UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Facing infinity in model checking expressive specification languages

Extending the support for temporal logic model checking in infinite-state systems

Enrico Magnago

Advisor
Dr. Alessandro Cimatti
Fondazione Bruno Kessler

Co-Advisor
Dr. Alberto Griggio
Fondazione Bruno Kessler

April 2022

Abstract

Society relies on increasingly complex software and hardware systems, hence techniques capable of proving that they behave as expected are of great and growing interest. Formal verification procedures employ mathematically sound reasoning to address this need.

This thesis proposes novel techniques for the verification and falsification of expressive specifications on timed and infinite-state systems. An expressive specification language allows the description of the intended behaviour of a system via compact formal statements written at an abstraction level that eases the review process. Falsifying a specification corresponds to identifying an execution of the system that violates the property (i.e. a witness). The capability of identifying witnesses is a key feature in the iterative refinement of the design of a system, since it provides a description of how a certain error can occur. The designer can analyse the witness and take correcting actions by refining either the description of the system or its specification.

The contribution of this thesis is twofold. First, we propose a semantics for Metric Temporal Logic *that considers four different models of time (*discrete, dense, super-discrete *and* super-dense*). We reduce its verification problem to finding an infinite fair execution (witness) for an infinite-state system with discrete time. Second, we define a novel SMT-based algorithm to identify such witnesses. The algorithm employs a general representation of such executions that is both informative to the designer and provides sufficient structure to automate the search of a witness.*

We apply the proposed techniques to benchmarks taken from software, infinite-state, timed and hybrid systems. The experimental results highlight that the proposed approaches compete and often outperform specific (application tailored) techniques currently used in the state of the art.

Keywords
[Formal Verification, Model Checking, Linear-time Temporal Logic, Metric Temporal Logic, Infinite-state Systems, Timed Systems, SMT]

Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my gratitude to both my advisors, Alessandro Cimatti and Alberto Griggio, for the support and guidance they provided me during the doctorate. Their insight and experience in formal verification greatly affected the quality of the work I have done in these years. This thesis is the summary and final product of that journey.

I would like to thank also Marco Roveri and Stefano Tonetta. The first for his contribution in the design of software modules implemented within nuXmv and for lending me his deep knowledge of the code-base upon which the model checker is built. The second one for his support as the head of the Embedded Systems unit in FBK and to him goes the credit for the initial encoding of the interval semantics required by MTL.

A key enabler for my work was the library pysmt. For this reason I am grateful to its creators and main maintainers: Andrea Micheli and Marco Gario.

I thank all my colleagues and friends for their support and different points of view on the many issues faced during my PhD.

Finally, these years have been characterised by a global pandemic that greatly affected our way of living. Reaching the conclusion of my doctorate program in such uncertain times would have been much harder without the continuous support of my family. For this reason, I am particularly grateful to my mother Monica, my father Pierluigi and my brother Valerio.

Chapter 1

Introduction

1.1 Context and Motivation

Society relies on computer and software systems to perform safety-critical tasks with absolute reliability. Some examples of domains where safety is paramount are transportation, health care and avionics. An error in such systems can lead to severe undesirable effects such as injuries and economic losses. In general, due to the global spread of technological devices, an issue in one of them can easily affect millions of people and cause severe global consequences even in contexts that are usually not considered as safety-critical. For these reasons, it is very important to define a development process that minimises the probability of such events. Verification and Validation (V&V) tasks aim at increasing the confidence that the system under development meets its requirements. The IEEE standard 1012 [1] defines a common framework for the inclusion of V&V activities in all system, software and hardware life-cycle processes.

A common approach to implement V&V tasks is by performing extensive tests on the product. This technique has two major disadvantages: it requires a fully developed system and it guarantees that the system behaves correctly only in the particular cases that were tested. In addition, mistakes made in the early stages of the development process will be de-

tected only at its end, when most of the cost has already been sustained. These observations highlight the need for tools that support V&V in the early phases of the development process.

Formal verification is a broad field encompassing techniques that aim at mathematically proving the correctness of a system with respect to its formal specification. It provides strategies and procedures to systematically analyse the system and either conclude its correctness or identify some issue. These techniques reason on a system described by means of some formal language, usually mathematical logic, and employ rigorous procedures to infer statements about it. Approaches in this context can be broadly characterised by the expressiveness of the mathematical formalism they employ, their level of automation and the artefacts they generate, such as proofs and counterexamples.

This thesis is placed in the context of *model checking*. Model checkers exhaustively explore the mathematical model of the system in order to conclude whether or not a given specification is satisfied. Model checking techniques automatically verify whether the formal model satisfies a set of specifications, also called properties, expressed in some logic. They offer a high degree of automation, usually support first-order and temporal logic, and are capable of generating counterexamples that show a given property to be false. Fig. 1.1 reports the abstract input-output schema of a model checker. A model checker, given the description of the system and a specification, explores the model until it either identifies a violation of the specification, called counterexample, or concludes that the specification is valid in the model. The analysis of the counterexample allows the user to identify errors in the model or in the specification itself, hence allowing their correction before proceeding to the next phases of the development.

Model checkers are usually characterised by the modelling and specification languages they support. These two elements represent the interface

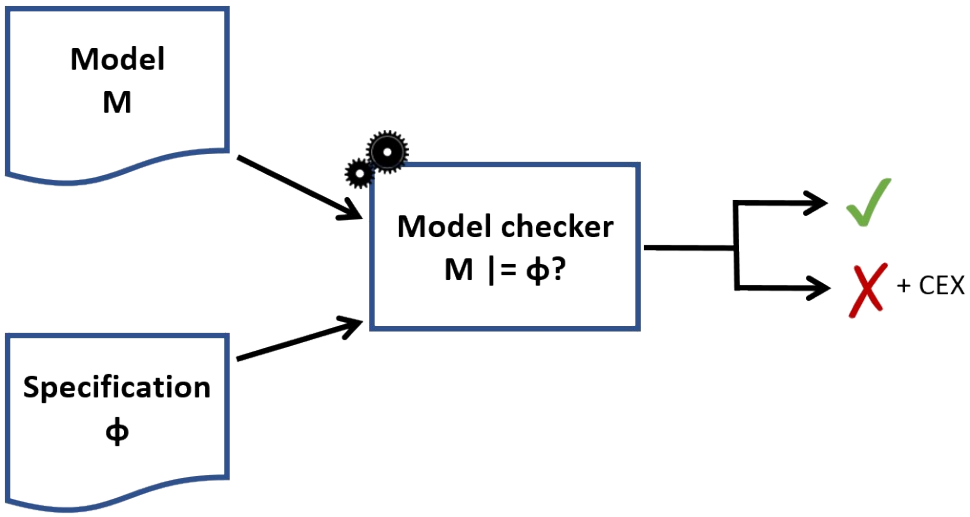

Figure 1.1: Schema of input-output of a model checker.

with the user and different languages are better suited for different contexts. Over the years many modelling formalisms have been proposed. They represent a system from different points of view and at different levels of detail. System designers use these formal representations to express a model that is precise enough to highlight possible inconsistencies and ease the review process. The modelling and specification languages should provide an abstraction that is as close as possible to the application scenario, in order to reduce the cost of creating such models and decrease the likelihood of modelling errors.

Timed, hybrid and infinite automata are some examples of formal models used to represent infinite-state systems. Infinite automata evolve through discrete steps and have been used to model software programs. Timed and hybrid automata, instead, describe systems composed of both discrete and continuous components. Some examples of systems that can be modelled using these formalisms are embedded systems that deal with physical phenomena or have real-time constraints. These kind of devices and software are ubiquitous in modern technology and we need to develop ways to prove their correctness, especially when they perform safety-critical tasks.

1.2 Contributions

This thesis is concerned with model checking expressive specification languages on timed and infinite-state systems. The contributions are organised along two directions. First, we define expressive temporal logics suitable to write specifications for timed systems and provide a novel reduction-based technique for their verification. Then, we enhance the capability of model checkers to identify and represent counterexamples.

1. We extend First-Order Linear-time Temporal Logic with Past (LTL) adding two operators "at next" and "at last"; we call this new logic LTL-EF. LTL-EF can be interpreted with four different models of time: *discrete*, *dense*, *super-discrete* and *super-dense*. We show that LTL-EF is expressive enough to encode $\text{MTL}_{0,\infty}$ with counting. Therefore, we provide a unifying semantics for both LTL-EF and $\text{MTL}_{0,\infty}$ specifications to be used for the verification of systems that rely on different models of time. We provide rewriting procedures to reduce the model checking problem to the discrete-time case and to remove the extra functional symbols. Therefore, we reduce model checking of LTL, LTL-EF and $\text{MTL}_{0,\infty}$ specifications on timed systems to the model checking problem of LTL specifications on infinite-state transition systems with discrete time model. The verification of LTL specifications on infinite-state transition systems with discrete time is a widely studied model checking problem and well-known procedures allow its reduction to the problem of identifying fair (infinite) executions of a corresponding infinite-state system.

   The results obtained in this direction have been published in [55] and [54]. These works have been performed in collaboration with Stefano Tonetta and Marco Roveri. In this thesis, we further extend them by considering an additional model of time called super-discrete.

2. In the second part of this thesis, we focus on the problem of identifying fair (infinite) executions of systems with a possibly infinite state-space. We first define a novel structure, called *funnel-loop*, for the representation of fair executions and show it to be relatively complete. Then, we partition the search space of funnel-loops along two orthogonal directions: *segmentation* and *decomposition*. We define two search procedures: one is a reduction to an established framework, the other is a direct procedure that exploits the partitioning of the search space. Finally, we show how decomposition allows for the use of specialised techniques to analyse portions of the system that meet some additional requirements.

   This thesis extends the decomposition approach we presented in [52] to support *ranking functions* and integrates it with the segmentation technique we developed in [51]. This integration has been published in the form of a Journal paper in [53]. Finally, this work generalises the approach we employed in [54] to identify infinite traces and presents it as a specialised technique for the compositional framework.

We implemented the reduction approach and the specialised technique to identify counterexamples with diverging symbols in the nuXmv model checker enabling the verification of LTL-EF and $\mathrm{MTL}_{0,\infty}$ specifications and significantly enhancing its capability of identifying counterexamples for timed systems. The direct procedure for the search of funnel-loops has been implemented in a tool called F3. We show the feasibility of the approach experimenting with several non-trivial valid and satisfiable $\mathrm{MTL}_{0,\infty}$ and LTL-EF formulae. In addition, we evaluated the effectiveness of our falsification procedures by considering verification problems taken from a broad range of contexts: software nontermination, infinite-state transition systems, timed automata, timed transition systems and hybrid systems.

1.3 Structure of the Thesis

This thesis is divided into four parts.

Part I introduces the background notions used throughout the thesis in Chapter 2 and describes the main problems it addresses in Chapter 3.

Part II is composed of a single chapter: Chapter 4. The chapter begins in Sections 4.1 and 4.2 that describe the contributions related to the definition of extended temporal logics over different time models. Then, Section 4.3 reports the reduction of their model checking problem to the discrete-time case. Finally, Chapter 4 concludes in Section 4.4 by discussing the related works on model checking metric temporal logics.

The novel representation and search procedures for infinite fair paths are described in Part III. Chapter 5 presents the structure used to represent such paths and Chapter 6 discusses the partitioning of the associated search space. The search procedures are detailed in Chapter 7, while Chapter 8 describes how the decomposition allows for the use of specialised techniques and also defines a novel one. Finally, Chapter 9 discusses the related works in different contexts. Section 9.1 considers term rewriting systems, while Section 9.2 discusses techniques for identifying nonterminating executions of software programs. The model checking problem of linear-time temporal logic on infinite-state systems, timed automata and hybrid systems are considered in Sections 9.3, 9.4 and 9.5 respectively.

Part IV introduces, in Chapter 10, the tools nuXmv and F3 that implement the techniques described in this thesis and provides some additional implementation details. Chapter 11 describes the experiments we performed using these techniques and discusses the results we obtained.

Chapter 12 concludes the thesis and outlines some possible future works.

Part I

Background Notions

Chapter 2

Background

This chapter introduces the background concepts used throughout this thesis. Most of the notions it presents are well established in the literature and we limit our discussion to the main concepts relevant for this thesis. An extensive and broader description of the subjects introduced in this chapter can be found in [15, 93].

We assume the reader is familiar with propositional logic, its satisfiability problem and SAT solvers. We use $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ for the sets of natural, integer, rational and real numbers respectively and write $\mathbb{R}_0^+$ for the set of non-negative reals. In addition, for $n \in \mathbb{N}$ we write $+_n$ for the sum modulo n.

The chapter is organised as follows. Sec. 2.1 introduces First-Order Logic and its satifiability problem. Then, we describe two structures that can be used to prove whether a first-order relation admits some infinite chain of elements: well-founded relations, in Sec. 2.2, and recurrent sets, in Sec. 2.3. Sec. 2.4 defines two formalisms based on First-Order Logic and well-founded relations that can be used to represent verification problems: CHC and E-CHC. The formalisms used in this thesis to represent discrete systems are presented in Sec. 2.5, while the ones used for timed system and the different models of time are discussed in Sec. 2.6. Both syntax and

semantics of the specification language LTL are reported in Sec. 2.7; its semantics is defined for each of the time models considered in this thesis. Finally, in Sec. 2.8 we briefly describe the model checking problem and discuss some techniques that have been proposed in the literature in the context of symbolic model checking.

2.1 First-Order Logic and SMT

Logical languages express mathematically precise statements about a set of objects. Propositional logic is one such language; it predicates about objects that can be interpreted as either true or false and statements are built using the propositional operators: conjunction, negation, disjunction, implication and iff. *First-Order Logic* (FOL) extends propositional logic with universal and existential quantifiers. In addition, in first-order logic an object can be not only a fact (true or false) but also a constant, a predicate or a function. An interpretation in first-order logic is not simply an assignment to the variables but maps every function and predicate symbol to a corresponding mathematical function and relation, i.e. a specific element in the set of functions and predicates with the same arity. First-order logic is expressive enough to formalise arithmetical operations over naturals, rationals and real numbers; for example, Peano arithmetic is a first-order theory (set for first-order formulae) that characterises arithmetic over the natural numbers.

We call first-order *signature* and write Σ for the set of predicate and function symbols, with their arity, that can appear in the FOL formulae. We assume the symbols $\top, \bot, =$ to be in Σ and interpreted as true, false and the identity function respectively. In addition, we write $\mathcal{V}$ for the set of variables.

FOL formulae and sentences. A 0-ary predicate symbol is called *Boolean atom*, while a 0-ary function symbol is a *constant*. A Σ-term is either a variable or an n-ary function symbol in Σ applied to n Σ-terms. Given a n-ary predicate $p \in \Sigma$ and a set $\{t_i\}_{i=1}^{n}$ of Σ-terms, $p(t_1, \ldots, t_n)$ is a Σ-atom. A *literal* is either a Σ-atom or its negation. We define Σ-formulae using Σ-atoms, the existential ($\exists$) quantifier, and the Boolean connectives conjunction ($\wedge$) and negation ($\neg$). Therefore, a FOL Σ-formula ϕ can be one of the following: a Σ-atom a, the negation of a FOL Σ-formula ϕ_0, the conjunction of two FOL Σ-formulae ϕ_0 and ϕ_1, or the exitential quantification of a set of symbols $X \subseteq \mathcal{V}$ for a FOL Σ-formula ϕ_0.

$$\phi \ :: \ a \mid \neg\phi \mid \phi \wedge \phi \mid \exists X : \phi$$

By combining the operators above it is possible to obtain the operators representing disjunction, implication, iff and universal quantification as follows:

- universal quantification, $\forall X : \phi$ can be written as $\neg\exists X : \neg\phi$;
- disjunction, $\phi \vee \psi$ can be written as $\neg(\neg\phi \wedge \neg\psi)$;
- implication, $\phi \rightarrow \psi$ can be written as $\neg(\phi \wedge \neg\psi)$;
- iff, $\phi \leftrightarrow \psi$ can be written as $(\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)$.

We call *clause* [resp. *cube*] a formula written as a disjunction [resp. conjunction] of literals.

A variable is *free* in a Σ-formula ϕ iff it is not quantified in ϕ. We call *sentence* a formula with no free variables.

Furthermore, we write $\phi[\psi_0/\psi_1]$ for the formula obtained by substituting every occurrence of ψ_0 with ψ_1 in ϕ.

FOL theories and models. A first-order Σ-theory $\mathcal{T}$ is a set of first-order sentences with signature Σ. A Σ-structure $\mathcal{M}$ is a model of a Σ-theory $\mathcal{T}$

iff $\mathcal{M}$ satisfies every sentence in $\mathcal{T}$. A Σ-formula ϕ is $\mathcal{T}$-satisfiable iff it is satisfiable in a model $\mathcal{M}$ of $\mathcal{T}$, written $\mathcal{M} \models_{\mathcal{T}} \phi$. A Σ-formula is $\mathcal{T}$-valid iff it is satisfiable in every model of $\mathcal{T}$ and the negation of a $\mathcal{T}$-valid formula is $\mathcal{T}$-unsatisfiable. Two Σ-formulae ϕ and ψ are $\mathcal{T}$-equivalent iff every model of ϕ is also a model of ψ and equisatisfiable if ϕ is $\mathcal{T}$-satisfiable iff ψ is also $\mathcal{T}$-satisfiable.

When the theory and signature are clear from the context we overload the $\models$ symbol. If ϕ and ψ are formulae, then $\phi \models \psi$ stands for entailment restricted to the models of the background theory. Simlarly, for a model $\mathcal{M}$ and formula ϕ we write $\mathcal{M} \models \phi$ omitting the backgound theory $\mathcal{T}$. In contrast, if M is a fair transition system and ψ is a linear temporal property, then $M \models \psi$ means that ψ holds in every infinite path of M; it is interpreted with the LTL semantics described in Sec. 2.7.

2.1.1 Normal Forms

FOL formulae can have a complex structure and their syntactical representation is not unique. In fact, equivalent formulae can have very different syntactical representation. As we have already seen above, a relatively small number of operators is sufficient to express all the others. Normal forms exploit such rewritings to simplify the syntactical structure of the formula and obtain an equivalent or equisatisfiable one with a fixed syntactical structure. This enables us to define procedures that rely on a fixed syntactical structure and prove their correctness by considering only a subset of the operators, without loss of generality. In the following, we describe three normal forms: *prenex*, *conjunctive* and *disjunctive* normal forms. In addition, we describe *Horn-clauses* as a special case of conjunctive normal form. Horn-clauses admit a polynomial-time solution for the satisfiability problem in the propositional case.

Prenex Normal Form

A FOL formula is in *prenex normal form* (PNF) if it is written as a sequence of quantifiers and bound variables followed by a quantifier-free formula. Every formula can be rewritten in PNF by recursively applying the following rules:

- $(\forall X : \phi) \wedge \psi$ is equivalent to $\forall X \phi \wedge \psi$;
- $(\exists X : \phi) \wedge \psi$ is equivalent to $\exists X \phi \wedge \psi$;
- $\neg \forall X : \phi$ is equivalent to $\exists X : \neg \phi$;
- $\neg \exists X : \phi$ is equivalent to $\forall X : \neg \phi$.

The rules above assume that ψ does not contain any variable in X. If this is not the case, then we first need to rewrite ψ by renaming all such variables.

Example 1 - PNF

Consider the formula $\forall x : x \neq 0 \rightarrow \exists y : x \cdot y = 1$, for real variables x and y. The formula states that all real values different from 0 admit a corresponding reciprocal real number ($\frac{1}{x}$). The equivalent PNF formula is $\forall x \exists y : x \neq 0 \rightarrow x \cdot y = 1$.

Conjunctive Normal Form

A FOL formula is in *conjunctive normal form* (CNF) if it is in PNF and the quantifier-free formula is a conjunction of clauses. Every formula can be rewritten in CNF, by first writing it in PNF and then recursively applying the following rules:

- $(\phi_0 \wedge \phi_1) \vee (\psi_0 \wedge \psi_1)$ is equivalent to $(\phi_0 \vee \psi_0) \wedge (\phi_0 \vee \psi_1) \wedge (\phi_1 \vee \psi_0) \wedge (\phi_1 \vee \psi_1)$;
- $\neg(\phi \wedge \psi)$ is equivalent to $\neg \phi \vee \neg \psi$;
- $\neg \neg \phi$ is equivalent to ϕ.

This rewriting produces equivalent CNF formulae that can be exponentially larger than the original one. Other rewriting approaches (e.g. Tseitin transformation [158]) avoid this problem by introducing additional Boolean variables to obtain an equisatisfiable formula in polynomial time.

Horn clauses. A relevant subcase of CNF formulae are *Horn* formulae. A *Horn* formula is a CNF formula such that each clause contains at most one non-negated literal, called Horn clause. Each Horn clause can be written equivalently as an implication such that the implicant (left-hand-side of the implication) is a conjunction of atoms and the implicate (right-hand-side of the implication) is either $\top$ or a single atom. The SAT problem in the case of Horn formulae has a polynomial time solution. In fact, it is sufficient to build a model by first propagating the truth value of every unit clause (clause containing a single literal) and then assigning all remaining Boolean variables to $\bot$. The Horn propositional formula is satisfiable iff it is satisfied by such model.

Example 2 - Horn Clauses
Given four Boolean atoms a, b, c and d, the formula $a \wedge (\neg a \vee b) \wedge (\neg b \vee \neg c \vee d)$ is in CNF and also composed of Horn clauses. It can be equivalently rewritten using implications as follows: $(\top \rightarrow a) \wedge (a \rightarrow b) \wedge (b \wedge c \rightarrow d)$. It is easy to see that the formula is satisfiable. a must hold, hence we must assign a to $\top$. $a \rightarrow b$ must also hold and, since we already assigned a to $\top$, then also b must be true. In these two steps we performed unit propagations: we selected the only possible truth assignment for the clauses composed of a single literal and propagated them to the other clauses. After the propagation of these two assignments, the only remaining subformula is $c \rightarrow d$. In order for $c \rightarrow d$ to hold it is sufficient to assign c to false. Therefore, a model for the initial formula is $a : \top$, $b : \top$, $c : \bot$ and $d : \bot$.

