Specification and Verification of Hyperproperties. In *Proc. 43rd MFCS*, LIPIcs 117, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[217] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods Syst. Des.*, 34(2):83–103, 2009.

[218] Orna Kupferman and Moshe Y. Vardi. Modular model checking. In Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors, *Compositionality: The*

Significant Difference, pages 381–401, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[219] Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[220] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. SMT techniques for fast predicate abstraction. In *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 424–437. Springer, 2006.

[221] Leslie Lamport. Temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16:872–923, 05 1994.

[222] Leslie Lamport. The operators of tla. 06 1997.

[223] Timo Latvala. Efficient Model Checking of Safety Properties. In *SPIN*, volume 2648 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2003.

[224] Jianwen Li, Rohit Dureja, Geguang Pu, Kristin Yvonne Rozier, and Moshe Y. Vardi. Simplecar: An efficient bug-finding tool based on approximate reachability. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 37–44, Cham, 2018. Springer International Publishing.

[225] Jianwen Li, Geguang Pu, Yueling Zhang, Moshe Y. Vardi, and Kristin Y. Rozier. Sat-based explicit ltlf satisfiability checking. *Artif. Intell.*, 289:103369, 2020.

[226] Jianwen Li, Moshe Y. Vardi, and Kristin Y. Rozier. Satisfiability checking for mission-time ltl. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 3–22, Cham, 2019. Springer International Publishing.

[227] Jianwen Li, Shufang Zhu, Yueling Zhang, Geguang Pu, and Moshe Y. Vardi. Safety model checking with complementary approximations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 95–100, 2017.

[228] O. Lichtenstein, A. Pnueli, and L.D. Zuck. The Glory of the Past. In *Logics of Programs*, pages 196–218, 1985.

[229] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Logic of Programs*, volume 193 of *LNCS*, pages 196—218. Springer, 1985.

[230] Christophe Limbree, Quentin Cappart, Charles Pecheur, and Stefano Tonetta. Verification of railway interlocking - compositional approach with ocra. pages 134–149, 06 2016.

[231] Cong Liu, Junaid Babar, Isaac Amundson, Karl Hoech, Darren D. Cofer, and Eric Mercer. Assume-guarantee reasoning with scheduled components. In *NASA Formal Methods*, 2022.

[232] Florian Lonsing and Uwe Egly. *Incremental QBF Solving*, page 514–530. Springer International Publishing, 2014.

[233] Florian Lonsing and Uwe Egly. Depqbf 6.0: A search-based qbf solver beyond traditional qcdcl. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 371–384, Cham, 2017. Springer International Publishing.

[234] Martin Lück. On the complexity of linear temporal logic with team semantics. *Theor. Comput. Sci.*, 837:1–25, 2020.

[235] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[236] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *REX*, volume 600 of *LNCS*, pages 447–484, 1991.

[237] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[238] Frédéric Mallet. Ccsl: specifying clock constraints with uml/marte. 01 2008.

[239] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.

[240] Nicolas Markey. Temporal logic with past is exponentially more succinct. *Bulletin of the EATCS*, 79:122–128, 01 2003.

[241] Corto Mascle and Martin Zimmermann. The Keys to Decidable HyperLTL Satisfiability: Small Models or Very Simple Formulas. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[242] D. McCullough. Noninterference and the composability of security properties. In *Proceedings. 1988 IEEE Symposium on Security and Privacy*, pages 177–186, 1988.

[243] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 79–93, 1994.

[244] K. L. McMillan. A compositional rule for hardware design refinement. In Orna Grumberg, editor, *Computer Aided Verification*, pages 24–35, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[245] K. L. McMillan. Circular compositional reasoning about liveness. In Laurence Pierre and Thomas Kropf, editors, *Correct Hardware Design and Verification Methods*, pages 342–346, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[246] K. L. McMillan. Interpolation and sat-based model checking. In Warren A. Hunt and Fabio Somenzi, editors, *Computer Aided Verification*, pages 1–13, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[247] Kenneth L. McMillan. Circular Compositional Reasoning about Liveness. In *CHARME*, volume 1703 of *Lecture Notes in Computer Science*, pages 342–345. Springer, 1999.

[248] Bertrand Meyer. Applying "design by contract". 1992.

[249] Anitha Murugesan, Michael W. Whalen, Sanjai Rayadurgam, and Mats P.E. Heimdahl. Compositional verification of a medical device system. In *Proceedings of the 2013 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '13, page 51–64, New York, NY, USA, 2013. Association for Computing Machinery.