Disjunctive Normal Form

A FOL formula is in *disjunctive normal form* (DNF) if it is in PNF and the quantifier-free formula is a disjunction of cubes. Every formula can be rewritten in CNF, by first writing it in PNF and then applying recursively the following rules:

- $(\phi_0 \vee \phi_1) \wedge (\psi_0 \vee \psi_1)$ is equivalent to $(\phi_0 \wedge \psi_0) \vee (\phi_0 \wedge \psi_1) \vee (\phi_1 \wedge \psi_0) \vee (\phi_1 \wedge \psi_1)$;
- $\neg(\phi \vee \psi)$ is equivalent to $\neg\phi \wedge \neg\psi$;
- $\neg\neg\phi$ is equivalent to ϕ.

Notice that these rules are the "reversed" version of the rules we used for the CNF case. In fact, given a formula ϕ it is possible to rewrite it in DNF by first computing the equivalent formula $\neg CNF(\neg\phi)$ and then pushing the negations on the atoms by applying the following rules:

- $\neg\forall X : \phi$ is equivalent to $\exists X : \neg\phi$;
- $\neg\exists X : \phi$ is equivalent to $\forall X : \neg\phi$;
- $\neg(\phi \wedge \psi)$ is equivalent to $\neg\phi \vee \neg\psi$;
- $\neg(\phi \vee \psi)$ is equivalent to $\neg\phi \wedge \neg\psi$;
- $\neg\neg\phi$ is equivalent to ϕ.

2.1.2 Approximations, Implicant, Unsat-Core and Interpolant

Given two Σ-formulae ϕ and ψ, we say that ϕ *underapproximates* ψ iff every model of ϕ is also a model for ψ, written $\phi \models_{\mathcal{T}} \psi$. In addition, we call the inverse relation *overapproximation*, hence if ϕ underapproximates ψ, then ψ overapproximates ϕ. Finally, if ψ is quantifier-free and ϕ is a conjunction of (a subset of) the Σ-atoms of ψ such that $\phi \models_{\mathcal{T}} \psi$, then ϕ is an *implicant* of ψ.

Example 3 - Implicant
Consider the formula $(x > 0 \wedge y < 0 \rightarrow a) \vee (\neg a \wedge x \cdot y = 0 \rightarrow x \geq 0)$, for Boolean atom a and real variables x and y. An implicant for the formula is $x > 0 \wedge y < 0 \wedge a$. Notice that a formula can admit more that one implicant, in our example another implicant is given by a.

Given an unsatisfiable Σ-formula ϕ, the formula ψ is a *unsat-core* of ϕ iff: ψ is in CNF, it is unsatisfiable and the clauses of ψ are a subset of the clauses of $CNF(\phi)$.

Example 4 - Unsat-core
Consider the CNF formula $x = 0 \wedge (a \vee x > 1) \wedge (b \vee x \leq 0) \wedge (\neg a \vee x < -1)$. The formula is unsatisfiable and the conjunction of the three clauses $x = 0$, $(a \vee x > 1)$ and $(\neg a \vee x < -1)$ is sufficient to conclude its unsatisfiability. In this case we say that the three clauses are an unsat-core for the formula.

Given two Σ-formulae ϕ_0 and ϕ_1 such that $\phi_0 \wedge \phi_1 \models_{\mathcal{T}} \bot$. The formula ψ is a *Graig interpolant*, or simply interpolant, of the pair $\langle \phi_0, \phi_1 \rangle$ iff ψ is defined over the common symbols of ϕ_0 and ϕ_1, $\phi_0 \models_{\mathcal{T}} \psi$ and $\psi \wedge \phi_1 \models \bot$.

Example 5 - Interpolant
Consider two formulae $\phi_0 \doteq z > 5 \wedge x \geq z \wedge x + y < 0$ and $\phi_1 \doteq y > 0 \wedge x \cdot y < 0$. The conjunction of the two formulae is unsatisfiable and an interpolant for $\langle \phi_0, \phi_1 \rangle$ is $\psi \doteq x > 0$. In fact, ϕ_0 implies $z > 5$ and $x \geq z$, hence it implies $x > 0$; while ϕ_1 requires $y > 0$, but then $x \cdot y < 0$ holds iff $x < 0$, which contradicts the interpolant. Therefore, $\phi_0 \models_{\mathcal{T}} x > 0$ and $x > 0 \wedge \phi_1 \models_{\mathcal{T}} \bot$.

2.1.3 SMT Problem and SMT Solvers

In this context the *Satisfiability Modulo Theory* problem (SMT($\mathcal{T}$)) is the problem of checking if a Σ-formula is satisfiable for some background theory $\mathcal{T}$. We refer to such problems also as SMT queries.

SMT-solvers are software tools that implement sound, but possibly incomplete, decision procedures to address the SMT problem. Some examples of such tools are CVC5, MATHSAT5, YICES and Z3. Most state of the art solvers implement the "lazy approach" in which a SAT-solver is integrated with a theory solver ($\mathcal{T}$-solver). The SAT-solver enumerates the truth assignments to the propositional formula obtained by replacing the predicates containing $\mathcal{T}$ information with fresh Boolean variables. Therefore, the SAT-solver completely ignores the theory. Each assignment identified by the SAT-solver is a model for the Boolean abstracted formula and maps directly to a conjunction of $\mathcal{T}$-atoms. This conjunction of theory atoms is then given to the $\mathcal{T}$-solver. If the theory solver concludes that the theory-formula is satisfiable, then also the original formula is satisfiable. Instead, if it finds the theory-formula to be unsatisfiable, it generates a conflict set representing a reason for the unsatisfiability. In this case, the procedure refines the Boolean abstracted formula by adding, as a conjunction, the negation of this conflict set. The refinement reduces the search space that the SAT-solver has to explore by avoiding the repetition of the same "mistake".

SMT-solvers are capable of building models for satisfiable formulae and proofs in the unsatisfiable case. In addition, from a proof of unsatisfiability they can often produce an unsat-core or an interpolant.

Finally, most SMT-solvers provide an incremental interface. They tackle sequences of SMT-queries efficiently by reusing information discovered in previous searches. The interface is usually stack based and provides the primitives ASSERT, PUSH, POP and SOLVE. ASSERT allows the assertion of a new formula by pushing it onto the stack, PUSH sets a backtrack point for the stack, SOLVE checks the satisfiability of the conjunction of all formulae currently on the stack and POP restores the solver state, asserted formulae and learned clauses, to the last backtrack point.

2.1.4 Theory of Interest

In this thesis we use the theory of *quantified mixed integer-real nonlinear arithmetic* (NIRA). NIRA subsumes both *nonlinear real arithmetic* (NRA) and *nonlinear integer arithmetic* (NIA). In the case of NRA the SMT problem is decidable and a decision procedure is *Cylindrical Algebraic Decomposition* (CAD) [65]. However, the worst case time complexity of CAD is doubly-exponential in the degree of the polynomials. In addition, the SMT problem in NIA is undecidable [138]. Therefore, it easily follows that the SMT satisfiability problem in NIRA is also undecidable.

The signature of NIRA is given by $\Sigma \doteq \{0, 1, +, -, \cdot, /, =, \geq\}$, where 0 and 1 are interpreted as constants and the predicate symbols $\{+, -, \cdot, /, =, \geq\}$ are interpreted with the corresponding operations and relations on either integers or reals depending on the context. The comparison operators $\{<, >, \leq, \neq\}$ can be obtained from Σ as follows: $v_0 \leq v_1$ as $v_1 \geq v_0$, $v_0 < v_1$ as $\neq (v_0 \geq v_1)$, $v_0 \neq v_1$ as $\neg v_0 = v_1$ and $v_0 > v_1$ as $v_0 \geq v_1 \wedge v_0 \neq v_1$. For simplicity of notation we implicitly convert integer constants, variables and expressions to real when necessary. Given a set of constants $\{c_i\}_{i=0}^{n+1}$ and variables $\{v_i\}_{i=0}^{n} \subseteq \mathcal{V}$, $c_{n+1} + \sum_{i=0}^{n} c_i \cdot v_i$ is an affine expression and $\sum_{i=0}^{n} c_i \cdot v_i$ is a linear expression. We call linear [resp. affine] predicates the atoms constructed from a linear [resp. affine] expression compared with the constant 0.

In the following we always assume NIRA as backgound theory and we simply refer to sentences, formulae, predicates, atoms and terms omitting the signature Σ and, in addition, we simply state a formula to be satisfiable, valid or a logical consequence of another formula without explicitly stating the background theory.

In this context we call V the union of the set of variables $\mathcal{V}$ with the Boolean atoms in Σ. We call V the set of variables or symbols and denote

with $V' \doteq \{v' \mid v \in V\}$ the set containing the primed version of the symbols. We write $\phi(V)$ for a Boolean formula over the symbols V and $\phi(V, V')$ for a Boolean formula or relation over $V \cup V'$. When clear from the context we omit the set of symbols and simply write ϕ, ψ, ϕ' for $\phi(V)$, $\psi(V, V')$ and $\phi(V')$ respectively. We denote with $\boldsymbol{v}$ a total assignment over V, i.e. a state. Given a set of symbols V_0 we call *projection* of $\boldsymbol{v}$ over V_0, written $\boldsymbol{v}_{\downarrow V_0}$ the restriction of the assignments of $\boldsymbol{v}$ to the symbols in $V_0 \cap V$. Given a formula $\phi(V, V')$ we write $\phi(\boldsymbol{v}, \boldsymbol{v}')$ for the sentence given by the evaluation of ϕ obtained by replacing every symbol in V and V' with its corresponding assignment in $\boldsymbol{v}$ and $\boldsymbol{v}'$ respectively. In addition, given a formula $\phi(V)$ and assignment $\boldsymbol{v'}$ we will write $\phi(\boldsymbol{v}')$ for the evaluation of ϕ where every symbol $v \in V$ has been replaced with the assignment of v' in $\boldsymbol{v}'$. Similarly, for a formula $\phi(V')$ and assignment $\boldsymbol{v}$ $\phi(\boldsymbol{v})$ is the evaluation of ϕ where every symbol $v' \in V'$ has been replaced with the assignment of v in $\boldsymbol{v}$. We do this to simplify the notation and avoid the need to introduce many substitution operators. Finally, we write $\boldsymbol{v}, \boldsymbol{v}' \models \phi$ iff $\models \phi(\boldsymbol{v}, \boldsymbol{v}')$.

2.2 Well-Founded Relations

A binary relation $\rho \subseteq Q \times Q$ is *well-founded* if every nonempty subset $U \subseteq Q$ has a minimal element with respect to ρ, i.e. there is $m \in U$ such that no $u \in U$ satisfies $\rho(u, m)$ (i.e. $\langle u, m \rangle \in \rho$):

$$\forall U \subseteq Q \; \exists m \in U \; \forall u \in U : U \neq \emptyset \rightarrow \neg \rho(u, m)$$

In particular, a well-founded relation $\rho \subseteq Q \times Q$ does not admit any infinite chain: there is no infinite sequence $q_0, q_1, \ldots$ of elements of Q such that $\rho(q_i, q_{i+1})$ holds for every $i \geq 0$.

Given a relation $\phi(V, V')$, a *ranking function* $\textsc{Rf}(V)$ is a function from the assignments to the variables V to some set Q, such that the relation $< \doteq \{\langle \textsc{Rf}(\boldsymbol{v_0}), \textsc{Rf}(\boldsymbol{v'_1}) \rangle \mid \boldsymbol{v_0}, \boldsymbol{v'_1} \models \phi\}$ is well-founded and we call $\mathbf{0}$

its minimal element. Given a set of ranking functions $\{\text{RF}_i\}_{i=0}^{n}$, we define their sum as $\text{RF} \doteq \sum_{i=0}^{n} \text{RF}_i \doteq \langle \text{RF}_0, \ldots, \text{RF}_n \rangle$; RF is a ranking function with minimal element $\mathbf{0} \doteq \langle \mathbf{0}_0, \ldots, \mathbf{0}_n \rangle$ and comparison operator $< \doteq \{\langle \boldsymbol{v}_0, \boldsymbol{v}_1 \rangle | (\bigwedge_{i=0}^{n} \text{RF}_i(\boldsymbol{v}_0) \leq_i \text{RF}_i(\boldsymbol{v}_1)) \wedge (\bigvee_{i=0}^{n} \text{RF}_i(\boldsymbol{v}_0) <_i \text{RF}_i(\boldsymbol{v}_1))\}$ where $<_i$ is the well-founded relation associated with RF_i and $\leq_i \doteq <_i \cup =$ is the relation obtained by the union of $<_i$ and the identity relation $=$.

Example 6 - Ranking Function

Assume we want to prove that the following imperative procedure always terminates. We represent every iteration of such loop using the set of

```
1: while q > 0 do
2:     if y > 0 then
3:         q ← q − y − 1
4:     else
5:         q ← q + y − 1
6:     end if
7: end while
```

Figure 2.1: simple-imperative-procedure

symbols $V \doteq \{q, y\}$ for the assignments to the variables at the beginning of an iteration and $V' \doteq \{q', y'\}$ for their assignment at the end of the iteration. The formula $T(V, V') \doteq q > 0 \wedge y' = y \wedge ((y > 0 \wedge q' = q - y - 1) \vee (y \leq 0 \wedge q' = q + y - 1))$ relates the assignments at the beginning of the loop with the corresponding ones obtained by performing a single iteration.

In order for the program to be terminating, T must be a well-founded relation: there cannot be infinite chains of states. We prove T well-founded by identifying a ranking function $\text{RF} : V \mapsto \mathbb{N}$. In particular, let $\text{RF} \doteq \lceil q \rceil$ with minimal element 0 and we define the comparison operator such that $\text{RF}' < \text{RF} \leftrightarrow \lceil q' \rceil < \lceil q \rceil$. It can be easily observed that $T(V, V') \rightarrow q > 0$, hence $\text{RF} > 0$, and $T(V, V') \rightarrow q' \leq q - 1$, hence $\text{RF}' < \text{RF}$. Therefore,

every pair in T implies the ranking function is greater than its minimal element, and every such step makes it smaller. The relation $<$ in $\mathbb{N}$ is well-founded, hence there cannot be an infinite sequence such that $\text{RF} > 0$ holds in every state and $\text{RF}' < \text{RF}$ is true at every step. Therefore, T must be well-founded and the procedure always terminates.

2.3 Recurrent Sets

A *recurrent set* [100, 49] for a relation $\rho(V, V')$ is a set of states that is visited infinitely often by some sequence in ρ. Recurrent sets can be thought of as the dual of ranking functions for well-founded relations. Ranking functions are witnesses for the well-foundedness of a relation, instead recurrent sets are witnesses for the *non* well-foundedness of a relation. In fact, in the following we will show that there exists a recurrent set for a relation ρ iff ρ is not well-founded, i.e. it admits at least one infinite chain [100].

In the following we distinguish two types of recurrent sets (open and closed) and outline their relationship.

Definition 1 - Open Recurrent Set
$R(V)$ is a open recurrent set for a relation $\rho(V, V')$ iff the following hold.

- $\exists V : R(V)$;
- $\forall V, \exists V' : R(V) \rightarrow \rho(V, V') \wedge R(V')$.

$R(V)$ represents a nonempty set of states such that every state in R admits a successor via the relation ρ that is still in the set R. It is easy to show (e.g. by induction) that there exists an open recurrent set for a binary relation iff the relation is not well-founded, i.e. it admits at least one infinite sequence [100]. Notice that states in an open recurrent set might have successors outside the set.

A *closed recurrent set* or *universal recurrent set* [17], proves that a relation ρ is not well-founded by identifying an underapproximation T of ρ with respect to a set R such that T is left-total in R and R is closed with respect to T.

Definition 2 - Closed Recurrent Set
A pair $\langle R(V), T(V,V')\rangle$ *is a closed recurrent set for a relation* $\rho(V,V')$ *iff the following hold.*

- $\exists V : R(V)$*;*
- $\forall V, \exists V' : R(V) \wedge T(V,V')$*;*
- $\forall V, V' : R(V) \wedge T(V,V') \rightarrow \rho(V,V') \wedge R(V')$.

Notice that this definition is slightly different from the one of [49] since we embed the underapproximation within the definition of closed recurrent set. Let ρ be a relation, then it admits an open recurrent set iff it admits also a closed recurrent set [66]. The existence of a closed recurrent set trivially implies the existence of an open one. In fact, the set R of every closed recurrent set is an open recurrent set. Viceversa, from an open recurrent set we can construct a corresponding closed one by defining the underapproximation $T(V,V')$ as $\rho(V,V') \wedge R(V')$.

Therefore, a relation $\rho(V,V')$ is not well-founded iff it admits an open or closed recurrent set.

Example 7 - Recurrent Set
Consider a relation $\rho(V,V') \doteq x > 0 \wedge y > 0 \wedge x' > x \cdot y$. An open recurrent set for ρ is $R(V) \doteq x > 1 \wedge y > 1$. A closed recurrent set for ρ is $\langle R(V), T(V,V')\rangle$ where: $R(V)$ is defined as above and $T(V,V') \doteq y' = y \wedge x' = x \cdot y + 1$.

2.4 Constrained Horn-like Clauses

Constrained Horn-like Clauses (CHC) [98, 27, 101] are a fragment of FOL that has been proposed as a formalism to represent many verification problems in a solver-independent way. It allows the separation of the development of a proof methodology and the algorithms and procedures used to solve the verification problem.

A CHCs problem is a sequence of logical implications and wf-predicates over a set of symbols V and a set of uninterpreted predicate symbols $\mathcal{Q}$. wf, for well-founded, is an interpreted unary predicate that holds for a relation T over $V \cup V'$ iff T is well-founded [1]. Let h range over queries and quantifier-free formulae over $V \cup V'$, let c be a quantifier-free formula over $V \cup V'$ and $\{q_i\}_{i=1}^{n}$ be queries over $V \cup V'$. Then, a Horn-like clause must be one of the following two kinds of formulae:

(i) an implication of the form

$$c \wedge \bigwedge_{i=1}^{n} q_i \rightarrow h,$$

(ii) a wf-predicate (a unit clause) applied to some query symbol q_j for some $0 < j \leq n$

$$wf(q_j),$$

such that each query symbol appears at most once in every clause. A solution for a CHCs problem is an interpretation for the query symbols in $\mathcal{Q}$ such that all clauses are valid (all clauses are true for all assignments to the symbols in $V \cup V'$) and for each wf-predicate $wf(q)$ the interpretation assigns some well-founded relation to q.

Notice that CHCs have a much richer syntax with respect to Horn clauses. In fact, they allow arbitrary formulae on both sides of the im-

[1] We use wf instead of dwf, for disjunctively well-founded, defined e.g. in [98], since in our context the type of the well-founded relation is not important.

plication and only require each clause to contain at most one non-negated query symbol. Unfortunately, this expressiveness makes the satisfiability problem of CHCs using the theory of arithmetic undecidable [101].

Example 8 - CHC

Consider the simple imperative procedure reported in Fig. 2.1. Let $V \doteq \{q, y\}$ and $V' \doteq \{q', y'\}$ be the sets of variables and $\mathcal{Q} \doteq \{T(V, V')\}$ be the set of uninterpreted predicates symbols. The existence of a solution for the following CHC problem implies that the software program always terminates.

$$
\begin{aligned}
& q > 0 \wedge y' = y \wedge y > 0 \wedge q' = q - y - 1 \rightarrow T(V, V'); \\
& q > 0 \wedge y' = y \wedge y \leq 0 \wedge q' = q + y - 1 \rightarrow T(V, V'); \\
& wf(T).
\end{aligned}
$$

The two implications make sure that any pair of states corresponding to a single iteration through the loop is in relation T. Therefore, any interpretation of T must be an overapproximation for the procedure. The wf-predicate requires such overapproximation to be well-founded. Any relation containing a subset of the elements of a well-founded relation is also well-founded. Therefore, if there exists an interpretation for T satisfying the constraints above, then the software program must be terminating.

2.4.1 Existentially-Quantified Constrained Horn-like Clauses

Existentially-quantified Constrained Horn-like Clauses (E-CHCs) [28] are an extension of CHCs that allow the use of existential quantifiers in the right-hand-side of the implications, while all other symbols remain implicitly universally quantified as in CHCs. This extension allows the use of a formalism with the same advantages (separation between proof methodology and algorithms) and very similar notation to CHCs to deal with existential verification problems. However, since E-CHCs are strictly more

expressive than CHCs, all undecidability results of the latter apply also to the former and E-CHCs are undecidable if we consider the arithmetic theory [101].

Example 9 - E-CHC

Consider again procedure reported in Fig. 2.1. We define a E-CHC problem that admits a solution only if the procedure does not terminate. As in the previous examples we define the sets of variables $V \doteq \{q, y\}$ and $V' \doteq \{q', y'\}$. Let $\mathcal{Q} \doteq \{R(V), T(V, V')\}$ be two uninterpreted predicates. We want to identify an underapproximation for the procedure that contains at least one and only nonterminating executions. We use R to represent the set of states that can appear in such executions and T must restrict the transition relation of the program by pruning all terminating executions.

$$
\begin{aligned}
R(V) \wedge T(V, V') &\rightarrow q > 0 \wedge y' = y \wedge y > 0 \wedge q' = q - y - 1; \\
R(V) \wedge T(V, V') &\rightarrow q > 0 \wedge y' = y \wedge y \leq 0 \wedge q' = q + y - 1; \\
T(V, V') &\rightarrow R(V'); \\
\top &\rightarrow \exists V : R(V); \\
R(V) &\rightarrow \exists V' : T(V, V').
\end{aligned}
$$

The first two implications ensure that every T transition starting from some state in R is also a transition for the software program: it is an underapproximation. The third implication ensures that R is closed with respect T, T cannot exit from R. However, these first three implications could be satisfied by simply interpreting both R and T as the constant $\bot$, which, of course, does not correspond to any nonterminating run of the program. The last two implication use the existential quantifiers to ensure that the set R is not empty and that the relation T is left-total: does not have any deadlock in R.