[250] Luan Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy Deshmukh, and Taylor Johnson. Hyperproperties of real-valued signals. pages 104–113, 09 2017.

[251] Aina Niemetz, Mathias Preiner, Clifford Wolf, and Armin Biere. Btor2 , btormc and boolector 3.0. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 587–595. Springer, 2018.

[252] James Jerson Ortiz, Moussa Amrani, and Pierre-Yves Schobbens. $\mathrm{ML}_{\mathcal{V}}$: A Distributed Real-Time Modal Logic. In *NFM*, pages 19–35, 2019.

[253] James Jerson Ortiz, Axel Legay, and Pierre-Yves Schobbens. Distributed Event Clock Automata - Extended Abstract. In *CIAA*, pages 250–263, 2011.

[254] Corina S. Pasareanu, Matthew B. Dwyer, and Michael Huth. Assume-Guarantee Model Checking of Software: A Comparative Case Study. In *SPIN*, volume 1680 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 1999.

[255] Corina S. Pasareanu, Dimitra Giannakopoulou, Mihaela Gheorghiu Bobaru, Jamieson M. Cobleigh, and Howard Barringer. Learning to divide and conquer: applying the l* algorithm to automate assume-guarantee reasoning. *Formal Methods Syst. Des.*, 32(3):175–205, 2008.

[256] Lee Pike and Steven D. Johnson. The formal verification of a reintegration protocol. In *EMSOFT*, pages 286–289. ACM, 2005.

[257] Amir Pnueli. The temporal logic of programs. pages 46–57, 09 1977.

[258] Pyvmt. `https://github.com/pyvmt/pyvmt`, 2022.

[259] Sophie Quinton and Susanne Graf. Contract-based verification of hierarchical systems of components. *2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 377–381, 2008.

[260] Markus N. Rabe. *A temporal logic approach to information-flow control*. PhD thesis, Saarland University, 2016.

[261] J.-F. Raskin and P.-Y. Schobbens. The Logic of Event Clocks - Decidability, Complexity and Expressiveness. *Journal of Automata, Languages and Combinatorics*, 4(3):247–286, 1999.

[262] Thomas Reinbacher, Kristin Rozier, and Johann Schumann. Temporal-logic based runtime observer pairs for system health management of real-time systems. 04 2014.

[263] Andoni Rodríguez and César Sánchez. Boolean abstractions for realizability modulo theories. In *CAV (3)*, volume 13966 of *Lecture Notes in Computer Science*, pages 305–328. Springer, 2023.

[264] Andoni Rodríguez and César Sánchez. Adaptive reactive synthesis for LTL and ltlf modulo theories. In *AAAI*, pages 10679–10686. AAAI Press, 2024.

[265] Guillermo Rodríguez-Navas and Julián Proenza. Using Timed Automata for Modeling Distributed Systems with Clocks: Challenges and Solutions. *IEEE Trans. Software Eng.*, 39(6):857–868, 2013.

[266] Willem-Paul Roever, Frank Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. 01 2001.

[267] Kristin Rozier and Moshe Vardi. Ltl satisfiability checking. *STTT*, 12:123–137, 01 2010.

[268] John M. Rushby and Friedrich W. von Henke. Formal verification of the interactive convergence clock synchronization algorithm using ehdm. 1989.

[269] Ondrej Rysavy and Jaroslav Rab. A formal model of composing components: The tla+ approach. *Innovations in Systems and Software Engineering*, 5:139–148, 06 2009.

[270] Meera Sampath, Raja Sengupta, Stephen Lafortune, Kazin Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Autom. Control.*, 40(9):1555–1575, 1995.

[271] Viktor Schuppan and Luthfi Darmawan. Evaluating LTL satisfiability solvers. In *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 397–413. Springer, 2011.

[272] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In Warren A. Hunt and Steven D. Johnson, editors, *Formal Methods in Computer-Aided Design*, pages 127–144, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[273] Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified boolean formulas. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 78–84, 2019.

[274] A. Prasad Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, 1985.

[275] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49:217–237, 1987.

[276] Chen Su, Min Zhou, Liangze Yin, Hai Wan, and Ming Gu. Modeling and verification of component-based systems with data passing using bip. In *2013 18th International Conference on Engineering of Complex Computer Systems*, pages 4–13, 2013.

[277] Yuheng Su, Qiusong Yang, Yiwei Ci, Tianjun Bu, and Ziyu Huang. The ric3 hardware model checker, 2025.

[278] Geoff Sutcliffe. The TPTP problem library and associated infrastructure - from CNF to th0, TPTP v6.4.0. *J. Autom. Reason.*, 59(4):483–502, 2017.