Therefore, if there is an interpretation for R and T such that all of the above are true, then there is at least one nonterminating execution of the

software program. In fact, given an interpretation such an execution can be computed as follows. Pick a state $\boldsymbol{v}_0$ in R, one such state must exist since R is not empty. Every state in R must have some successor via the relation T (no deadlocks), hence there must be at least one state $\boldsymbol{v}_1$ such that $\boldsymbol{v}_0, \boldsymbol{v}'_1 \models T$, where $\boldsymbol{v}'_1$ assigns to every $v' \in V'$ the value assigned by $\boldsymbol{v}_1$ to the corresponding $v \in V$. R is closed with respect to T, hence $\boldsymbol{v}_1$ is still in R. We can apply the same reasoning infinitely many times, every state $\boldsymbol{v}_i$ in R must admit a T-successor $\boldsymbol{v}_{i+1}$ that is still in R. By induction, we can obtain an infinite sequence of R-states by adding an unbounded number of T-steps. Therefore, every pair of consecutive states in the sequence satisfies $R(V) \wedge T(V, V')$; the first two implication must hold and imply that every such step is also a step for the procedure. Therefore, the infinite sequence of states is a infinite (i.e. nonterminating) execution of the program.

2.5 Fair Transition Systems

In this thesis we formally describe a system as a *fair transition system* [15] and call *model* a formal description of the system. A transition system can be visualised as a directed graph in which the nodes represent the states of the system and the edges are its transitions. A transition system is *finite* (FTS) if such graph has a finite number of nodes and *infinite* (ITS) otherwise. The *initial states* of the transition system are the nodes that represent all starting configurations of the model, while the *transitions* describe how the system moves from one state to another. Every path on the graph starting from an initial state represents a possible execution of the system (also called path or trace) and it is uniquely identified by a possibly infinite sequence of states. States with no outgoing edges are called *deadlock states*, hence every deadlock state is either unreachable or the last state of some finite path. From an automata-theoretic point of

view it is possible to borrow a slightly different terminology. The set of all possible paths of the transition system is called language of the automata and every trace is a word. Two transition systems M_0, M_1 are *equivalent* if they accept the same language, written $M_0 \equiv M_1 \iff \mathcal{L}(M_0) = \mathcal{L}(M_1)$, where $\mathcal{L}(M)$ is the language of model M. A transition system is *fair* if it denotes some of its states as *fair states* and its language is restricted such that all infinite paths visit the fair states infinitely often.

There are many different representations of transition systems. They are slightly different notations to represent these objects and adopt different acceptance conditions. The representations can be broadly classified in two categories: *explicit-state* and *symbolic*. Explicit-state representations (e.g. Kripke structures, Büchi automata and Rabin automata) rely on the graph representation of the model and implement operations and procedures as visits on such graph. Symbolic representations, instead, describe the system using first-order formulae. In this setting each model of the formula corresponds to a state and a formula denotes the set of states given by the set of all its models. Therefore, the logical operators $\neg$, $\vee$ and $\wedge$ in the symbolic setting correspond, respectively, to the set operators complement, union and intersection in explicit-state representations. This thesis is concerned with symbolic techniques and we represent fair transition systems symbolically as follows.

Definition 3 - Generalised Fair Transition System
Given a first-order signature Σ and a set of variables $\mathcal{V}$, let $\mathcal{A} \doteq \{a \mid a \in \Sigma$ and a is a 0-ary predicate$\}$ be the set of Boolean atoms of Σ. A symbolic fair transition system M is a 4-tuple $\langle V, I(V), T(V, V'), \mathcal{F} \rangle$ such that:

- *$V \subseteq \mathcal{V} \cup \mathcal{A}$ is the set of state variables;*
- *$I(V)$ is a quantifier-free formula over V denoting the initial states;*
- *$T(V, V')$ is a quantifier-free formula over $V \cup V'$ and denotes the tran-*

sitions;

- $\mathcal{F}$ *is a finite set of quantifier-free formulae over* V*, each of which denotes a fairness condition.*

The transition system is finite iff all symbols in V have finite domain, infinite otherwise. When the fairness conditions are not relevant we will simply talk about transition systems defined as the triple $\langle V, I, T\rangle$, that correpond to the generalised fair transition system $\langle V, I, T, \{\top\}\rangle$. We can now formally define a deadlock state $\boldsymbol{v}$ for a transition system $M \doteq \langle V, I, T\rangle$ as a state such that $\boldsymbol{v} \models \forall V' : \neg T(V, V')$.

Definition 4 - Path, Trace, Execution
A finite or infinite sequence of states $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots$ *is a path (also called trace or execution) of a generalised fair transition system* $M \doteq \langle V, I, T, \mathcal{F}\rangle$ *iff:*

- $\boldsymbol{v}_0 \models I$*;*
- $\forall i \geq 0 : (\boldsymbol{v}_i, \boldsymbol{v}'_{i+1} \models T)$*;*

where $\boldsymbol{v}'_i$ *assigns every* $v' \in V'$ *to the value assigned by* $\boldsymbol{v}_i$ *to the corresponding* $v \in V$*.*

Given a path $\pi \doteq \boldsymbol{v}_0, \boldsymbol{v}_1, \ldots$ and a set of symbols V_0 we call projection of π over V_0 the path $\pi_{\downarrow V_0} \doteq \boldsymbol{v}_{0_{\downarrow V_0}}, \boldsymbol{v}_{1_{\downarrow V_0}}, \ldots$, given by the projection of each of its states. A state $\boldsymbol{v}$ is *reachable* in M iff there is a finite path of M ending in $\boldsymbol{v}$, written $M \rightsquigarrow \boldsymbol{v}$. With a slight abuse of notation, given a formula $\phi(V)$ we write $M \rightsquigarrow \phi$ iff M can reach a state in ϕ, hence there exists a state $\boldsymbol{v}$ such that $M \rightsquigarrow \boldsymbol{v}$ and $\boldsymbol{v} \models \phi$.

Definition 5 - Fair Path
A path $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots$ *of a generalised fair transition system* $M \doteq \langle V, I, T, \mathcal{F}\rangle$ *is fair iff for every* $F_i \in \mathcal{F}$ *and* $j \geq 0$*, there exists* $h > j$ *such that* $\boldsymbol{v}_h \models F_i$

Note that from the definition of fair paths it immediately follows that a fair path must also be an infinite path.

Definition 6 - Language of Generalised Fair Transition System
Given a fair transition system M, its language is defined as

$$\mathcal{L}(M) \doteq \{\pi \mid \pi \text{ is a fair path of } M\}.$$

We write $\mathcal{L}_{\downarrow V_0}(M) \doteq \{\pi_{\downarrow V_0} \mid \pi \in \mathcal{L}(M)\}$ for a set of symbols V_0 for the set of paths of M projected over V_0.

The synchronous composition operator for generalised fair transition systems computes a transition system M from two input systems M_0 and M_1 such that the language of M is the intersection of the languages of M_0 and M_1: $\mathcal{L}(M) = \mathcal{L}(M_0) \cap \mathcal{L}(M_1)$.

Definition 7 - Synchronous Composition
Given two generalised fair transition systems $M_0 \doteq \langle V_0, I_0, T_0, \mathcal{F}_0 \rangle$ and $M_1 \doteq \langle V_1, I_1, T_1, \mathcal{F}_1 \rangle$ their synchronous composition $M \doteq M_0 \times M_1$ is a transitions system $M \doteq \langle V, I, T, \mathcal{F} \rangle$ such that: (i) $V \doteq V_0 \cup V_1$; (ii) $I \doteq I_0 \wedge I_1$; (iii) $T \doteq T_0 \wedge T_1$; (iv) $\mathcal{F} \doteq \mathcal{F}_0 \cup \mathcal{F}_1$.

This operator is employed in many transformations in model checking. Another relevant transformation is the *degeneralisation*. Given a generalised fair transition system M_0 the transformation computes another generalised fair transition system M such that M has a single fairness condition and the set of paths in the language of M projected over the symbols of M_0 is exactly the language of M_0.

Definition 8 - Degeneralisation
Given a fair transition system $M_0 \doteq \langle V_0, I_0, T_0, \mathcal{F}_0 \rangle$ with $\mathcal{F}_0 \doteq \{F_i\}_{i=0}^{n-1}$, let $M \doteq \langle V, I, T, \mathcal{F} \rangle$ such that:

- $V \doteq V_0 \cup \{c\}$ *where c is a fresh variable with domain $\{i\}_{i=0}^{n-1}$;*
- $I \doteq I_0 \wedge c = 0$;
- $T \doteq T_0 \wedge ((c = n-1 \wedge F_{n-1}) \rightarrow c' = 0) \wedge (\bigwedge_{i=0}^{n-2} (F_i \wedge c = i) \rightarrow c' = i+1)$;

- $\mathcal{F} \doteq \{c = n - 1 \wedge F_{n-1}\}$.

M_0 and M are such that $\mathcal{L}(M_0) = \mathcal{L}_{\downarrow V_0}(M)$ holds. We will call a degeneralised fair transition system M simply fair transition system and write $M \doteq \langle V, I, T, F \rangle$, where F is the quantifier-free formula representing its only fairness condition.

2.6 Timed Fair Transition Systems

In transition systems changes happen atomically and the evolution of the model is given by a sequence of discrete steps. We refer to such systems as *discrete* or *untimed* transition systems. In some cases this is not sufficient and the modelling of a system requires the capability to express different behaviours depending on some notion of time. Examples of this are systems involving timers, timeouts or synchronisation based on shared clocks.

Timed transition systems (TTS) have a built-in notion of *time*, for this reason they are a more natural formalism in which to describe time-aware systems. Paths of timed systems move along two orthogonal directions. One direction represents the passage of time, while the other represents the atomic steps performed by the system. A possible visualisation for a path of a timed system is given in Fig. 2.2, which also highlights that "time" can only move forward. In more detail, timed fair transition systems extend fair transition systems with a new type of symbols, called *clocks*, that keep track of the elapse of time. A transition is either a time elapse (or *timed transition*) or an atomic (also called *instantaneous*) transition. In timed transitions, all discrete (non-clock) symbols retain their values while all clocks increase by the same amount, equal to the elapsed time. In this setting "time" is a clock symbol that is initially 0. As all other clocks, also the evaluation of "time" remains constant in discrete transitions and increases by a positive amount in every time elapse. The domain of the

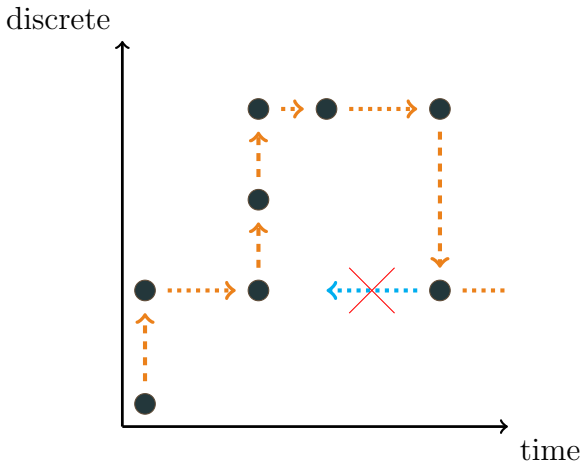

Figure 2.2: Representation of a path of a timed system along the two directions.

clock symbols can be either a dense domain, such as $\mathbb{R}_0^+$, or a discrete domain, such as $\mathbb{N}$. We will explicitly state their domain when a particular result applies only to one of the two cases, we will otherwise simply talk about "clocks" without specifying their domain.

In the following, we first introduce two important concepts for the verification of timed systems and then formally define the models of time we consider in this thesis.

Zeno paths. A natural assumption used when modelling timed systems is that every infinite path will eventually reach any point in the future, hence that time cannot be halted. This is an important assumption that needs to be considered in modelling and reasoning on the system. A popular example to illustrate and highlight the relevance of this assumption is Zeno's paradox about Achilles and the tortoise. In the paradox Achilles is trying to reach and overtake a tortoise in a footrace. For simplicity's sake, assume that both of them have constant velocity and that the speed of the tortoise is half the speed of Achilles. The reasoning leading to the paradox is as follows. Achilles, before overtaking the tortoise, must reach the current position of the animal. However, in the time it takes him to do

that the tortoise will move forward of half their previous distance. Therefore, Achilles must now cover half of the initial distance and after that the tortoise will be in front of him by a quarter of the original distance. We can repeat this process any number of times and Achilles will keep halving his distance from the tortoise. However, for any number of iterations the distance will be greater than zero. This process is depicted in Fig. 2.3 and the paradox states that this reasoning proves that Achilles will keep getting closer to the tortoise but he will never reach nor overtake the animal. However, the infinite path described by the paradox has an upper bound to the total amount of elapsed time. Achilles and the tortoise are moving at a constant speed, hence the distance they travel for each unit of time is constant. This implies that the time spent in every iteration also keeps halving. If the time required by Achilles to cover the initial distance is δ, we can write the infinite sum of the time spent in every iteration as $\sum_{i=0}^{+\infty} \frac{\delta}{2^i}$. The series converges to 2δ, hence 2δ is an upper bound for the amount of time that can elapse in the system. Therefore, an infinite number of such iterations happens in a finite amount of time.

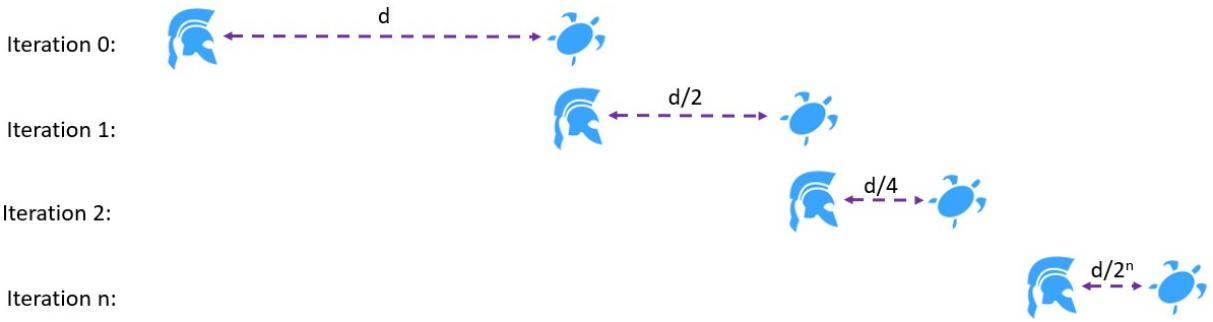

Figure 2.3: At every iteration Achilles halves the distance from the tortoise.

In the literature executions that do not allow progress with respect to time are known as *Zeno paths*. We call a infinite path of a timed system Zeno iff there exists an upper bound for the value of time in the path and non-Zeno otherwise. In this setting, when dealing with infinite paths, we

want to consider only those that are non-Zeno. If we consider a discrete domain of time (e.g. $\mathbb{N}$), this simply requires time to increase infinitely often in the path. However, on dense domains of time (e.g. $\mathbb{R}_0^+$) this is not sufficient. It could be the case that time increases infinitely often but it still has some constant upper bound. This happens when the sequence of time elapses in the path describes or is bounded from above by a converging series (e.g. the geometric series $\sum_{n=1}^{+\infty} \frac{1}{2^n} < 1$).

Finite variability. Another relevant concept for timed systems is that of *finite variability*. A path of a timed system could prescribe an increasing number of actions between time elapses and there could be no upper bound on the number of such actions. Therefore, in the limit, there is an infinite sequence of states between any consecutive time elapses. Finite variability requires the existence of an upper bound to the number of changes that can happen in any finite interval of time. The paths that do not have this property represent behaviours that can arise in the mathematical description of the system, but often cannot occur in the system itself (e.g. no computing machine can perform an infinite number of operations in 0 time). For this reason, timed systems are often analysed under the finite variability assumption and their language is restricted to only the paths that satisfy the finite variability property.

We say that a trace is *fine* with respect to a formula φ in a time interval iff the truth assignment of φ is constant in the interval, i.e. all time points in the interval agree on the truth assignment of φ. A trace has the finite variability property iff for every formula φ there exist a sequence of time points such that the path is fine in every pair of consecutive time points.

2.6.1 Time Models

We define a time interval as a convex subset of either $\mathbb{N}$ or $\mathbb{R}_0^+$. We talk about discrete intervals in the first case and dense intervals in the latter. We generically talk about intervals, without specifying whether it is discrete or dense, when the type is not important or clear from the context. For $t_l, t_r \in \mathbb{N}$ or $t_l, t_r \in \mathbb{R}_0^+$ such that $t_l \leq t_r$ we write an interval as $[t_l, t_r]$, $(t_l, t_r]$, $[t_l, t_r)$ or (t_l, t_r) where the square bracket is used to indicate that the element is included in the interval and the parenthesis to indicate that it is excluded. Given an interval I we write $l(I)$ and $r(I)$ for the left and right endpoints of the interval respectively. Two intervals I_0 and I_1 are *almost adjacent* iff $r(I_0) = l(I_1)$, hence iff they overlap in at most one time point. We say that an interval I is *singular* iff it contains a single time point, $I \doteq [a, a]$. Finally, a *time interval sequence* is a sequence $I_0, I_1, \ldots$ of time intervals of the same type (discrete or dense) such that I_i and I_{i+1} are almost adjacent for all $i \geq 0$ and $\bigcup_{i=0}^{+\infty} I_i$ is either $\mathbb{N}$ or $\mathbb{R}_0^+$ depending on the intervals type.

We define time models in the style of [7] and [8].

Definition 9 - Time Model
A time model is 4-tuple $\tau \doteq \langle \mathbb{T}, <, \mathbf{0}, v \rangle$*, where:*

- $\mathbb{T}$ *is the temporal domain;*
- $<$ *is a total order over* $\mathbb{T}$*;*
- $\mathbf{0} \in \mathbb{T}$ *is the minimal element of* $<$*;*
- $v : \mathbb{T} \mapsto \mathbb{R}_0^+$ *is a function that represents the real time of a time point in* $\mathbb{T}$ *such that any infinite sequence of time points* $\{t_i \in \tau\}_{i=1}^{+\infty}$ *such that* $\forall i \in \mathbb{N} : t_i < t_{i+1}$*,* $\{v(t_i)\}_{i=1}^{+\infty}$ *is a nondecreasing divergent sequence.*

The fact that v maps sequences of time points into nondecreasing diver-

gent sequences on $\mathbb{R}_0^+$ forbids sequences of time points that move backward in "time" (nondecreasing) and also excludes all Zeno paths (divergent).

We define four different time models: *discrete*, *dense*, *super-dense* and *super-discrete*.

Discrete Time Model

Intuitively, the discrete time models describe uni-dimensional sequences of time-points, hence strictly monotonic sequences of time points over the $\mathbb{N}$. "Untimed" systems and specifications over them use this model of time and in these cases executions are discrete sequences of states.

Definition 10 - Discrete Time Model

A time model $\langle \mathbb{T}, <, \mathbf{0}, v \rangle$ *is discrete iff*

- $\mathbb{T} \doteq \mathbb{N}$*;*
- $<$ *is the standard less-then relation over the natural numbers;*
- $\mathbf{0}$ *is the natural number zero;*
- v *is a monotonically increasing function.*

Dense Time Model

The dense time model corresponds to a strictly monotonic sequence of time points over the $\mathbb{R}_0^+$. While, in discrete time model every time interval admits at most some countable number of time points, in dense time model this is not the case and there can be an uncountable number of time points in every non-empty interval.

Definition 11 - Dense Time Model

A time model $\langle \mathbb{T}, <, \mathbf{0}, v \rangle$ *is dense iff:*

- $\mathbb{T} \doteq \mathbb{R}_0^+$*;*
- $<$ *is the standard less-then relation over the real numbers;*

- 0 *is the real number zero;*
- v *is the identity function.*

Super-Dense Time Model

The super-dense time models are defined over pairs of $\mathbb{N}\times\mathbb{R}_0^+$. They consist of alternating discrete sequences and dense set. The discrete sequences are countable sequences of time points in which the discrete part increases while the continuous one remains constant to some time stamp $r \in \mathbb{R}_0^+$; i.e. of the form $\{\langle i, r\rangle\}, \{\langle i+1, r\rangle, \ldots$. The dense sets, instead, correspond to an uncountable number of time points with different time stamps, but constant discrete counter $i \in \mathbb{N}$; i.e. of the form $\langle i, r\rangle, \langle i, r'\rangle, \ldots$. This model of time corresponds to the one we informally described above and represented in Fig. 2.2 in the case of "time" with dense domain.

Definition 12 - Super-Dense Time Model
A time model $\langle \mathbb{T}, <, \mathbf{0}, v\rangle$ *is super-dense iff:*

- $\mathbb{T} \subset \mathbb{N} \times \mathbb{R}_0^+$ *such that the sequence of sets* $I_0, I_1, \ldots$, *where, for all* $i \geq 0$, $I_i \doteq \{r \mid \langle i, r\rangle \in \mathbb{T}\}$ *is a time interval sequence;*
- $< \doteq \{\langle\langle i, r\rangle, \langle i', r'\rangle\rangle \mid \langle i, r\rangle, \langle i', r'\rangle \in \mathbb{T} \wedge (i < i' \vee (i = i' \wedge r < r'))\}$*;*
- $\mathbf{0} \doteq \langle 0, 0\rangle \in \mathbb{T}$*;*
- $v(\langle i, r\rangle) = r$.

Notice that any time interval sequence in a super-dense time model such that the intervals are not overlapping is isomorphic to the dense time model.

Super-Discrete Time Model

Finally, the super-discrete time models are defined over pairs of $\mathbb{N} \times \mathbb{N}$. Similarly to the super-dense case, the first component is used to represent

the instantaneous steps while the second component describes the time dimension. However, in this case they both correspond to countable sequences of time points and the super-discrete time models correspond to the one we informally described above and represented in Fig. 2.2 in the case of "time" with discrete domain.

Definition 13 - Super-Discrete Time Model
A time model $\langle \mathbb{T}, <, \mathbf{0}, v \rangle$ *is super-dense iff:*

- $\mathbb{T} \subset \mathbb{N} \times \mathbb{N}$ *such that the sequence of sets* $I_0, I_1, \ldots$, *where, for all* $i \geq 0$, $I_i \doteq \{r \mid \langle i, r \rangle \in \mathbb{T}\}$ *is a time interval sequence;*
- $< \doteq \{\langle \langle i, r \rangle, \langle i', r' \rangle \rangle \mid \langle i, r \rangle, \langle i', r' \rangle \in \mathbb{T} \wedge (i < i' \vee (i = i' \wedge r < r'))\}$;
- $\mathbf{0} \doteq \langle 0, 0 \rangle \in \mathbb{T}$,
- $v(\langle i, r \rangle) = r$.

Path, Trace, Execution

Def. 4 defines a path for a transition system as a discrete sequence of states. In fact, transition systems are interpreted over the discrete time model such that the i^{th} state is associated with time point i, i.e. $v(i) = i$. However, traces might prescribe a different mapping from time points to time value (the function v) and when considering the dense or super-dense time models we need to represent paths as dense sequences of states. For this reason, in the following we consider a more general representation of traces that is suitable to represent all four types of time models and, in particular, to represent paths of timed systems.

Definition 14 - Path, Trace, Execution
Given a set of symbols V *a trace is a triple* $\sigma \doteq \langle M, \tau, \overline{\mu} \rangle$ *such that* M *is a first-order structure,* V^M *is the set of states with first-order structure* M, τ *is the time model and* $\overline{\mu}$ *is a mapping from the domain of* τ *into* V^M.

Given a time point $t \in \tau$, we denote with $\sigma(t)$ the state $\langle M, \overline{\mu}(t) \rangle$. The first-order structure M is shared by all time-points. Therefore, all time points have the same interpretation for the symbols in Σ. However, the interpretation of such symbols may not be unique in the background theory. We call these "uninterpreted" symbols *parameters*. A parameter is a constant symbol in the signature σ such that its value is not determined by the theory and does not vary with time.

In this context, when considering super-dense and super-discrete time models, the instantaneous transitions are the ones in which the first component of the time-model increases, while the timed transitions are the ones in which the second component increases.

Finally, in the case of discrete time model, fixed the first-order structure M, Def. 14 and Def. 4 are equivalent.

2.6.2 Timed Automata

A timed automaton (TA) [5] is a TTS that can be represented by a finite graph in which nodes are called *locations* and edges represent instantaneous transitions. In timed automata the clock variables have domain $\mathbb{R}_0^+$. All clock symbols are initialised to 0 and time can elapse inside locations constrained by some *location invariant*. Every instantaneous transition between locations is associated with a *guard* condition and a set of clocks called *reset*. A timed automaton can perform an instantaneous transition only if it satisfies the guards of the edge and when such transition is performed all clocks specified in its reset are set to 0. In addition, it can enter or remain in a location only as long as the associated location invariant is satisfied. Therefore, location invariants impose progress conditions on the automaton, while guards and resets on the edges constrain its behaviour. Given a set of clocks X, a clock variable $x \in X$ and a rational constant $c \in \mathbb{Q}$, each guard and location invariant $\phi(X)$ must be a quantifier-free

formula over the symbols X in the language $\mathcal{B}(X)$ defined as follows:

$$\phi :: x \leq c \mid c \leq x \mid x < c \mid c < x \mid \phi_0 \wedge \phi_1$$

Therefore, clocks constraints in timed automata are limited to conjunctions of clocks compared with constants.

In the following we propose a definition of timed automata that slightly differs from the one of [5]; we do not consider labels on the edges. We do this in order to simplify the presentation and without loss of generality. In terms of the semantics of the timed automaton as a transition system, the labels can be modelled by considering an additional state symbol that prescribes the label of the next transition.

Definition 15 - Timed Automaton
A timed automaton M is a 5-tuple $\langle L, L_0, X, I, E \rangle$ such that:

- *L is a finite set of locations;*
- *$L_0 \subseteq L$ is the set of initial locations;*
- *X is the set of clock symbols;*
- *$I : L \mapsto \mathcal{B}(X)$ is a function that maps each location to its corresponding location invariant;*
- *$E \subseteq L \times \mathcal{B}(X) \times 2^X \times L$ is the set of edges, each edge has a starting location, a guard, a set of clocks to be reset and a target location.*

For brevity we write $l \xrightarrow{g,r} l'$ for $(l, g, r, l') \in E$.

A *clock interpretation* μ is a function that maps every clock to a real value, $\mu : X \mapsto \mathbb{R}^{|X|}$. We write $\mu + \delta$, with $\delta \in \mathbb{R}_0^+$, for the interpretation that assigns to each clock the value given by μ plus a positive real constant δ.

$$\forall x \in X : (\mu + \delta)(x) = \mu(x) + \delta.$$

In addition, given a set of clocks r and a clock interpretation μ, we write $[r \mapsto 0]\mu$ for the interpretation that assigns 0 to all the clocks in r and agrees with μ on all other assignments.

$$[r \mapsto 0]\mu(x) \doteq \begin{cases} 0 & \text{if } x \in r, \\ \mu(x) & \text{otherwise.} \end{cases}$$

The semantics of a timed automaton is defined in terms of the corresponding transition system defined over symbols $\hat{X} \cup \{\hat{l}\}$, where $\hat{X}$ contains a real-valued symbol for every clock in X and $\hat{l}$ is a symbol not in $\hat{X}$ with domain L. Therefore, each state of the timed automaton is a pair $\langle l, \mu \rangle$ such that $l \in L$ prescribes the assignment to $\hat{l}$ and μ is a clock interpretation for X that corresponds to a total assignment over $\hat{X}$. We write $\langle l, \mu \rangle \xrightarrow{\delta} \langle l, \mu + \delta \rangle$ for the elapse of $\delta \in \mathbb{R}_0^+$ time in location l and $\langle l, \mu \rangle \rightarrow \langle l', \mu' \rangle$ for the instantaneous transition from $\langle l, \mu \rangle$ to $\langle l', \mu' \rangle$, where $\mu' \doteq [r \mapsto 0]\mu$ and r is the reset of the transition. The transitions are defined as follows:

- $\langle l, \mu \rangle \xrightarrow{\delta} \langle l, \mu + d \rangle \iff (\mu \models I(l)) \wedge (\mu + \delta \models I(l))$, for $\delta \in \mathbb{R}_0^+$;
- $\langle l, \mu \rangle \rightarrow \langle l', \mu' \rangle \iff \exists (l, g, r, l') \in E : (\mu \models g) \wedge (\mu' = [r \mapsto 0]\mu) \wedge (\mu' \models I(l'))$.

It is possible to perform a time elapse of $\delta \in \mathbb{R}_0^+$ time units in location l if the clock interpretations at the beginning and end of the transition (μ and $\mu + \delta$) both satisfy the location invariant $I(l)$ of l. Notice that $I(l)$ must be a convex formula, since $\mathcal{B}(X)$ allows only for convex clock constraints. Therefore, from $\mu \models I(l)$ and $\mu + \delta \models I(l)$ we can conclude that for any $0 \leq \delta_0 \leq \delta$, $\mu + \delta_0 \models I(l)$ and also any intermediate state satisfies the location invariant. In a timed automaton it is possible to perform an instantaneous transition from state $\langle \mu, l \rangle$ to state $\langle \mu', l' \rangle$ iff (i) there is an edge in E from l to l', (ii) the current clock interpretation μ satisfies the guard of the transition, (iii) the clock interpretation μ' is equal

to $[r \mapsto 0]\mu$, where r is the reset of the transition, and finally (iv) μ' satisfies the location invariant $I(l')$ of l'.

Example 10 - Timed Automaton

Fig. 2.4 shows a timed automaton with a single clock symbol c and two locations l_0 and l_1. Each location has a location invariant: $c \leq 5$ for l_0 and $c < 15$ in location l_1. The automaton has two instantaneous transitions, one from l_0 to l_1 with guard $c \geq 5$ and reset $\{c\}$ and the other from l_1 to l_0 with guard $c > 3$ and empty reset. l_0 is the initial location, hence the initial state is $\langle l_0, \mu_0 \rangle$ where μ_0 assigns c to 0. From the semantics described above, the timed automaton in Fig. 2.4 follows the following behaviour. It

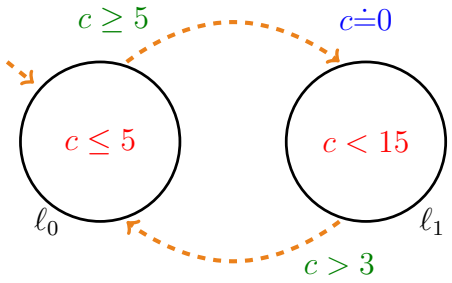

Figure 2.4: Timed automaton, location invariants are highlighted in red, guards in green and resets in blue.

is possible to enter location l_0 only if $c \leq 5$. The only outgoing transition from l_0 has guard $c \geq 5$, hence we must remain in l_0 until $c = 5$ and must take the transition to l_1 exactly at that point. The transition resets c to 0, hence the location invariant of l_1 is satisfied. The only outgoing transition from l_1 has guard $c > 3$, for this reason we have to remain in l_1 for at least (strictly more) 3 time units. The location invariant requires $c < 15$, hence we could wait up to 15 (excluded) time units in l_1. However, the only outgoing transition leads to l_0 that requires $c \leq 5$ and the transition does not reset c. For this reason, if we wait more that 5 time units in l_1, then the location invariant would forbid us to perform the transition to l_0. In this case we must remain in l_1 constrained by its location invariant and, while

the remaining available paths are infinitely many, they are either Zeno (c gets closer and closer to 15 without reaching it) or finite.

Timed automata [5] allow clocks only in comparisons with rational numeric constants. Later works extend them by allowing differences between clocks, e.g. $c_0 - c_1 < 5$. These constraints are called *diagonal constraints* and we talk about *diagonal timed automata*. Diagonal timed automata are as expressive as timed automata [25], but they can be exponentially more succinct [38]. In addition, in many practical cases the system to be verified can be modelled as a timed automaton without the need for diagonal constraints [36].

2.6.3 Hybrid Systems

Hybrid automata have been introduced in 1996 by Henzinger [106] as a formalism to represent cyber-physical systems (CPS). CPS are dynamical systems with both discrete and continuous components. The discrete component, described by the set of locations or modes, can be used to model some electronic controller, while the continuous component, described via *continuous* symbols and *flow conditions*, can define a mathematical model for some physical phenomena. For example, a thermostat and a braking system of a vehicle can be modelled as a CPS where the locations represent the state of the controller and the physical system is represented via the continuous variables, e.g. temperature and velocity. Usually hybrid systems (HS) are described assuming a dense domain of time. However, in some cases a discrete domain is used as an approximation of the dense counterpart.

Continuous symbols in hybrid automata are symbols whose assignment during time elapses is described by flow conditions. Flow conditions describe the evolution over time of continuous symbols and can be expressed

in three different forms: (i) as constraints over the derivatives of continuous values, (ii) as ordinary differential equations (ODE), (iii) as explicit functions of time without derivatives. Continuous symbols generalise clocks, clocks are continuous symbols with constant derivative equal to 1. Given a set of continuous symbols X we denote with $\dot{X} \doteq \{\dot{x} \mid x \in X\}$ the set of their first derivatives with respect to time. A hybrid automaton is said to be *initialised* if every time the flow of some variable changes, then the variable is reset.

We report a definition of hybrid automata that slightly differs from the one of [106]. As for the timed automata case, we do not consider labels on the edges (events), but they can be encoded via an additional state variable.

Definition 16 - Hybrid Automaton
A hybrid automaton M is a 6-tuple $\langle L, L_0, X, I, Flow, E\rangle$ such that:

- *L is a finite set of locations (also called modes);*
- *L_0 is a function that maps each location to a formula over X;*
- *X is a finite set of continuous symbols;*
- *L_0, I and $Flow$ are functions that map each location to a formula over X (L_0 and I) and $X \cup \dot{X}$ ($Flow$);*
- *E is the set of transitions. Each transition $\langle l, g, r, l'\rangle \in E$ is such that: (i) $l, l' \in L$ are the source and destination location of the transition, (ii) g is a formula over X representing the guard of the transition and (iii) r is a formula over $X \cup X'$ representing the reset conditions.*

For brevity we write $l \xrightarrow{g,r} l'$ for $(l, g, r, l') \in E$.

An interpretation for the continuous symbols μ is a function that maps every continuous variable to a real value: $\mu : X \mapsto \mathbb{R}^{|X|}$. The semantics of a hybrid automaton is defined in terms of the corresponding transition

system defined over symbols $\hat{X} \cup \{\hat{l}\}$, where $\hat{X}$ contains a real-valued symbol for each continuous variable in X and $\hat{l}$ is a symbol not in $\hat{X}$ with domain L. Therefore, each state of the hybrid automaton is a pair $\langle l_0, \mu \rangle$ such that $l \in L$ prescribes the assignment to $\hat{l}$ and μ is an interpretation for the continuous symbols in X that corresponds to a total assignment over $\hat{X}$. We write $\langle l, \mu \rangle \xrightarrow{\delta} \langle l, \mu' \rangle$ for the elapse of $\delta \in \mathbb{R}_0^+$ time in location l and $\langle l, \mu \rangle \rightarrow \langle l', \mu' \rangle$ for the instantaneous transition from $\langle l, \mu \rangle$ to $\langle l', \mu' \rangle$. The transitions are defined as follows:

- $\langle l, \mu \rangle \xrightarrow{\delta} \langle l, \mu' \rangle$ iff there exists a continuous differentiable function $f : [0, \delta] \mapsto \mathbb{R}^{|X|}$ such that $f(0) = \mu$, $f(\delta) = \mu'$ and $\forall \epsilon \in [0; \delta] : (\langle f(\epsilon), \dot{f}(\epsilon) \rangle \models Flow(l)) \wedge f(\epsilon) \models I(l)$;
- $\langle l, \mu \rangle \rightarrow \langle l', \mu' \rangle \iff \exists (l, g, r, l') \in E : (\mu \models g) \wedge (\langle \mu, \mu' \rangle \models r) \wedge (\mu' \models I(l'))$.

It is possible to perform a time elapse of $\delta \in \mathbb{R}_0^+$ within location l if there is a differentiable function f that describes the continuous evolution of the symbols X throughout the time elapse compatible with the flow condition $Flow(l)$ and such that the location invariant $I(l)$ is preserved in all the states in the dense sequence given by the time elapse. As for timed automata, if the flow condition and the location invariant are both convex then it is sufficient for the location invariant to hold at the beginning and at the end of the time elapse. A hybrid automaton allows for a instantaneous transition from state $\langle \mu, l \rangle$ to state $\langle \mu', l' \rangle$ iff there exists $\langle l, g, r, l' \rangle \in E$ such that μ satisfies the guard condition g and $\langle \mu, \mu' \rangle$ satisfies the reset condition r.

Example 11 - Hybrid Automaton

Fig. 2.5 shows a hybrid automaton that keeps the temperature t between 18 and 21 degrees. The thermostat has two locations *off* and *on*. In the first location the temperature decreases over time. In this location its derivative

$\dot{t}$ is negative and at any point in time it must be equal to some value in the open interval $(-0.2t; -0.1t)$. Therefore, the higher the temperature the faster it will decrease. The only outgoing transition requires the temperature to be less than 19. For this reason the thermostat cannot switch on until the temperature falls below this threshold. However, the location has the invariant $t \geq 18$, hence the controller must switch the heather on at some point where the temperature is between 18 and 19 degrees. When the thermostat moves to the *on* location, it starts increasing the temperature over time. In particular, the derivative of the temperature with respect to time is given by the flow condition $\dot{t} = 5 - 0.1t$. Therefore, the lower the temperature the quicker it will rise. The location invariant ensures that we will not heath the room over 22 degrees and the only outgoing transition allows the thermostat to switch off again only if the room has reached at least 21 degrees.

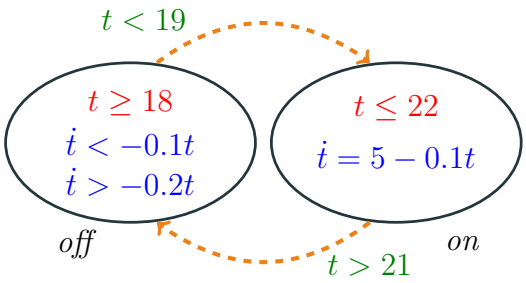

Figure 2.5: Hybrid automaton representing a thermostat, location invariants are highlighted in red, guards in green and flow conditions in blue.

2.6.4 Relationship between TA, HS and ITS

There exists a reduction from the verification problem of hybrid systems to the one of infinite-state transition systems [58]. The reduction allows for a unified approach for the verification of timed, hybrid and infinite-state transition systems. However, it increases the size of both the model and the specification to be verified.

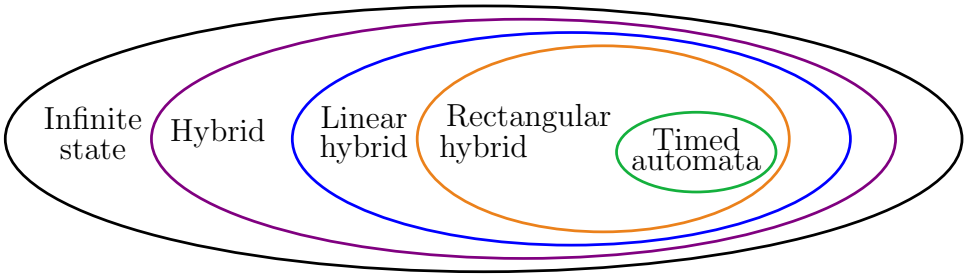

Figure 2.6: Relationship between the expressiveness of the languages of timed, hybrid and infinite-state systems.

Fig. 2.6 shows the relationship between the expressiveness of the languages of the different formalisms. The language of infinite-state transition systems includes all the others. Hybrid automata restrict the infinite-state variables to be continuous with respect to the time development of the system. Linear hybrid automata impose an additional constraint on the language, i.e. linearity. Rectangular systems require the linear functions to be constants and, in addition, in every location the flow of each continuous variable must be bounded from above and below by some rational constants. Finally, timed automata impose all derivatives of continuous variables to be constant and equal to 1.

In this work we represent timed and hybrid systems as the corresponding infinite-state transition systems.

2.7 Linear-Time Temporal Logic

Linear-Time Temporal Logic (LTL) [144] is a temporal logic that reasons on executions as single time lines and each moment in time has a unique possible future. This contrasts with *Branching-Time Logics*, such as *Computation Tree Logic* (CTL) [63], where the model of time is a tree-like structure and each state can split into multiple paths. In both cases we talk about temporal properties and both languages predicate over the rel-

ative ordering of events. The discussion about the respective merits of the two models of computation dates back to the 1980's [140] and it is outside the scope of this thesis. We simply remark that the languages of LTL and CTL are incomparable. In fact, while there are formulae that can be expressed in both languages, there are also formulae that can be expressed only in LTL and not in CTL and vice versa. CTL* [82] is a temporal logic that combines, and subsumes, both LTL and CTL. The class of languages expressible in LTL specifications is the class of *star-free ω-regular* languages [44]; while, CTL and CTL* can be represented using *Hesitant Alternating Automata* [128].

In this work we restrict our attention to linear-time temporal logics and consider first-order LTL extended with past operators.

This section first introduces the syntax of LTL formulae, then formally defines its semantics and finally outlines some well-known procedures employed for the verification of such specifications on transition systems.

2.7.1 Syntax

A LTL formula is a formula in quantifier-free first-order logic extended with temporal modalities, also called operators. **X** (next), **G** (globally), **F** (finally) and **U** (until) are the modalities that express conditions over the future states, while **Y** (yesterday), **H** (historically), **P** (previously) and **S** (since) are the operators that refer to past events.

We define the syntax of a LTL formula φ as follows. A Σ-atom is a LTL formula, a Boolean operator applied to LTL formulae is a LTL formula and a temporal operator applied to LTL formulae is a LTL formula.

$$\begin{aligned}\varphi ::\ & p(u,\ldots,u) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid \\ & \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi\mathbf{U}\varphi \mid \mathbf{Y}\varphi \mid \mathbf{H}\varphi \mid \mathbf{P}\varphi \mid \varphi\mathbf{S}\varphi \\ u ::\ & c \mid x \mid f(u,\ldots,u) \mid \mathbf{Ite}(\varphi,u,u)\end{aligned}$$

where x is a variable in V and p, u, f and c are, respectively, a predicate symbol, a term, a function symbol and a constant in Σ. Furthermore, in discrete time-model we consider an additional function *next* and extend the syntax for terms as follows.

$$u :: \; c \mid x \mid f(u, \ldots, u) \mid next(u)$$

2.7.2 Semantics

In the following we report the semantics of LTL with respect to a time model τ (either discrete, dense, super-dense or super-discrete).

Let $inst_\tau(t, t')$ be a formula that holds for $t, t' \in \tau$ iff there is an instantaneous transition from t to t':

$$inst_\tau(t, t') \doteq \begin{cases} \mathbf{0} < t < t' \wedge \neg \exists t'' \in \tau : t < t'' < t' & \text{if } \tau \in \{discrete, dense\}; \\ \exists i \geq 0, r \geq 0 : t = \langle i, r \rangle \wedge t' = \langle i + 1, r \rangle & \text{otherwise.} \end{cases}$$

The formula ensures that there exists no time point t'' between t and t'. Notice that in a dense set it is always possible to find such a point, hence $inst_\tau$ is always false in the dense time model. Instead, in the discrete time model $step(t, t')$ holds iff $t' = t + 1$. Finally, in the super-dense and super-discrete time models it holds iff $v(t) = v(t')$ (i.e. we do not move along the "time" direction) and the first component of t' is the first component of t increased by 1 (i.e. single instantaneous step). Since, the domain of the first component is $\mathbb{N}$, this implies that there exists no timepoint t'' such that $t < t'' < t'$. In this context, we use $inst_\tau$ to define the $\mathbf{X}$ and $\mathbf{Y}$ modalities such that they are always false in time elapses and hold in instantaneous transitions iff the argument holds in the next/previous time point.