[279] Niklas Sörensson and Niklas Een. Minisat v1.13-a sat solver with conflict-clause minimization. *International Conference on Theory and Applications of Satisfiability Testing*, 01 2005.

[280] Nadra Tabassam, Martin Fränzle, Muhammad Waleed Ansari, and Ghazala Shaheen. Review: Contract-based design methodologies for cyber-physical systems. In Atulya Nagar, Dharm Singh Jat, Durgesh Mishra, and Amit Joshi, editors, *Intelligent Sustainable Systems*, pages 137–148, Singapore, 2025. Springer Nature Singapore.

[281] Stefano Tonetta. Abstract model checking without computing the abstraction. In Ana Cavalcanti and Dennis R. Dams, editors, *FM 2009: Formal Methods*, pages 89–105, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[282] Stefano Tonetta. Linear-time Temporal Logic with Event Freezing Functions. In *GandALF*, volume 256 of *EPTCS*, pages 195–209, 2017.

[283] Hoang-Viet Tran, Pham Ngoc Hung, Viet-Ha Nguyen, and Toshiaki Aoki. A framework for assume-guarantee regression verification of evolving software. *Science of Computer Programming*, 193:102439, 2020.

[284] Hiroshi Unno, Tachio Terauchi, and Eric Koskinen. Constraint-based relational verification. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification*, pages 742–766, Cham, 2021. Springer International Publishing.

[285] Moshe Y. Vardi. Branching vs. linear time: Final showdown. In *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[286] Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society, 1986.

[287] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994.

[288] Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity. In *Proc. 41st IARCS FSTTCS*, LIPIcs 213, pages 52:1–52:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[289] Yakir Vizel and Orna Grumberg. Interpolation-sequence based model checking. *2009 Formal Methods in Computer-Aided Design*, pages 1–8, 2009.

[290] Masaki Waga and Étienne André. Hyper parametric timed ctl. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):4286–4297, 2024.

[291] Farn Wang, Aloysius K. Mok, and E. Allen Emerson. Distributed Real-Time System Specification and Verification in APTL. *TOSEM*, 2(4):346–378, 1993.

[292] Yu Wang, Siddhartha Nalluri, and Miroslav Pajic. Hyperproperties for robotics: Planning via hyperltl. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8462–8468, 2019.

[293] Yu Wang, Mojtaba Zarei, Borzoo Bonakdarpour, and Miroslav. Pajic. Statistical verification of hyperproperties for cyber-physical systems. *ACM Transactions on Embedded Computing systems (TECS)*, 18(5s):92:1–92:23, 2019.

[294] Zili Wang, Katherine Kosaian, and Kristin Yvonne Rozier. Formally verifying a transformation from mltl formulas to regular expressions. In Arie Gurfinkel and Marijn Heule, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–275, Cham, 2025. Springer Nature Switzerland.

[295] Kazuki Watanabe, Clovis Eberhart, Kazuyuki Asada, and Ichiro Hasuo. Compositional probabilistic model checking with string diagrams of mdps. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification*, pages 40–61, Cham, 2023. Springer Nature Switzerland.

[296] Yechuan Xia, Anna Becchi, Alessandro Cimatti, Alberto Griggio, Jianwen Li, and Geguang Pu. *Searching for i-Good Lemmas to Accelerate Safety Model Checking*, pages 288–308. 07 2023.

[297] Yechuan Xia, Alessandro Cimatti, Alberto Griggio, and Jianwen Li. Avoiding the shoals - a new approach to liveness checking. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification*, pages 234–254, Cham, 2024. Springer Nature Switzerland.

[298] Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *Proc. of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 29–43. IEEE, 2003.

[299] Yuanrui Zhang, Frederic Mallet, Min Zhang, and Zhiming Liu. Specification and verification of multi-clock systems using a temporal logic with clock constraints. *Form. Asp. Comput.*, 36(2), June 2024.

[300] Jianing Zhao, Shaoyuan Li, and Xiang Yin. A unified framework for verification of observational properties for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 69:4710–4717, 2022.

[301] Shufang Zhu, Giuseppe De Giacomo, Geguang Pu, and Moshe Y. Vardi. Ltlf synthesis with fairness and stability assumptions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):3088–3095, Apr. 2020.

[302] Shufang Zhu, Lucas Tabajara, Jianwen Li, Geguang Pu, and Moshe Vardi. Symbolic ltlf synthesis. pages 1362–1369, 08 2017.