Given a path $\sigma \doteq \langle M, \tau, \overline{\mu} \rangle$, a time point $t \in \tau$ and a LTL formula φ, we define $\sigma, t \models \varphi$ recursively as follows.

- $\sigma, t \models p(u_1, \ldots, u_n)$ iff $\sigma(t) \models p(u_1, \ldots, u_n)$, where $p(u_1, \ldots, u_n)$ is a Σ-atom;
- $\sigma(t)(\mathbf{Ite}(\varphi, u_0, u_1)) = \begin{cases} \sigma(t)(u_0) & \text{if } \sigma, t \models \varphi; \\ \sigma(t)(u_1) & \text{otherwise;} \end{cases}$
- $\sigma, t \models \varphi_0 \wedge \varphi_1$ iff $\sigma, t \models \varphi_0$ and $\sigma, t \models \varphi_1$;
- $\sigma, t \models \neg\varphi$ iff $\sigma(t) \not\models \varphi$;
- $\sigma, t \models \mathbf{X}\varphi$ iff $\exists t' \in \tau$ such that $inst_\tau(t, t')$ holds and $\sigma, t' \models \varphi$;
- $\sigma, t \models \mathbf{Y}\varphi$ iff $\exists t' \in \tau$ such that $inst_\tau(t', t)$ holds and $\sigma, t' \models \varphi$;
- $\sigma, t \models \varphi_0 \mathbf{U} \varphi_1$ iff $\exists t' \in \tau : t' \geq t$ such that $\sigma, t' \models \varphi_1$ and $\forall t'' \in \tau : t \leq t'' < t'$ then $\sigma, t'' \models \varphi_0$;
- $\sigma, t \models \varphi_0 \mathbf{S} \varphi_1$ iff $\exists t' \in \tau : t' \leq t$ such that $\sigma, t' \models \varphi_1$ and $\forall t'' \in \tau : t' < t'' \leq t$ then $\sigma, t'' \models \varphi_0$.

The semantics of *next* is defined only for the discrete time model as follows: $\sigma(t)(next(u)) \doteq \sigma(t+1)(u)$.

Finally, we define the semantics of the other temporal and Boolean operators in terms of the ones above as:

- $\sigma, t \models \varphi_0 \vee \varphi_1$ iff $\sigma, t \models \neg(\neg\varphi_0 \wedge \neg\varphi_1)$;
- $\sigma, t \models \varphi_0 \rightarrow \varphi_1$ iff $\sigma, t \models \neg(\varphi_0 \wedge \neg\varphi_1)$;
- $\sigma, t \models \varphi_0 \leftrightarrow \varphi_1$ iff $\sigma, t \models (\varphi_0 \wedge \varphi_1) \vee (\neg\varphi_0 \wedge \neg\varphi_1)$;
- $\sigma, t \models \mathbf{F}\varphi$ iff $\sigma, t \models \top \mathbf{U} \varphi$;
- $\sigma, t \models \mathbf{G}\varphi$ iff $\sigma, t \models \neg(\top \mathbf{U} \neg\varphi)$;
- $\sigma, t \models \mathbf{P}\varphi$ iff $\sigma, t \models \top \mathbf{S} \varphi$;
- $\sigma, t \models \mathbf{H}\varphi$ iff $\sigma, t \models \neg(\top \mathbf{S} \neg\varphi)$.

The if-then-else operator is not standard in LTL. However, every if-then-else term $\mathbf{Ite}(\phi, u_\top, u_\bot)$ appearing in some predicate $p(u_0, \ldots, u_n)$ can

be removed by bringing the condition ϕ outside the predicate.

$$\sigma, t \models p(u_0, \ldots, u_n) \text{ iff } \sigma, t \models (\phi \wedge p(u_0, \ldots, u_i)[\mathbf{Ite}(\phi, u_\top, u_\bot)/u_\top]) \vee (\neg\phi \wedge p(u_0, \ldots, u_i)[\mathbf{Ite}(\phi, u_\top, u_\bot)/u_\bot])$$

Therefore, the operator does not change the expressiveness of the language, but allows for shorter and easier to read formulae.

A path σ satisfies a temporal property φ, written $\sigma \models \varphi$ if and only if the property holds in its initial state: $\sigma, \mathbf{0} \models \varphi$. A temporal formula φ is satisfiable [resp. valid] iff there exists [resp. for all] σ such that $\sigma \models \varphi$. Two temporal formulae are equivalent $\varphi_0 \equiv \varphi_1$ iff every model of φ_0 is a model for φ_1 and viceversa.

$$\equiv \doteq \{\langle \varphi_0, \varphi_1 \rangle \mid \forall \sigma : \sigma, \mathbf{0} \models \varphi_0 \iff \sigma, \mathbf{0} \models \varphi_1\}$$

The notion of equivalence requires two formulae to hold on the same traces starting from the first time point. We now introduce stronger notions of entailment and equivalence to represent formulae that also hold on the same sub-traces. Given two temporal formulae φ_0 and φ_1, φ_0 *globally entails* φ_1, written $\varphi_0 \models_G \varphi_1$, iff for all traces σ, for all time points t, $\sigma, t \models \varphi_0$ implies $\sigma, t \models \varphi_1$. We say that φ_0 and φ_1 are *globally equivalent*, written $\varphi_0 \equiv_G \varphi_1$, iff for every trace σ and every time point t, $\sigma, t \models \varphi_0$ iff $\sigma, t \models \varphi_1$.

$$\equiv_G \doteq \{\langle \varphi_0, \varphi_1 \rangle \mid \forall \sigma, t : \sigma, t \models \varphi_0 \iff \sigma, t \models \varphi_1\}$$

Therefore, since every trace must admit at least one time point, $\equiv \subseteq \equiv_G$. We exemplify the difference between the two equivalence relations by considering rewritings. In rewritings we want to replace subformulae obtaining equivalent formulae. In this cases we need to replace a subformula with a globally equivalent formula, since the subformula can appear within other temporal operators that require it to hold at previous or future time points. For example, $\mathbf{Y}a \equiv \mathbf{Y}\neg a$, however the equivalence cannot be used as a rewriting: $\mathbf{XY}a \not\equiv \mathbf{XY}\neg a$.

We remark that $\mathbf{Y}$ is always false in the first time point ($t = \mathbf{0}$) and both $\mathbf{X}$ and $\mathbf{Y}$ are always false in the dense time model. This follows from the definition of $inst_\tau$ and implies that, in general, $\mathbf{X}\neg\varphi$ is not equivalent to $\neg\mathbf{X}\varphi$. In fact, this equivalence holds only when considering the discrete time model. It is possible to introduce a counterpart for $\mathbf{X}$ [resp. $\mathbf{Y}$] that predicates about the next [resp. previous] time point in a dense sequence or along the time dimension of the super-discrete and super-dense time models. We do not introduce such operators here since they are not standard LTL operators; they are defined in Sec 4.1 as $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$.

Notice that the semantics defined above in the case of discrete time model reduces to the usual LTL semantics described for "untimed" (i.e. discrete) systems [144]. In fact, in the case of discrete time model, a path can be described as a sequence of states $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots$, where state $\boldsymbol{v}_i$ corresponds to time point i and $v(i) = i$. Then, the temporal operators can be interpreted as follows. $\mathbf{X}\varphi_0$ (next) holds in the current state if φ_0 holds in the next state, $\varphi_0\mathbf{U}\varphi_1$ (until) states that φ_1 holds in the current state, or there exists a state $\boldsymbol{v}_{\varphi_1}$ in the future that satisfies φ_1 and all states between the current one and $\boldsymbol{v}_{\varphi_1}$ satisfy φ_0, $\mathbf{G}\varphi_0$ (globally) holds in the current state if φ_0 will hold from now on, $\mathbf{F}\varphi_0$ (finally) states that there exists a state in the future that satisfies φ_0; $\mathbf{Y}\varphi_0$ (yesterday), $\varphi_0\mathbf{S}\varphi_1$ (since), $\mathbf{H}\varphi_0$ (historically) and $\mathbf{P}\varphi_0$ (previously) are the symmetric temporal operators that consider past states.

2.7.3 Model Checking Problem

Given a transition system M and a LTL specification φ the model checking problem is the problem of deciding whether M satisfies φ. We say that M satisfies φ, written $M \models \varphi$ if and only if every path of M satisfies the property φ.

$$M \models \varphi \iff \forall \pi \in \mathcal{L}(M) : \pi \models \varphi$$

A widely used approach to perform model checking of LTL specifications, known as *automata-theoretic* approach [159], relies on the construction of a (generalised) fair transition system whose language contains all and only the paths in which the specification holds. Such fair transition system is called *tableau* of the LTL specification. In general, the resulting transition system can be exponentially larger than the LTL formula and many different procedures have been proposed to perform this transformation while trying to minimise the size of the resulting transition system [74, 154, 90, 13] or on-the-fly [91]. In this work we do not go into the details of these transformations, but simply recall why they are important in the context of LTL model checking.

The tableau construction for LTL specifications allows the reduction of the LTL satisfiability and model checking problems to deciding whether the language of some fair transition system is empty. In more detail, let φ be an LTL formula, M be a fair transition system and $M_{\neg\varphi}$ be the tableau of the negation of the formula φ, then $\mathcal{L}(M_{\neg\varphi})$ is the set of all paths such that $\neg\varphi$ holds.

$$\pi \in \mathcal{L}(M_{\neg\varphi}) \iff \pi \models \neg\varphi$$

The synchronous composition $M \times M_{\neg\varphi}$ of the transition system with the tableau is a fair transition system whose language is empty iff every path in $\mathcal{L}(M)$ satisfies φ. In fact, a path $\pi \in \mathcal{L}(M)$ such that $\pi \models \neg\varphi$, by construction, would also be a path of the tableau, i.e. $\pi\mathcal{L}(M_{\neg\varphi})$. Therefore, π is a path in the synchronous product of the two transition systems.

$$(M \models \varphi) \iff (\mathcal{L}(M \times M_{\neg\varphi}) = \emptyset)$$

A classical result in LTL model checking for finite-state system states that if a finite-state transition system M admits a path π such that $\pi \not\models \varphi$, for some LTL property φ, then M must admit also a lasso-shaped path π' such that $\pi' \not\models \varphi$. A path is called lasso-shaped iff it is composed of a

stem, also called *prefix*, and a *loop* as depicted in Fig. 2.7.

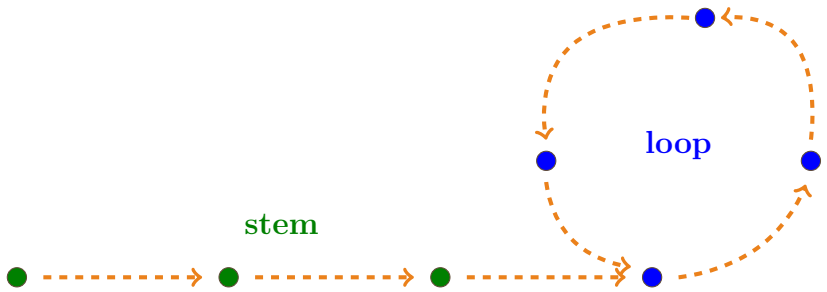

Figure 2.7: Lasso-shaped path.

Intuitively, this follows from the observation that given a finite number of states, it is impossible to build a infinite sequence of distinct states. Therefore, any infinite sequence must contain some state multiple times; these repetitions correspond to the loops of lasso-shaped paths.

This result guarantees that, in the finite-state case, it is sufficient to look for lasso-shaped paths in order to solve the LTL model checking problem. This also implies that if the language of a fair transition system is not empty, then it must contain at least one lasso-shaped path. Notice that the loop of such path must contain at least one fair state.

These observations lead to a procedure that in polynomial-time decides whether the language of a finite-state fair transition system is empty. This procedure, combined with the exponential-time computation of the tableau of a LTL formula, leads to an exponential-time decision procedure for the LTL model checking problem.

We can decide whether the language of a fair transition system M is empty in polynomial-time with respect to the number of states in the system. An example of such decision procedures is the *double-nested DFS* algorithm [70]. The algorithm performs a visit on the graph corresponding to the transition system looking for a path from an initial state to a fair state. If it identifies such a path, then it performs another visit starting from the fair states and trying to reach a state in the path. If it succeeds

then it found a lasso-shaped path such that the loop contains a fair state and the language of the system is not empty. If no such path is found, then the system admits no lasso-shaped fair path and we can conclude that its language is empty.

Unfortunately, these results do not hold in the case of infinite-state systems. In fact, in such systems there can be infinite sequences of distinct states and such sequences can correspond to fair paths in its language. Intuitively, exploring an infinite sequence can require an infinite number of steps and the LTL model checking problem on infinite-state transition systems is, in general, undecidable. A 2-counter machine can be easily represented as a infinite-state transition system and LTL specifications are semantic properties over the system behaviour, hence by Rice's theorem [114] the model checking problem is undecidable.

2.8 Symbolic Model Checking

Model checking procedures can be classified in two main categories: symbolic and explicit-state.

Explicit-state techniques try to decide the validity of some property ϕ on a transition system M by exploring the graph representing it. Each node in the graph corresponds to a single state and the edges represent the transitions between states. The double-nested DFS procedure we briefly described in §2.7.3 is an example of a explicit-state algorithm for LTL model checking.

As the number of states grows explicit approaches become unpractical. The number of states grows exponentially in the number of variables and explicit techniques analyse one state at a time. Symbolic techniques mitigate this issue by reasoning about sets of states and transitions at a time. The sets of states and transitions are usually represented as first-order for-

mulae. A formula $\psi(V)$ denotes the set of states where the formula holds $\lfloor\psi\rfloor \doteq \{\boldsymbol{v} \mid \boldsymbol{v} \models \psi(V)\}$. In this context we say that a state $\boldsymbol{v}$ is in ψ meaning that $\boldsymbol{v} \in \lfloor\psi\rfloor$. Notice that the size of the formula is not related to the cardinality of the set it represents. For example, consider a transition system with a single real valued variable x. In this setting, $x \geq 0$ represents an infinite set of states (the positive reals) and $x = 0$ represents a single state. However, given a set of states there could be no finite formula to represent it. Most symbolic techniques rely on *Binary Decision Diagrams* (BDD) or SAT/SMT-solvers to perform operations on such sets. In this work we are interested in symbolic techniques for infinite-state systems and we rely on SMT-solvers.

In the following we briefly present some relevant SMT-based symbolic model checking techniques.

2.8.1 Reachability

The reachability problem requires to decide whether a transition system M admits a finite path ending in some state in ϕ, written $M \rightsquigarrow \phi$. The reachability problem is the complement of the invariant model checking problem, which involves deciding whether a quantifier-free formula ϕ holds in all reachable states of M ($M \models \phi$). $M \models \phi$ holds if and only if $M \rightsquigarrow \phi$ does not.

Reachability is a fundamental concept in model checking. Many techniques and algorithms effectively reduce the model checking problem to a sequence of reachability problems. For example, the double-nested DFS algorithm outlined in §2.7.3 requires to solve two distinct reachability problems. The first one to identify a path from an initial state to a fair state and the second one to find a path from the fair state to one of the other states already visited.

2.8.2 Bounded Model Checking

Bounded Model Checking (BMC) [32, 31] is a symbolic approach that incrementally builds formulae representing all paths of a system of length k. In this sense BMC performs a BFS on the graph representing the system. It progressively explores the state-space by increasing the bound k until a counterexample is found or some upper bound of k is reached. At each step k BMC builds a formula whose models are only counterexamples for the property, hence the model checking problem is reduced to SAT, in the finite-state case, and SMT in the infinite-state case.

We will describe how to apply BMC to solve the reachability problem. However, BMC can also be used to directly search for a lasso-shaped counterexample for a LTL property [31], avoiding the expensive tableau and product construction. Notice that this still leads to an exponential-time procedure in the finite case. In fact, the problem is reduced to a sequence of SAT queries, each of which requires exponential time.

Let $M \doteq \langle V, I, T \rangle$ be a transition system and $\phi(V)$ a quantifier-free formula. Consider the associated reachability problem $M \rightsquigarrow \phi$. For $k \in \mathbb{N}$ the BMC encoding unrolls k transitions of the system and builds a formula that is satisfiable iff M admits a path of length k ending in some state in ϕ. This can be written as

$$I(V_0) \wedge \phi(V_k) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}),$$

where each $V_i \doteq \{v_i \mid v \in V\}$ is a copy of the symbols in V representing the assignment of the i^{th} state in the path.

In practical cases, BMC proved to be very effective in falsifying properties. The intuition behind its success is that most of the time bugs that cause a property to be false do not require a huge number of steps; they are "shallow" and the unrolling will only visit a portion of the whole state-

space. Notice that, since we progressively increase the bound k, BMC will always find the path falsifying the property that requires the smallest amount of steps.

Consider a model M and an invariant property ϕ defined as formulae over a decidable SMT theory. Assume the theory to be LIA, M is still expressive enough to model any 2-counter machine. For this reason, both invariant model checking and reachability are undecidable for infinite-state systems. Notice that in the case of a decidable theory, each SMT query performed by BMC is decidable and BMC is a semi-procedure for the reachability problem. This fact implies that the invariant model checking problem cannot be semi-decidable.

2.8.3 Liveness to Safety

A popular approach to exploit algorithms originally developed for invariant checking in LTL model checking is *liveness-to-safety* (L2S) [29]. At the cost of doubling the size of the model, it reduces the language emptiness problem (hence also of the LTL model checking problem) into an invariant verification problem. Given a fair transition system $M \doteq \langle V, I, T, F \rangle$ it constructs a transition system $M_{L2S} \doteq \langle V_{L2S}, I_{L2S}, T_{L2S} \rangle$ and a property $\phi_{L2S}(V_{L2S})$ such that $M_{L2S} \rightsquigarrow \phi_{L2S}$ implies $\exists \pi \in \mathcal{L}(M)$. In the case of finite-state systems, the procedure is both sound and complete, i.e. $M_{L2S} \rightsquigarrow \phi \iff \exists \pi \in \mathcal{L}(M)$.

L2S builds a reachability problem that encodes the search for a fair lasso-path of M. For every state variable in V it introduces a "monitor" in the form of a fresh variable of the same type. The assignment to each monitor is nondeterministically chosen in the initial state and then can never change. L2S requires to search for a finite path that visits two times the state where each variable in V is equal to its corresponding monitor (i.e. a loop), and between them there is a fair state (i.e. a fair loop).

More formally we define the transformation as follows.

Definition 17 - Liveness-to-Safety
Let $M \doteq \langle V, I, T, F \rangle$ *be a fair transition system, its liveness-to-safety transformation is the transition system* $M_{L2S} \doteq L2S(M) \doteq \langle V_{L2S}, I_{L2S}, T_{L2S} \rangle$ *such that:*

- $V_{L2S} \doteq V \cup \{m_v \mid v \in V\} \cup \{in\_loop\}$, *where* in_loop *is a fresh Boolean atom and for each* $v \in V$ m_v *is a fresh variable with the same domain of* v*;*
- $I_{L2S} \doteq I \wedge \neg in\_loop$*;*
- $T_{L2S} \doteq T \wedge (\bigwedge_{v \in V} m'_v = m_v) \wedge in\_loop' \rightarrow (in\_loop \vee (F \wedge \bigwedge_{v \in V} v = m_v))$*;*
- $\phi_{L2S} \doteq in\_loop \wedge \bigwedge_{v \in V} v = m_v$.

In this encoding we require the loop-back state ($\bigwedge_{v \in V} v = m_v$) to be a fair state. This does not affect the generality of the approach and simplifies the encoding. The assignment to the monitors m_v is chosen at the beginning and then never changes. T_{L2S} ensures that the Boolean symbol in_loop becomes true the first time we visit the fair state described by the monitors. The reachability problem, given by ϕ_{L2S}, requires us to find a path ending in a state where in_loop holds and the assignment to the symbols in V corresponds to the assignment to the monitors. The last state, since in_loop holds, must be the second time the path visits such state. Therefore, the path is lasso-shaped and, since the loop-back state is fair, it must correspond to a fair lasso-shaped path over the symbols in V for M. Notice that, since we encode the search of a lasso-shaped path, in infinite-state systems L2S can be used only to identify counterexamples and does not allow to conclude that the property holds.

2.8.4 Abstraction

In 1977 Cousot and Cousot described a technique called *abstract interpretation* (AI) [72]. They propose to study properties of programs by abstracting away all additional information that is not necessary to prove the property. For example, to compute the sign of result of a multiplication, it is sufficient to know the signs of all factors and we do not need to know their precise value. While Cousot's focus is on verification of software, the applicability of AI is much broader and, as he stated in [71], provides a mathematical formalisation of the concept of *approximation*. In this context an *abstraction function* α maps objects from a concrete domain to a corresponding abstract space and a *concretisation function* γ maps elements from the abstract space into their concretisation (i.e. element in the concrete domain).

Given two fair transition systems $M_u \doteq \langle V_u, I_u, T_u, F_u \rangle$ and $M \doteq \langle V, I, T, F \rangle$ we say that M_u *underapproximates* M and that M *overapproximates* M_u, iff $\mathcal{L}(M_u) \subseteq \mathcal{L}_{\downarrow V_u}(M)$.

Identifying the correct/ideal level of abstraction to prove a property is a non-trivial challenge. In some specific contexts, such as invariant and LTL verification on TA, there exist abstractions ensuring that the property holds in the concrete system iff it does on the abstract one. However, in general, when defining the abstraction function, on one hand we would like to disregard as many details as possible in order to reason on a much simpler system, but on the other hand if we abstract to much information, then we could be unable to (dis)prove our property. In addition, identifying the "right" level of abstraction a-priori is often unfeasible, for this reason a key concept related to abstraction is the one of *refinement*. Given an abstraction that is too coarse to (dis)prove some property, we refine it obtaining another abstraction that is more "precise". Key elements for

refinements are the identification of some root cause for the abstraction being too coarse and the application of a refinement to remove such cause.

Counterexample-Guided Abstraction Refinement

A popular approach for abstraction-refinement is *counterexample-guided abstraction refinement* (CEGAR) [64], also referred to as CEGAR-loop. Given a model checking problem $M \models \varphi$, we compute an overapproximation M_o of M such that $M_o \doteq \alpha(M)$ and $M_o \models \varphi$ implies that $M \models \varphi$ is also true. Therefore, we can apply model checking techniques in the abstract space; if the property holds in the abstract system then it must hold also in the concrete system. However, if $M_o \not\models \varphi_o$, then we obtain a path $\pi_o \in \mathcal{L}(M_o)$ such that $\pi_o \not\models \varphi_o$. π_o is a path in the abstract space that corresponds to a set of paths in the concrete space given by $\gamma(\pi_o)$. If there is some concrete path corresponding to π_o in the language of M, then this is a counterexample in the concrete system and we can conclude that $M \not\models \varphi$. Otherwise, if π_o has no concrete counterpart in $\mathcal{L}(M)$, we call π_o *spurious*. In this case we refine the abstraction function α such that we remove at least the spurious counterexample π_o from the language of the abstract system. We define α' such that $\pi_o \notin \mathcal{L}(\alpha'(M))$ and $\alpha'(M)$ is still an overapproximation of M. In many cases, and depending on the abstraction function considered, it is possible to compute a "reason" for π_o being spurious and refine the abstraction by removing a possibly infinite set of spurious counterexamples at a time.

Predicate Abstraction

A widely used abstraction is *predicate abstraction* where the system is abstracted by considering the truth assignments to a finite set of predicates $\mathbb{P}$. Notice that this implies that the abstract space is always finite. The abstraction is such that if $\widehat{M_{\mathbb{P}}}$ is the predicate abstraction of M and a

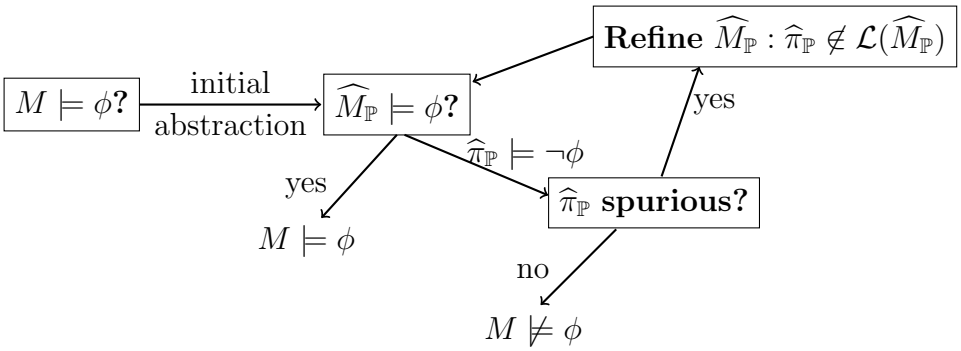

Figure 2.8: CEGAR loop.

condition ϕ is reachable in M then it must be reachable also in $\widehat{M}_{\mathbb{P}}$, written

$$(M \rightsquigarrow \phi) \rightarrow (\widehat{M}_{\mathbb{P}} \rightsquigarrow \phi).$$

Given a transition system $M \doteq \langle V, I, T \rangle$ and a set of n predicates over V $\mathbb{P} \doteq \{p_i(V) \mid n \in \mathbb{N} \wedge 0 \leq i < n\}$ we define the abstract space as the set of assignments to the abstract symbols $V_{\mathbb{P}} \doteq \{v_p \mid p \in \mathbb{P}\}$, where each v_p is a Boolean symbol representing the truth assignment of predicate p. The abstraction relation is defined as $\alpha_{\mathbb{P}}(V, V_{\mathbb{P}}) \doteq \bigwedge_{p \in \mathbb{P}} v_p \leftrightarrow p(V)$. The relation associates each abstract state (total assignment over $V_{\mathbb{P}}$) to the region in the concrete space described by the corresponding assignments to the predicates in $\mathbb{P}$. The abstraction of a formula $\phi(V)$ is defined as $\widehat{\phi}_{\mathbb{P}}(V_{\mathbb{P}}) \doteq \exists V : \phi(V) \wedge \alpha_{\mathbb{P}}(V, V_{\mathbb{P}})$. Similarly, we define the abstraction of a relation $\psi(V, V')$ as $\widehat{\psi}_{\mathbb{P}}(V_{\mathbb{P}}, V'_{\mathbb{P}}) \doteq \exists V, V' : \psi(V, V') \wedge \alpha_{\mathbb{P}}(V, V_{\mathbb{P}}) \wedge \alpha_{\mathbb{P}}(V', V'_{\mathbb{P}})$. Finally, we define the abstraction of transition system M with respect to predicates $\mathbb{P}$ as $\widehat{M}_{\mathbb{P}} \doteq \langle V_{\mathbb{P}}, \widehat{I}_{\mathbb{P}}, \widehat{T}_{\mathbb{P}} \rangle$, where $\widehat{I}_{\mathbb{P}}$ is the abstraction of the initial states of M and $\widehat{T}_{\mathbb{P}}$ is the formula corresponding to the abstraction of the transition relation of M.

Abstraction for Infinite-State Model Checking

Recently, abstraction has been exploited also in SMT-based model checking [129]. Many SAT-based model checking techniques for finite-state

systems have been extended for the infinite-state case by replacing the SAT-solver with a SMT-solver combined with abstraction techniques.

Abstraction techniques have been applied to symbolic algorithms, such as BMC [125], k-induction [124] and *Property Directed Reachability* (PDR) [56], to obtain approaches capable of checking invariant specifications on infinite-state transition systems. Simply replacing the SAT-solver with an SMT-solver would not lead to viable model checking algorithms. The refinement step applied by some of these algorithms (e.g. PDR) would remove a single point from an infinite set, thus it is unlikely that the procedure will converge and eventually terminate. For this reason, different approaches have been proposed that try to improve the refinement by generalising the state to be removed. In this context [34, 112, 116, 124, 130, 161] propose generalisation techniques that are tailored to a specific theory used by the SMT-solver or rely on particular restrictions of the modelling language, while others [33, 56, 118, 120] try to reduce as much as possible the theory specific component by reasoning on an abstract over-approximation of the model.

Implicit Abstraction

Abstraction techniques require the computation of an abstract system. The explicit computation of the abstract space could lead to the state-explosion issue we already described for explicit-state techniques. *Implicit abstraction* [155] avoids the upfront computation of the abstract system, while preserving the advantages of reasoning on an abstract space.

In the following we describe implicit predicate abstraction, an approach to embed the definition of the predicate abstraction in the formula used to represent paths of the transition system. As in predicate abstraction, we consider a finite set of predicates $\mathbb{P}$ over the symbols of the transition system $M \doteq \langle V, I, T \rangle$. We define a relation $EQ_{\mathbb{P}}$ such that two states $\overline{\boldsymbol{v}}$,

$\boldsymbol{v}$ are in the relation iff they correspond to the same abstract state. The relation can be symbolically represented as

$$EQ_{\mathbb{P}}(\overline{V}, V) \doteq \bigwedge_{p \in \mathbb{P}} p(\overline{V}) \leftrightarrow p(V),$$

where $\overline{V} \doteq \{\overline{v} \mid v \in V\}$. Given this equivalence relation, we can symbolically represent a path of length k of the abstraction of M as

$$\widehat{Path}_{k,\mathbb{P}} \doteq T(\overline{V}_{k-1}, V_k) \bigwedge_{1 \leq i < k} T(\overline{V}_{i-1}, V_i) \wedge EQ_{\mathbb{P}}(\overline{V}_{i-1}, V_i).$$

Chapter 3

Problem Definition

This chapter describes the main problem we address in this thesis.

In many systems, time and delays play a significant role in their correct operation. For example, real-time systems often require actions to be performed with precise timing constraints and in distributed systems the interactions between components are often associated with timeouts that allow the computation to continue even in the event of network or node failures. In these contexts, we need a specification language capable of precisely capture the interplay between computation and time. For this reason, temporal logics have been extended with metric operators to quantify the distance of events with respect to time [39]. However, there is a lack of tools capable of verifying such properties and they often rely on additional restrictions of the modelling or specification languages.

In this work, we propose the semantics for an extended version of *Metric Temporal Logic* [127] that can be interpreted over different models of time. We consider its model checking problem on timed systems and propose a reduction to model checking of LTL specifications on ITS. This allows us to employ all model checking techniques developed in this context. These techniques show to be reasonably effective in proving the validity of the properties, however they have limited falsification capability. In fact, exist-

ing model checking approaches are capable of identifying only lasso-shaped counterexamples. Other techniques capable of identifying counterexamples not in lasso shape usually consider particular types of systems (e.g. lasso programs) and represent counterexamples using ad hoc structures. Therefore, they lack the generality we require and do not consider fairness.

These limitations severely hinder the definition of formal models for infinite-state and timed systems. For any meaningful system the definition of its formal representation is usually an iterative process and careful analysis is necessary to reach sufficient confidence in the correctness of the model and its specifications. The formalisation of the system under consideration is progressively refined by analysing the counterexamples to the specifications. If the model checker is too limited in its falsification capability it will fail to provide a definite answer, hence it will provide no information to the designer. At this point the designer could lift the abstraction level of the formalisation. However, identifying the correct abstraction level and the component of the system to be abstracted is a challenging problem that relies on the knowledge of both the formalisation languages and verification procedures.

Furthermore, a model with an empty language satisfies all properties. For this reason, in the development of the formalisation of a system it is important to ensure that both model and specifications do not contain contradictions. Therefore, counterexamples are important not only to identify issues in the formalisation, but also to ensure that it correctly represent the intended behaviours. This can be achieved by exploiting the falsification capability of model checkers to ensure that the model allows every intended behaviour. For every such behaviour it is sufficient to define a specification that negates its existence. If the model checker identifies a counterexample for such property we are guaranteed that the model admits at least one such desired behaviour. From this perspective we can organise the specifi-

cations of a system into two categories: valid properties that ensure that the system does not admit any undesired behaviour and false properties whose counterexamples correspond to desired executions of the system.

In this work, we address this issue by proposing novel techniques to represent and identify fair paths at the ITS level. While the problem is in general undecidable, the techniques proved to be effective on a wide range of benchmarks and also capable of identifying witness that all other tools could not find.

Part II

Extending temporal logics

Chapter 4

Extending Temporal Logics with Metric Operators

Note. The work presented in this chapter is the result of the collaboration with Stefano Tonetta and Marco Roveri. This chapter extends our work [55] by considering also the super-discrete time model and adapting the semantics and reductions accordingly.

Temporal logics, such as LTL, predicate about the relative ordering of events but cannot quantify their distance with respect to some notion of time. In the context of timed systems the capability of quantifying such distance is, at the very least, desirable and timed temporal logics have been defined and studied since the 1990's. Many extensions of both branching and linear time logics have been proposed [39], in this thesis we focus on the linear-time case and consider *Metric Temporal Logic* (MTL) [127]. MTL extends LTL with decorators on the temporal operators in the form of intervals specifying the time boundaries in which the formula must be satisfied. In the literature there are different definitions for the semantics of MTL that also consider different models of time.

The semantics of temporal logics can be partitioned into two broad categories: *discrete-observation* (or *pointwise*) and *continuous-observation* (or

continuous) semantics [39]. They differ in the definition of which part of the system is observed. In discrete-observation semantics the satisfaction of subformulae is observed only at discrete time points and not while time elapses; instead, in continuous-observation semantics every clock evaluation in the time elapse is observed. This distinction can change the truth value of a formula and also affects the complexity of the model checking problem [37].

Specifications with discrete-observation semantics can always be reduced to the discrete and super-discrete time models, while continuous-observation semantics require either the dense or super-dense time models. For this reason, we will implicitly assume discrete-observation semantics when dealing with discrete and super-discrete time models and continuous-observation semantics in dense and super-dense time models.

The chapter is structured as follows. Sec. 4.1 defines LTL-EF: an extension of LTL with event-freezing functions. Then, we extend LTL-EF with metric temporal operators and define MTLC in Sec. 4.2. MTLC extends MTL, interpreted over first-order predicates, with parametric intervals and counting operators. Sec. 4.3 reduces the model checking problem of $\text{MTLC}_{0,\infty}$ (a fragment of MTLC) on dense, super-dense or super-discrete time to LTL verification on discrete time model. The reduction is obtained by combining the following three steps. First, a $\text{MTLC}_{0,\infty}$ formula is rewritten in LTL-EF (§4.3.1). Then, for every dense, super-dense and super-discrete trace we show how to build a corresponding trace and LTL-EF formula with discrete time model (§4.3.2). Finally, we remove the event-freezing functions from the LTL-EF specifications obtaining the problem of verifying a LTL property on a ITS (§4.3.3).

4.1 LTL with Event-Freezing Functions

This section defines the syntax and semantics of LTL-EF. LTL-EF extends LTL (2.7) with four new temporal operators: $\tilde{\mathbf{X}}$, $\tilde{\mathbf{Y}}$, $@\tilde{\mathbf{F}}$ and $@\tilde{\mathbf{P}}$. $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ are the timed counterparts of $\mathbf{X}$ and $\mathbf{Y}$. They predicate about the "next state" along the time dimension, while $\mathbf{X}$ and $\mathbf{Y}$ specify constraints about the next state reached via an instantaneous transition. The event-freezing functions $u@\tilde{\mathbf{F}}(\varphi)$ (at next) and $u@\tilde{\mathbf{P}}(\varphi)$ (at last), take as input a term u and a formula φ, and represent the value of u at the next point in the future or last point in the past, respectively, in which φ holds. If no such time point exists, then we define them equal to a default value represented by a variable $def_{u@\tilde{\mathbf{F}}(\varphi)}$ for $@\tilde{\mathbf{F}}$ and $def_{u@\tilde{\mathbf{P}}(\varphi)}$ for $@\tilde{\mathbf{P}}$. Therefore, all occurrences of the same $@\tilde{\mathbf{F}}$ [resp. $@\tilde{\mathbf{P}}$] formula share the same default value.

4.1.1 Syntax

The syntax of a LTL-EF is defined as follows.

$$\begin{aligned}\varphi ::\ & p(u,\ldots,u) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid \\ & \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi\mathbf{U}\varphi \mid \mathbf{Y}\varphi \mid \mathbf{H}\varphi \mid \mathbf{P}\varphi \mid \varphi\mathbf{S}\varphi \mid \tilde{\mathbf{X}}\varphi \mid \tilde{\mathbf{Y}}\varphi \\ u ::\ & c \mid x \mid f(u,\ldots,u) \mid u@\tilde{\mathbf{F}}(\varphi) \mid u@\tilde{\mathbf{P}}(\varphi) \mid \mathbf{Ite}(\varphi,u,u)\end{aligned}$$

where x, p, f and c are a variable in V, a predicate, a function and a constant symbol in Σ respectively.

Therefore, LTL-EF allows for all the propositional and temporal operators of LTL. LTL-EF introduces two additional temporal operators ($\tilde{\mathbf{X}}$, $\tilde{\mathbf{Y}}$) that evaluate to either true or false and two operators ($@\tilde{\mathbf{F}}$, $@\tilde{\mathbf{P}}$) that can be used in terms and whose evaluation type depends on their arguments. We define the last two operators, $@\tilde{\mathbf{F}}$ and $@\tilde{\mathbf{P}}$, such that their evaluation is always some value in the domain of their first argument.

4.1.2 Semantics

We now define the semantics of LTL-EF with respect to a time model τ, which is either discrete, dense, super-dense or super-discrete.

Let $\mathit{ordered}_\tau(t, t'', t')$ be a formula that holds for $t, t', t'' \in \tau$ if $[t, t'', t']$ is an ordered sequence of time points:

$$\mathit{ordered}_\tau(t, t'', t') \doteq \begin{cases} \mathbf{0} \leq t < t'' < t' & \text{if } \tau \in \{\mathit{discrete}, \mathit{dense}\}; \\ \exists i, r, i', r', r'' : 0 \leq i \leq i' \wedge 0 \leq r < r'' \leq r' \wedge \\ \quad t = \langle i, r \rangle \wedge t' = \langle i', r' \rangle \wedge t'' = \langle i, r'' \rangle & \text{otherwise.} \end{cases}$$

The formula $\forall t' > t \; \exists t'' : \mathit{ordered}_\tau(t, t'', t')$, requires the existence of a intermediate time point t'' for any successor t' of t. It describes the time points t that are followed by a time-elapse transition. In particular, the formula is always false in the discrete time model, in fact for $t' \doteq t + 1$ there is no t'' in between. In the dense case $t, t' \in \mathbb{R}_0^+$, hence for every two points t and t' there always exists some point t'' in between, e.g. $\frac{t+t'}{2}$. Consider now the super-dense and super-discrete time models and let $t \doteq \langle i, r \rangle$. The universal quantification requires the formula to hold for every $t' > t$. If τ admits a successor $t' \doteq \langle i + 1, r \rangle$ for t, the formula is false, since there is no r'' such that $r < r'' \leq r$. Otherwise, τ admits a successor along the second dimension of the time model. In the super-dense case the formula implies the existence of a dense sequence of successors for t. Let $t' \doteq \langle i, r' \rangle$, with $r' > r$, be one time point in the dense interval, then $t'' \doteq t'$ satisfies the formula. In the super-discrete case there is no dense sequence and the smallest time point is $t' \doteq \langle i, r + 1 \rangle$, again we can define $t'' \doteq t'$ and the formula holds.

Given a path $\sigma \doteq \langle M, \tau, \overline{\mu} \rangle$, a time point $t \in \tau$ and a LTL-EF formula φ, we define $\sigma, t \models \varphi$ recursively as follows. The semantics of the Boolean connectives and temporal modalities is defined as the one reported in §2.7.2 for LTL. In the following we report the semantics of the operators introduced

by LTL-EF.

- $\sigma, t \models \tilde{\mathbf{X}}\varphi$ iff $\forall t' \in \tau : t' > t$, $\exists t'' \in \tau$ such that $ordered_\tau(t, t'', t')$ holds and $\sigma, t'' \models \varphi$;
- $\sigma, t \models \tilde{\mathbf{Y}}\varphi$ iff $t > \mathbf{0}$ and $\forall t' \in \tau : t' < t$, $\exists t'' \in \tau$ such that $ordered_\tau(t', t'', t)$ holds and $\sigma, t'' \models \varphi$;
- $\sigma(t)(u@\tilde{\mathbf{F}}(\varphi)) = \begin{cases} \sigma(t)(u) & \text{if } \sigma, t \models \tilde{\mathbf{X}}\varphi; \\ \sigma(t')(u) & \text{if } \exists t' > t : \sigma, t' \models \varphi \vee \tilde{\mathbf{X}}\varphi \text{ and} \\ & \forall t'' \in \tau : t < t'' < t' \rightarrow \sigma, t'' \not\models \varphi; \\ \sigma(t)(def_{u@\tilde{\mathbf{F}}(\varphi)}) & \text{otherwise;} \end{cases}$
- $\sigma(t)(u@\tilde{\mathbf{P}}(\varphi)) = \begin{cases} \sigma(t)(u) & \text{if } \sigma, t \models \tilde{\mathbf{Y}}\varphi; \\ \sigma(t')(u) & \text{if } \exists t' < t : \sigma, t' \models \varphi \vee \tilde{\mathbf{Y}}\varphi \text{ and} \\ & \forall t'' \in \tau : t' < t'' < t \rightarrow \sigma, t'' \not\models \varphi; \\ \sigma(t)(def_{u@\tilde{\mathbf{P}}(\varphi)}) & \text{otherwise;} \end{cases}$

where $def_{u@\tilde{\mathbf{F}}(\varphi)}$ and $def_{u@\tilde{\mathbf{P}}(\varphi)}$ are fresh variables representing the default value of the expressions $u@\tilde{\mathbf{F}}(\varphi)$ and $u@\tilde{\mathbf{P}}(\varphi)$ respectively.

Given a trace σ, we say that a formula φ holds in the right [resp. left] open-interval of time point t iff $\sigma, t \models \tilde{\mathbf{X}}\varphi$ [resp. $\sigma, t \models \tilde{\mathbf{Y}}\varphi$].

We now highlight some subtleties of the semantics of $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$. Similarly to the $\mathbf{Y}$ modality, also $\tilde{\mathbf{Y}}$ is always false in the first state ($t = \mathbf{0}$). However, while $\mathbf{X}$ and $\mathbf{Y}$ are always false in the dense time model, their dense counterparts $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ are always false in the discrete time model. Therefore, in general, $\tilde{\mathbf{X}}\neg\varphi$ is not equivalent to $\neg\tilde{\mathbf{X}}\varphi$. The equivalence holds only when considering the dense time model. We can express a condition φ on a generic "next" state via the disjunction $(\mathbf{X}\varphi) \vee (\tilde{\mathbf{X}}\varphi)$ and the following equivalence holds:

$$\neg((\mathbf{X}\varphi) \vee (\tilde{\mathbf{X}}\varphi)) \equiv (\mathbf{X}\neg\varphi) \vee (\tilde{\mathbf{X}}\neg\varphi).$$

$\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ can be thought of as temporal operators that predicate on the right, resp. left, open-interval of the current time point. However, the interpretation of such open interval in the case of super-discrete time may not be straightforward. Consider the formula $\tilde{\mathbf{X}}\varphi$ and two time points $t \doteq \langle i, r\rangle$ and t' such that $t < t'$ and $v(t') - v(t) > 0$. In super-discrete time this implies $v(t') - v(t) \geq 1$ and we can define $t'' \doteq \langle i, r+1\rangle$ as the immediate successor of t along time. t and t'' are such that there exists no intermediate time point $\bar{t}$, $t < \bar{t} < t''$. Therefore, the semantics of $\tilde{\mathbf{X}}$ in the super-discrete case ensures that $\tilde{\mathbf{X}}\varphi$ holds at t iff φ holds at its immediate time-successor t''. A similar argument holds for $\tilde{\mathbf{Y}}$. $\tilde{\mathbf{Y}}\varphi$ holds at $t \doteq \langle i, r\rangle$ iff $r \geq 1$ and φ holds at its immediate time-predecessor $\langle i, r-1\rangle$.

Abbreviations. We define shorthand notations for the strict and non-strict versions of the temporal operators. We refer to the strict temporal modalities also with counting as follows.

$$\tilde{\mathbf{F}}\varphi \doteq (\tilde{\mathbf{X}}\mathbf{F}\varphi) \vee (\mathbf{X}\mathbf{F}\varphi); \qquad \tilde{\mathbf{P}}\varphi \doteq (\tilde{\mathbf{Y}}\mathbf{P}\varphi) \vee (\mathbf{Y}\mathbf{P}\varphi);$$
$$\tilde{\mathbf{F}}^1\varphi \doteq \tilde{\mathbf{F}}\varphi; \qquad \tilde{\mathbf{P}}^1\varphi \doteq \tilde{\mathbf{P}}\varphi;$$
$$\tilde{\mathbf{F}}^k\varphi \doteq \tilde{\mathbf{F}}(\varphi \wedge \tilde{\mathbf{F}}^{k-1}\varphi); \qquad \tilde{\mathbf{P}}^k\varphi \doteq \tilde{\mathbf{P}}(\varphi \wedge \tilde{\mathbf{P}}^{k-1}\varphi);$$
$$\tilde{\mathbf{G}}\varphi \doteq \neg\tilde{\mathbf{F}}\neg\varphi; \qquad \tilde{\mathbf{H}}\varphi \doteq \neg\tilde{\mathbf{P}}\neg\varphi;$$
$$\tilde{\mathbf{G}}^k\varphi \doteq \neg\tilde{\mathbf{F}}^k\neg\varphi; \qquad \tilde{\mathbf{H}}^k\varphi \doteq \neg\tilde{\mathbf{P}}^k\neg\varphi.$$

With respect to the event-freezing functions we define:

$$u@\mathbf{F}(\varphi) \doteq \mathbf{Ite}(\varphi, u, u@\tilde{\mathbf{F}}(\varphi)); \qquad u@\mathbf{P}(\varphi) \doteq \mathbf{Ite}(\varphi, u, u@\tilde{\mathbf{P}}(\varphi));$$
$$u@\tilde{\mathbf{F}}^1(\varphi) \doteq u@\tilde{\mathbf{F}}(\varphi); \qquad u@\tilde{\mathbf{P}}^1(\varphi) \doteq u@\tilde{\mathbf{P}}(\varphi);$$
$$u@\tilde{\mathbf{F}}^k(\varphi) \doteq (u@\tilde{\mathbf{F}}(\varphi))@\tilde{\mathbf{F}}^{k-1}(\varphi); \qquad u@\tilde{\mathbf{P}}^k(\varphi) \doteq (u@\tilde{\mathbf{P}}(\varphi))@\tilde{\mathbf{P}}^{k-1}(\varphi).$$

Finally, $next(u)$ can be defined as an abbreviation of $u@\tilde{\mathbf{F}}(\top)$. This definition, with respect to the one reported in §2.7.2, assigns a semantics to the operator in every time model and not only in the discrete one.

4.1.3 Next Occurrence in Dense and Super-Dense Time

We now describe some subtleties that arise when considering dense and super-dense time models. A well-known and intuitive fact about LTL is that in the discrete-time setting $\mathbf{F}\varphi \equiv \neg\varphi\mathbf{U}\varphi$. The equivalence states that if there is a point $t_{\mathbf{F}}$ in the future in which φ holds, then there is a (possibly empty) sequence of points in which φ does not hold before reaching a point $t_{\mathbf{U}} \leq t_{\mathbf{F}}$ in which φ holds and vice versa. Therefore, in the discrete-time case $\neg\varphi\mathbf{U}\varphi$ characterises the first time point in which φ holds. However, this is not the case in dense-time settings (dense or super-dense time models) [35]. In fact, φ can hold in a left-open interval I. Any time point in such interval will have an unbounded number of predecessors in which φ holds as well. In fact, any time value $v \in I$ must be greater than the lower bound of I, $v > l(I)$, and the interval $(l(I), v) \subset \mathbb{R}_0^+$ is a nonempty dense set.

We solve this issue by employing $\tilde{\mathbf{X}}\varphi$ to identify the time points in which φ holds in a left-open interval whose lower bound is the current time point.

$$\varphi_0\mathbf{U}_{\mathbf{C}}\varphi_1 \doteq \varphi_0\mathbf{U}(\varphi_1 \vee (\varphi_0 \wedge \tilde{\mathbf{X}}\varphi_1))$$

Using this definition we are guaranteed that there exists a minimum time point satisfying the left-hand-side of the $\mathbf{U}$ and, assuming finite variability, the following equivalence holds.

$$\mathbf{F}\varphi \equiv (\neg\varphi)\mathbf{U}_{\mathbf{C}}\varphi$$

4.1.4 LTL-EF with Explicit Time

We now extend LTL-EF with an explicit notion of time and define XLTL-EF. XLTL-EF has an additional symbol *time* that represents the time elapsed from the initial state. We restrict the use of this symbol such that it can be compared only with constants. We define the syntax of XLTL-EF as

follows.

$$
\begin{aligned}
\varphi &:: p(u,\ldots,u) \mid tu \bowtie cu \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid \\
&\quad \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi\mathbf{U}\varphi \mid \mathbf{Y}\varphi \mid \mathbf{H}\varphi \mid \mathbf{P}\varphi \mid \varphi\mathbf{S}\varphi \mid \tilde{\mathbf{X}}\varphi \mid \tilde{\mathbf{Y}}\varphi \\
u &:: c \mid x \mid f(u,\ldots,u) \mid u@\tilde{\mathbf{F}}(\varphi) \mid u@\tilde{\mathbf{P}}(\varphi) \mid \mathbf{Ite}(\varphi,u,u) \\
cu &:: c \mid f(cu,\ldots,cu) \mid cu@\tilde{\mathbf{F}}(\varphi) \mid cu@\tilde{\mathbf{P}}(\varphi) \mid \mathbf{Ite}(\varphi,cu,cu) \\
tu &:: \mathit{time} \mid \mathit{time}@\tilde{\mathbf{F}}(\varphi) \mid \mathit{time}@\tilde{\mathbf{P}}(\varphi) \mid \\
&\quad \mathit{time}@\tilde{\mathbf{F}}(\varphi) - \mathit{time} \mid \mathit{time} - \mathit{time}@\tilde{\mathbf{P}}(\varphi) \\
\bowtie &:: \leq \mid < \mid = \mid \neq \mid > \mid \geq
\end{aligned}
$$

The semantic of XLTL-EF simply extends the one of LTL-EF by setting $\sigma(t)(\mathit{time}) \doteq v(t)$, for every trace σ and time point t of σ.

In addition, we will write $\mathit{time\_until}(\varphi)$ for $\mathit{time}@\tilde{\mathbf{F}}(\varphi) - \mathit{time}$ and $\mathit{time\_since}(\varphi)$ for $\mathit{time} - \mathit{time}@\tilde{\mathbf{P}}(\varphi)$.

4.2 MTL with Counting Operators

In this section we define MTLC as an extension of LTL-EF with bounds on the temporal modalities. MTLC can also be seen as an extension of MTL along the following directions: (i) it decorates temporal modalities with parametric intervals; (ii) it allows for counting operators that predicate about the number of times some event happens in a given interval and (iii) it adds the temporal modalities we introduced for LTL-EF: $\tilde{\mathbf{X}}$, $\tilde{\mathbf{Y}}$, $@\tilde{\mathbf{F}}$ and $@\tilde{\mathbf{P}}$.

4.2.1 Syntax

We define the syntax of a MTLC formula φ as follows.

$$
\begin{aligned}
\varphi :: \; & p(u,\ldots,u) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid \\
& \mathbf{X}\varphi \mid \tilde{\mathbf{X}}\varphi \mid \mathbf{Y}\varphi \mid \tilde{\mathbf{Y}}\varphi \mid \\
& \mathbf{G}_I\varphi \mid \mathbf{F}_I\varphi \mid \varphi\mathbf{U}_I\varphi \mid \mathbf{H}_I\varphi \mid \mathbf{P}_I\varphi \mid \varphi\mathbf{S}_I\varphi \mid \overrightarrow{\mathbf{C}}^k_{[0,cu)}\varphi \mid \overleftarrow{\mathbf{C}}^k_{[0,cu)}\varphi \\
u :: \; & c \mid x \mid f(u,\ldots,u) \mid u@\tilde{\mathbf{F}}(\varphi) \mid u@\tilde{\mathbf{P}}(\varphi) \mid \mathbf{Ite}(\varphi,u,u) \\
I :: \; & [cu,cu] \mid [cu,cu) \mid (cu,cu] \mid (cu,cu) \mid [cu,\infty) \mid (cu,\infty) \\
cu :: \; & c \mid f(cu,\ldots,cu)
\end{aligned}
$$

where x, p, f and c are a variable in V, a predicate symbol, a function symbol and a constant in Σ respectively. cu is a Σ-term that does not contain any variable. Therefore, the bounds on the temporal modalities are rigid and may contain parameters, i.e. their interpretation cannot change with time. For a temporal operator $\mathcal{O} \in \{\mathbf{U}, \mathbf{F}, \mathbf{G}, \mathbf{S}, \mathbf{P}, \mathbf{H}\}$, we will also use the following abbreviations for the intervals: $\mathcal{O}_{=cu}$ for $\mathcal{O}_{[cu,cu]}$, $\mathcal{O}_{\leq cu}$ for $\mathcal{O}_{[0,cu]}$, $\mathcal{O}_{<cu}$ for $\mathcal{O}_{[0,cu)}$, $\mathcal{O}_{>cu}$ for $\mathcal{O}_{(cu,\infty)}$ and $\mathcal{O}_{\geq cu}$ for $\mathcal{O}_{[cu,\infty)}$.

4.2.2 Semantics

Given a path $\sigma \doteq \langle M, \tau, \overline{\mu} \rangle$, a time point $t \in \tau$ and a MTLC formula φ, we define $\sigma, t \models \varphi$ recursively as follows. We define the semantics for the bounded version of the temporal modalities; the semantics of all other operators is the one reported in Sec. 4.1 for LTL-EF and is not repeated here.

- $\sigma, t \models \varphi_0 \mathbf{U}_I \varphi_1$ iff $\exists t' \in \tau : t' \geq t \wedge v(t') - v(t) \in M(I)$ such that $\sigma, t' \models \varphi_1$ and $\forall t'' \in \tau : t \leq t'' < t'$ then $\sigma, t'' \models \varphi_0$;
- $\sigma, t \models \varphi_0 \mathbf{S}_I \varphi_1$ iff $\exists t' \in \tau : t' \leq t \wedge v(t) - v(t') \in M(I)$ such that $\sigma, t' \models \varphi_1$ and $\forall t'' \in \tau : t' < t'' \leq t$ then $\sigma, t'' \models \varphi_0$;

- $\sigma, t \models \overrightarrow{\mathbf{C}}^k_{<cu}\varphi$ iff $\exists t_1, \ldots t_k \in \tau : t < t_1 < \ldots < t_k \wedge v(t_k) - v(t) < M(cu)$ such that $\forall i \in \{j\}_{j=1}^k : \sigma, t_i \models \varphi$;
- $\sigma, t \models \overleftarrow{\mathbf{C}}^k_{<cu}\varphi$ iff $\exists t_1, \ldots t_k \in \tau : \mathbf{0} \leq t_k < \ldots < t_1 < t \wedge v(t) - v(t_k) < M(cu)$ such that $\forall i \in \{j\}_{j=1}^k : \sigma, t_i \models \varphi$.

Where $M(I)$ is the set obtained from I by substituting the terms at the endpoints with their interpretation.

The semantics of the other temporal operators is defined in terms of the ones above as in the LTL case.

$$\sigma, t \models \mathbf{F}_I\varphi \text{ iff } \sigma, t \models \top\mathbf{U}_I\varphi; \qquad \sigma, t \models \mathbf{G}_I\varphi \text{ iff } \sigma, t \models \neg(\top\mathbf{U}_I\neg\varphi);$$
$$\sigma, t \models \mathbf{P}_I\varphi \text{ iff } \sigma, t \models \top\mathbf{S}_I\varphi; \qquad \sigma, t \models \mathbf{H}_I\varphi \text{ iff } \sigma, t \models \neg(\top\mathbf{S}_I\neg\varphi).$$

We remark that the bounds on the temporal operators are to be interpreted as a distance with respect to the time dimension when considering super-dense and super-discrete time models. Instead, in the discrete and dense time models the distance is interpreted along the only dimension they prescribe, i.e. the difference between the time points. We highlight that if we consider the discrete time model with $v(i) = i$, i.e. a discrete system in which the time value increases by one at every step, then the time intervals specified in the MTL formulae correspond to the number of discrete steps that the system performs. Other definitions of v still describe systems with monotonically increasing discrete time, but define a different mapping between the number of steps and the corresponding time value. For example, $v(0) = 0$ and $v(i) = v(i-1) + \delta$ for some $\delta \in \mathbb{R}_0^+$ describes a system with discrete time where δ is the discretization factor and every step corresponds to a time elapse of δ. In addition, we highlight the significance of this definition of MTL by pointing out that the intervals are specified using expressions that can contain parameters, not only constants. For this reason it is not possible to rewrite such operators via a simple nesting of $\mathbf{X}$ operators.

Finally, notice that for $I \doteq [0, \infty)$ the semantics of every bounded temporal operator is equivalent to the semantics of its unbounded counterpart, since every evaluation of the time points will always lie in the interval. Therefore the following equivalences hold:

$$\varphi_0 \mathbf{U}_{\geq 0} \varphi_1 \equiv \varphi_0 \mathbf{U} \varphi_1; \qquad \varphi_0 \mathbf{S}_{\geq 0} \varphi_1 \equiv \varphi_0 \mathbf{S} \varphi_1.$$

We define $\mathrm{MTLC}_{0,\infty}$ as the fragment of MTLC where the syntax of the intervals is restricted to:

$$I :: \ [0, cu] \mid [0, cu) \mid (0, cu] \mid (0, cu) \mid [cu, \infty) \mid (cu, \infty)$$

Therefore, $\mathrm{MTLC}_{0,\infty}$ allows only intervals such that the lower bound is 0 or the upper bound is ∞.

4.2.3 $\mathrm{MTLC}_{0,\infty}$ Examples

We now provide some examples of $\mathrm{MTLC}_{0,\infty}$ formulae interpreted over different time models and motivate their satisfiability or validity.

Bounded Finally. The formula $\mathbf{G}(\mathbf{X}b \rightarrow \mathbf{F}_{=0}b)$ states that it is always the case that if we reach a state in which b holds via an instantaneous transition, then we also reach a state in which b holds in 0 time. The formula is not valid if we consider the discrete time model. Consider a trace such that b is false at time point t and holds at $t+1$. The distance between these time points is $v(t+1) - v(t) > 0$, hence $\mathbf{F}_{=0}b$ holds in $t+1$ but not in t.

In all other time models (dense, super-dense and super-discrete) the formula is valid. In the dense time model $\mathbf{X}b$ is always false, hence the implication is valid. In the super-discrete and super-dense time models, let $t = \langle i, r \rangle \in \tau$ such that $\mathbf{X}b$ holds in t; then, by definition of $\mathbf{X}$, b holds in the time point $t' \doteq \langle i+1, r \rangle$. The distance between t and t' is defined as $v(t') - v(t) = r - r = 0$, therefore the implication holds.

Bounded Globally. The formula $(\mathbf{G}_{\leq k} b \wedge \mathbf{G}_{\geq k} b) \leftrightarrow \mathbf{G} b$, where k is a parameter, states that b holds at every time up to k (included) and from k to infinity, iff b always holds. The formula is valid in all time models since the union of the two intervals is exactly $[0, \infty)$.

Bounded Once. The formula $\mathbf{G}(\mathbf{P}_{=0} b \rightarrow b)$ states that if no time has passed since that last time we observed b, then b must hold at the current time point. The formula is valid only if we consider the dense or discrete time models. In fact, these models are strictly monotonic. Therefore, the distance between any two consecutive time points t, t' is strictly positive ($v(t') > v(t)$). For this reason, $\mathbf{P}_{=0} b$ holds at the current time point iff b does. In the other cases (super-discrete and super-dense) the time model is not strictly monotonic and allows for consecutive time points t and t' such that $v(t) = v(t')$. Therefore, if b holds in t' then $\mathbf{P}_{=0} b$ holds in t, but b can be false in t.

4.3 Reduction to LTL Model Checking on ITS

In this section we show how the model checking problem of $\mathrm{MTLC}_{0,\infty}$ and XLTL-EF on timed models with dense, super-dense or super-discrete time model can be reduced to LTL model checking on a "discrete" system, i.e. with discrete time model.

First, in §4.3.1, we show how $\mathrm{MTLC}_{0,\infty}$ formulae can be equivalently written in XLTL-EF. Then, in §4.3.2, we define for every XLTL-EF formula over dense, super-dense or super-discrete time model a corresponding equisatisfiable XLTL-EF formula with discrete time model. Finally, in §4.3.3, we describe how, in discrete time model, a XLTL-EF formula can be written into a equisatisfiable LTL formula by rewriting the event-freezing functions using an additional fresh variable and the *next* operator.

4.3.1 $\mathrm{MTLC}_{0,\infty}$ to XLTL-EF

Now, we provide a sequence of global equivalences between $\mathrm{MTLC}_{0,\infty}$ and XLTL-EF formulae. The global equivalences allow the recursive rewriting of the $\mathrm{MTLC}_{0,\infty}$ formula into a (globally) equivalent XLTL-EF one. The simple equivalence relations would not allow such recursive rewriting because it would only guarantee equivalence at the first time point.

$$\varphi_1 \mathbf{U}_{(0,p)} \varphi_2 \equiv_G \varphi_1 \mathbf{U}(\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time < p); \tag{4.1}$$

$$\varphi_1 \mathbf{S}_{(0,p)} \varphi_2 \equiv_G \varphi_1 \mathbf{S}(\tilde{\mathbf{Y}}\varphi_1 \mathbf{S}\varphi_2 \wedge time - time@\tilde{\mathbf{P}}(\varphi_2) < p); \tag{4.2}$$

$$\varphi_1 \mathbf{U}_{[0,p)} \varphi_2 \equiv_G \varphi_1 \mathbf{U}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time < p; \tag{4.3}$$

$$\varphi_1 \mathbf{S}_{[0,p)} \varphi_2 \equiv_G \varphi_1 \mathbf{S}\varphi_2 \wedge time - time@\mathbf{P}(\varphi_2) < p; \tag{4.4}$$

$$\overrightarrow{\mathbf{C}}^k_{[0,p)}(\varphi) \equiv_G time@\tilde{\mathbf{F}}^k(\varphi) - time < p \wedge \tilde{\mathbf{F}}^k(\varphi); \tag{4.5}$$

$$\overleftarrow{\mathbf{C}}^k_{[0,p)}(\varphi) \equiv_G time - time@\tilde{\mathbf{P}}^k(\varphi) < p \wedge \tilde{\mathbf{P}}^k(\varphi); \tag{4.6}$$

$$\begin{aligned} \varphi_1 \mathbf{U}_{(0,p]} \varphi_2 \equiv_G\ & \varphi_1 \mathbf{U}((\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2) \wedge \\ & (((\tilde{\mathbf{X}}\neg\varphi_2 \mathbf{U}\varphi_2) \wedge (time@\tilde{\mathbf{F}}(\varphi_2) - time \leq p)) \vee \\ & ((\tilde{\mathbf{X}}\neg\varphi_2 \mathbf{U}\tilde{\mathbf{X}}\varphi_2) \wedge (time@\tilde{\mathbf{F}}(\varphi_2) - time < p)))); \end{aligned} \tag{4.7}$$

$$\begin{aligned} \varphi_1 \mathbf{S}_{(0,p]} \varphi_2 \equiv_G\ & \varphi_1 \mathbf{S}((\tilde{\mathbf{Y}}\varphi_1 \mathbf{S}\varphi_2) \wedge \\ & (((\tilde{\mathbf{Y}}\neg\varphi_2 \mathbf{S}\varphi_2) \wedge (time - time@\tilde{\mathbf{P}}(\varphi_2) \leq p)) \vee \\ & ((\tilde{\mathbf{Y}}\neg\varphi_2 \mathbf{S}\tilde{\mathbf{Y}}\varphi_2) \wedge (time - time@\tilde{\mathbf{P}}(\varphi_2) < p)))); \end{aligned} \tag{4.8}$$

$$\begin{aligned} \varphi_1 \mathbf{U}_{[0,p]} \varphi_2 \equiv_G\ & \varphi_1 \mathbf{U}\varphi_2 \wedge \\ & (\neg\varphi_2 \mathbf{U}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time \leq p \vee \\ & \neg\varphi_2 \mathbf{U}\tilde{\mathbf{X}}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time < p); \end{aligned} \tag{4.9}$$

$$\begin{aligned} \varphi_1 \mathbf{S}_{[0,p]} \varphi_2 \equiv_G\ & \varphi_1 \mathbf{S}\varphi_2 \wedge \\ & (\neg\varphi_2 \mathbf{S}\varphi_2 \wedge time - time@\mathbf{P}(\varphi_2) \leq p \vee \\ & \neg\varphi_2 \mathbf{S}\tilde{\mathbf{Y}}\varphi_2 \wedge time - time@\mathbf{P}(\varphi_2) < p). \end{aligned} \tag{4.10}$$

The remaining metric operators can be rewritten in terms of the ones above as follows:

$$\varphi_1 \mathbf{U}_{(p,\infty)} \varphi_2 \equiv_G (p = 0 \wedge \varphi_1 \mathbf{U}_{\neq 0} \varphi_2) \vee (p > 0 \wedge \mathbf{G}_{[0,p]}(\varphi_1 \wedge \varphi_1 \mathbf{U}_{\neq 0} \varphi_2)); \tag{4.11}$$

$$\varphi_1 \mathbf{S}_{(p,\infty)} \varphi_2 \equiv_G (p = 0 \wedge \varphi_1 \mathbf{S}_{\neq 0} \varphi_2) \vee (p > 0 \wedge \mathit{time} > p \wedge H_{[0,p]}(\varphi_1 \wedge \varphi_1 \mathbf{S}_{\neq 0} \varphi_2)); \tag{4.12}$$

$$\varphi_1 \mathbf{U}_{[p,\infty)} \varphi_2 \equiv_G (p = 0 \wedge \varphi_1 \mathbf{U} \varphi_2) \vee (p > 0 \wedge \mathbf{G}_{[0,p)} \varphi_1 \wedge \mathbf{G}_{[0,p]} \mathbf{P}_{=0}(\tilde{\mathbf{H}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{U} \varphi_2)); \tag{4.13}$$

$$\varphi_1 \mathbf{S}_{[p,\infty)} \varphi_2 \equiv_G (p = 0 \wedge \varphi_1 \mathbf{S} \varphi_2) \vee (p > 0 \wedge \mathit{time} \geq p \wedge \mathbf{H}_{[0,p)} \varphi_1 \wedge \mathbf{H}_{[0,p]} \mathbf{F}_{=0}(\tilde{\mathbf{G}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{S} \varphi_2)). \tag{4.14}$$

Where $\varphi_1 \mathbf{U}_{\neq 0} \varphi_2$ and $\varphi_1 \mathbf{S}_{\neq 0} \varphi_2$ are used as abbreviations of $\varphi_1 \mathbf{U} \tilde{\mathbf{X}}(\varphi_1 \mathbf{U} \varphi_2)$ and $\varphi_1 \mathbf{S} \tilde{\mathbf{Y}}(\varphi_1 \mathbf{S} \varphi_2)$ respectively.

We highlight the use of $\mathbf{P}_{=0}$ and $\mathbf{F}_{=0}$ in equations (4.13) and (4.14). In the discrete and dense time models they can be simplified away. In fact, these time models are monotonically increasing, hence $\mathbf{P}_{=0}\varphi$ and $\mathbf{F}_{=0}\varphi$ hold at some time point iff φ does. Instead, due to instantaneous transitions, in the super-discrete and super-dense time models there can be multiple time points associated with the same time value. For example, consider the case in which the formula φ_2 holds exactly after p time. Then, there exists one of the time points associated with the current time plus p in which φ_2 holds and all its predecessors [resp. successors] satisfy φ_1. We express these conditions via the $\mathbf{F}_{=0}$ and $\mathbf{P}_{=0}$ operators.

The following theorem proves the correctness of these global equivalences, hence it allows the rewriting of $\mathrm{MTLC}_{0,\infty}$ formulae into XLTL-EF formulae.

Theorem 1 - MTLC$_{0,\infty}$ to XLTL-EF
Every MTLC$_{0,\infty}$ *formula can be equivalently written as a* XLTL-EF *formula.*

Proof. We show that all the global equivalences (4.1)-(4.14) are correct. Let M be the first-order structure of trace σ and $M(p)$ the evaluation of p in σ. We first consider the future cases.
We prove Equivalence (4.5) by induction on k.

- If $k = 1$: $\overrightarrow{\mathbf{C}}^k_{[0,p)}(\varphi) \equiv_G \tilde{\mathbf{F}}_{[0,p)}\varphi \equiv_G \tilde{\mathbf{F}}^k(\varphi) \wedge time@\tilde{\mathbf{F}}^k(\varphi) - time < p$.
- Assume the equivalence holds for $k-1$. $\sigma, t \models \overrightarrow{\mathbf{C}}^k_{[0,p)}$ iff there exist $t_1, \ldots, t_k$ with $t < t_1 < \ldots < t_k$, $v(t_k) - v(t) < M(p)$, such that for all $i \in \{j\}_{j=1}^k$, $\sigma, t_i \models \varphi$. Let $t' = t_{k-1}$ and $p' = v(t_k) - v(t_{k-1})$. Then, $\sigma, t \models \overrightarrow{\mathbf{C}}^k_{[0,p)}$ iff there exists t' such that $v(t') - v(t) = p'$ and $\sigma, t' \models \varphi \wedge \overrightarrow{\mathbf{C}}^{k-1}_{[0,p-p')}\varphi$. By induction, $\sigma, t \models \overrightarrow{\mathbf{C}}^k_{[0,p)}$ iff there exists t' such that $v(t') - v(t) = p'$ and $\sigma, t' \models \varphi \wedge \tilde{\mathbf{F}}^{k-1}(\varphi) \wedge time@\tilde{\mathbf{F}}^{k-1}(\varphi) - time < p - p'$, thus iff $\sigma, t \models \varphi \wedge \tilde{\mathbf{F}}^k(\varphi) \wedge time@\tilde{\mathbf{F}}^k(\varphi) - time < p$.

Equivalence (4.1)

- $\varphi_1 \mathbf{U}_{(0,p)}\varphi_2 \models_G \varphi_1 \mathbf{U}(\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time < p)$.
  If $\sigma, t \models \varphi_1 \mathbf{U}_{(0,p)}\varphi_2$ then there exists $t' < t$ such that $0 < v(t') - v(t) < M(p)$, $\sigma, t' \models \varphi_2$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. Let t_0 be the greatest time point such that $t \leq t_0$ and $v(t) = v(t_0)$. Thus, for all t'', $t \leq t'' \leq t_0$, $\sigma, t \models \varphi_1$. Moreover, there exists $\bar{t}'$, $t_0 < \bar{t}' \leq t'$, $\sigma, \bar{t}' \models \varphi_2$, $v(\bar{t}') - v(t_0) < M(p)$, and for all t'', $t_0 \leq t'' < \bar{t}'$, $\sigma, t'' \models \neg\varphi_2$. Thus, $\sigma, t_0 \models \tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time < p$ and $\sigma, t \models \varphi_1 \mathbf{U}(\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time < p)$.

- $\varphi_1 \mathbf{U}(\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time < p) \models_G \varphi_1 \mathbf{U}_{(0,p)}\varphi_2$.
  If $\sigma, t \models \varphi_1 \mathbf{U}(\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time < p)$, then there exists t', $t \leq t'$, $\sigma, t' \models \tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time < p$ and for all t'',

$t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. Therefore, there exists $\bar{t}' > t'$, such that $\sigma, \bar{t}' \models \varphi_2$, $0 < v(\bar{t}') - v(t') < M(p)$, and for all t'', $t' < t'' < \bar{t}'$, $\sigma, t'' \models \varphi_1$. Thus $\sigma, t \models \varphi_1 \mathbf{U}_{(0,p)} \varphi_2$.

Equivalence (4.3)

- $\varphi_1 \mathbf{U}_{[0,p)} \varphi_2 \models_G \varphi_1 \mathbf{U} \varphi_2 \wedge \mathit{time}@\mathbf{F}(\varphi_2) - \mathit{time} < p$.
  If $\sigma, t \models \varphi_1 \mathbf{U}_{[0,p)} \varphi_2$ then there exists $t' \geq t$ such that $v(t') - v(t) < M(p)$, $\sigma, t' \models \varphi_2$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. Thus, $\sigma, t \models \varphi_1 \mathbf{U} \varphi_2$. Moreover, there exists $\bar{t}'$, $t \leq \bar{t}' \leq t'$, $\sigma, \bar{t}' \models \varphi_2 \vee \tilde{\mathbf{X}} \varphi_2$, $v(\bar{t}') - v(t) < M(p)$, and for all t'', $t \leq t'' < \bar{t}'$, $\sigma, t'' \models \neg \varphi_2$. Thus, $\sigma, t \models \mathit{time}@\mathbf{F}(\phi_2) - \mathit{time} < p$.

- $\varphi_1 \mathbf{U} \varphi_2 \wedge \mathit{time}@\mathbf{F}(\varphi_2) - \mathit{time} < p \models_G \varphi_1 \mathbf{U}_{[0,p)} \varphi_2$.
  If $\sigma, t \models \varphi_1 \mathbf{U} \varphi_2$, then there exists t', $t \leq t'$, $\sigma, t' \models \varphi_2 \vee \tilde{\mathbf{X}} \varphi_2$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \neg \varphi_2$. If $\sigma, t \models \mathit{time}@\mathbf{F}(\varphi_2) - \mathit{time} < p$, then $v(t') - v(t) < M(p)$. Thus, $\sigma, t \models \varphi_1 \mathbf{U}_{[0,p)} \varphi_2$.

Equivalence (4.7)

- $\varphi_1 \mathbf{U}_{(0,p]} \varphi_2 \models_G \varphi_1 \mathbf{U}((\tilde{\mathbf{X}} \varphi_1 \mathbf{U} \varphi_2) \wedge (((\tilde{\mathbf{X}} \neg \varphi_2 \mathbf{U} \varphi_2) \wedge (\mathit{time}@\tilde{\mathbf{F}}(\varphi_2) - \mathit{time} \leq p)) \vee ((\tilde{\mathbf{X}} \neg \varphi_2 \mathbf{U} \tilde{\mathbf{X}} \varphi_2) \wedge (\mathit{time}@\tilde{\mathbf{F}}(\varphi_2) - \mathit{time} < p))))$.
  If $\sigma, t \models \varphi_1 \mathbf{U}_{(0,p]} \varphi_2$ then there exists $t' > t$ such that $0 < v(t') - v(t) \leq M(p)$, $\sigma, t' \models \varphi_2$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. Let t_0 be the greatest point such that $t \leq t_0$ and $v(t) = v(t_0)$. Therefore, for all t'', $t \leq t'' \leq t_0$, $\sigma, t \models \varphi_1$. Moreover, $\sigma, t_0 \models \tilde{\mathbf{X}} \varphi_1 \mathbf{U} \varphi_2$. Thus, either $\sigma, t_0 \models \tilde{\mathbf{X}} \neg \varphi_2 \mathbf{U} \varphi_2$ or $\sigma, t_0 \models \tilde{\mathbf{X}} \neg \varphi_2 \mathbf{U} \tilde{\mathbf{X}} \varphi_2$. In the first case, there exists $\bar{t}'$, $t_0 < \bar{t}' \leq t'$, $\sigma, \bar{t}' \models \varphi_2$, and for all t'', $t_0 \leq t'' < \bar{t}'$, $\sigma, t'' \models \neg \varphi_2$. Since $\bar{t}' \leq t'$, $\sigma, t_0 \models \mathit{time}@\tilde{\mathbf{F}}(\varphi_2) - \mathit{time} \leq p$. Similarly, in the second case, there exists $\bar{t}'$, $t_0 < \bar{t}' \leq t'$, $\sigma, \bar{t}' \models \tilde{\mathbf{X}} \varphi_2$, and for all t'', $t_0 \leq t'' < \bar{t}'$, $\sigma, t'' \models \neg \varphi_2$. Since $\bar{t}' \leq t'$, $\sigma, t_0 \models \mathit{time}@\tilde{\mathbf{F}}(\varphi_2) - \mathit{time} \leq p$. Thus, $\sigma, t \models \varphi_1 \mathbf{U}((\tilde{\mathbf{X}} \varphi_1 \mathbf{U} \varphi_2) \wedge (((\neg \varphi_2 \mathbf{U} \varphi_2) \wedge (\mathit{time}@\tilde{\mathbf{F}}(\varphi_2) - \mathit{time} \leq p)) \vee ((\neg \varphi_2 \mathbf{U} \tilde{\mathbf{X}} \varphi_2) \wedge (\mathit{time}@\tilde{\mathbf{F}}(\varphi_2) - \mathit{time} < p))))$.

- $\varphi_1 \mathbf{U}((\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2) \wedge (((\tilde{\mathbf{X}}\neg\varphi_2 \mathbf{U}\varphi_2) \wedge (time@\tilde{\mathbf{F}}(\varphi_2) - time \leq p)) \vee ((\tilde{\mathbf{X}}\neg\varphi_2 \mathbf{U}\tilde{\mathbf{X}}\varphi_2) \wedge (time@\tilde{\mathbf{F}}(\varphi_2) - time < p)))) \models_G \varphi_1 \mathbf{U}_{(0,p]}\varphi_2$.
  If $\sigma, t \models \varphi_1 \mathbf{U}((\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2) \wedge (((\neg\varphi_2 \mathbf{U}\varphi_2) \wedge (time@\tilde{\mathbf{F}}(\varphi_2) - time \leq p)) \vee ((\neg\varphi_2 \mathbf{U}\tilde{\mathbf{X}}\varphi_2) \wedge (time@\tilde{\mathbf{F}}(\varphi_2) - time < p))))$ then there exists t', $t \leq t'$, $\sigma, t' \models (\tilde{\mathbf{X}}\varphi_1 \mathbf{U}\varphi_2 \wedge (\neg\varphi_2 \mathbf{U}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time \leq p \vee \neg\varphi_2 \mathbf{U}\tilde{\mathbf{X}}\varphi_2 \wedge time@\tilde{\mathbf{F}}(\varphi_2) - time < p))$ and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. Thus, there exists $\bar{t}' > t'$, such that $\sigma, \bar{t}' \models \varphi_2$, $0 < v(\bar{t}') - v(t') \leq M(p)$, and for all t'', $t' < t'' < \bar{t}'$, $\sigma, t'' \models \varphi_1$ and $\sigma, t \models \varphi_1 \mathbf{U}_{(0,p]}\varphi_2$.

Equivalence (4.9)

- $\varphi_1 \mathbf{U}_{[0,p]}\varphi_2 \models_G \varphi_1 \mathbf{U}\varphi_2 \wedge (\neg\varphi_2 \mathbf{U}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time \leq p \vee \neg\varphi_2 \mathbf{U}\tilde{\mathbf{X}}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time < p)$.
  If $\sigma, t \models \varphi_1 \mathbf{U}_{[0,p]}\varphi_2$ then there exists $t' \geq t$ such that $v(t') - v(t) \leq M(p)$, $\sigma, t' \models \varphi_2$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. Thus, $\sigma, t \models \varphi_1 \mathbf{U}\varphi_2$. Moreover, there exists $\bar{t}'$, $t \leq \bar{t}' \leq t'$, $\sigma, \bar{t}' \models \varphi_2 \vee \tilde{\mathbf{X}}\varphi_2$, and for all t'', $t \leq t'' < \bar{t}'$, $\sigma, t'' \models \neg\varphi_2$. Also, either $\sigma, \bar{t}' \models \varphi_2$ and $v(\bar{t}') - v(t) \leq M(p)$, or $\sigma, \bar{t}' \models \tilde{\mathbf{X}}\varphi_2$ and $v(\bar{t}') - v(t) < M(p)$. Thus, $\sigma, t \models (\varphi_1 \wedge \neg\varphi_2)\mathbf{U}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time \leq p \vee (\varphi_1 \wedge \neg\varphi_2)\mathbf{U}(\varphi_1 \wedge \tilde{\mathbf{X}}\varphi_2) \wedge time@\mathbf{F}(\varphi_2) - time < p$.

- $\varphi_1 \mathbf{U}\varphi_2 \wedge (\neg\varphi_2 \mathbf{U}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time \leq p \vee \neg\varphi_2 \mathbf{U}\tilde{\mathbf{X}}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time < p) \models_G \varphi_1 \mathbf{U}_{[0,p]}\varphi_2$.
  If $\sigma, t \models \varphi_1 \mathbf{U}\varphi_2$, then there exists t', $t \leq t'$, $\sigma, t' \models \varphi_2$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. If $\sigma, t \models \neg\varphi_2 \mathbf{U}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time \leq p$, then there exists $\bar{t}'$, $t \leq \bar{t}' \leq t'$, such that $\sigma, \bar{t}' \models \varphi_2$, and for all t'', $t \leq t'' < \bar{t}'$, $\sigma, t'' \models \neg\varphi_2$ and $v(\bar{t}') - v(t) \leq M(p)$. Since $\bar{t}' \leq t'$, $\sigma, t \models \phi_1 \mathbf{U}_{[0,p]}\varphi_2$. Similarly, if $\sigma, t \models \neg\varphi_2 \mathbf{U}\tilde{\mathbf{X}}\varphi_2 \wedge time@\mathbf{F}(\varphi_2) - time < p$, then there exists $\bar{t}'$, $t \leq \bar{t}' \leq t'$, $\sigma, \bar{t}' \models \tilde{\mathbf{X}}\varphi_2$, and for all t'', $t \leq t'' < \bar{t}'$, $\sigma, t'' \models \neg\varphi_2$ and $v(\bar{t}') - v(t) \leq M(p)$. Thus, $\sigma, t \models \varphi_1 \mathbf{U}_{[0,p]}\varphi_2$.

Equivalence (4.11)

- $\varphi_1 \mathbf{U}_{(p,\infty)} \varphi_2 \models_G (p = 0 \wedge \varphi_1 \mathbf{U}_{\neq 0} \varphi_2) \vee (p > 0 \wedge \mathbf{G}_{[0,p]}(\varphi_1 \wedge \varphi_1 \mathbf{U}_{\neq 0} \varphi_2))$.
  If $M(p) = 0$, then the equivalence is trivial. Suppose instead that $M(p) > 0$. If $\sigma, t \models \varphi_1 \mathbf{U}_{(p,\infty)} \varphi_2$, then there exists $t' > t$ such that $\sigma, t' \models \varphi_2$, $v(t') - v(t) > M(p)$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. For all $\bar{t}'' \geq t$, if $v(\bar{t}'') - v(t) \leq M(p)$, then $t' > \bar{t}''$. Thus, $\sigma, \bar{t}'' \models \varphi_1 \wedge \varphi_1 \mathbf{U}_{\neq 0} \varphi_2$.

- $(p = 0 \wedge \varphi_1 \mathbf{U}_{\neq 0} \varphi_2) \vee (p > 0 \wedge \mathbf{G}_{[0,p]}(\varphi_1 \wedge \varphi_1 \mathbf{U}_{\neq 0} \varphi_2)) \models_G \varphi_1 \mathbf{U}_{(p,\infty)} \varphi_2$.
  If $M(p) = 0$, then the equivalence is trivial. Suppose instead that $M(p) > 0$. If $\sigma, t \models \mathbf{G}_{[0,p]}(\varphi_1 \wedge \varphi_1 \mathbf{U}_{\neq 0} \varphi_2)$, then there exists t' such that $v(t') - v(t) = M(p)$ and $\sigma, t' \models \varphi_1 \mathbf{U}_{(0,\infty)} \varphi_2$. Moreover, for all t'', $t \leq t'' \leq t'$, $\sigma, t'' \models \varphi_1$. Thus, $\sigma, t \models \varphi_1 \mathbf{U}_{(p,\infty)} \varphi_2$.

Equivalence (4.13)

- $\varphi_1 \mathbf{U}_{[p,\infty)} \varphi_2 \models_G (p = 0 \wedge \varphi_1 \mathbf{U} \varphi_2) \vee (p > 0 \wedge \mathbf{G}_{[0,p)} \varphi_1 \wedge \mathbf{G}_{[0,p]} \mathbf{P}_{=0}(\tilde{\mathbf{H}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{U} \varphi_2))$.
  If $M(p) = 0$, then the equivalence is trivial. Suppose instead that $M(p) > 0$. If $\sigma, t \models \varphi_1 \mathbf{U}_{[p,\infty]} \varphi_2$, then there exists $t' > t$, such that $\sigma, t' \models \varphi_2$, $v(t') - v(t) \geq M(p)$, and for all t'', $t \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. For all $\bar{t}'' \geq t$, if $v(\bar{t}'') - v(t) < M(p)$, then $t' > \bar{t}''$. Thus, $\sigma, \bar{t}'' \models \varphi_1$. Moreover, if $v(t') - v(t) > M(p)$, then for all t'', $t \leq t''$, if $v(t'') - v(t) \leq M(p)$, then $\sigma, t'' \models \varphi_1 \mathbf{U} \varphi_2$ and thus, $\sigma, t'' \models \mathbf{P}_{=0}(\tilde{\mathbf{H}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{U} \varphi_2)$. Similarly, if $v(t') - v(t) = M(p)$, then for all t'', $t \leq t'' \leq t'$, then $\sigma, t'' \models \varphi_1 \mathbf{U} \varphi_2$ and thus, $\sigma, t'' \models \mathbf{P}_{=0}(\tilde{\mathbf{H}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{U} \varphi_2)$. Finally, for all $t'' > t'$, $v(t'') = v(t')$, $\sigma, t'' \models \mathbf{P}_{=0}(\tilde{\mathbf{H}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{U} \varphi_2)$.

- $(p = 0 \wedge \varphi_1 \mathbf{U} \varphi_2) \vee (p > 0 \wedge \mathbf{G}_{[0,p)} \varphi_1 \wedge \mathbf{G}_{[0,p]} \mathbf{P}_{=0}(\tilde{\mathbf{H}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{U} \varphi_2)) \models_G \varphi_1 \mathbf{U}_{[p,\infty)} \varphi_2$.
  If $M(p) = 0$, then the equivalence is trivial. Suppose instead that $M(p) > 0$ and $\sigma, t \models \mathbf{G}_{[0,p)} \varphi_1 \wedge \mathbf{G}_{[0,p]} \mathbf{P}_{=0}(\tilde{\mathbf{H}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{U} \varphi_2)$. Let t_0 be the greatest point such that $t \leq t_0$ and $v(t_0) - v(t) = M(p)$. $\sigma, t_0 \models$

$\mathbf{P}_{=0}(\tilde{\mathbf{H}}_{=0}\varphi_1 \wedge \varphi_1 \mathbf{U} \varphi_2)$. Thus there exists t', with $v(t') - v(t) \geq M(p)$, such that $\sigma, t' \models \varphi_1 \mathbf{U} \varphi_2$, and for all $t'' < t'$, if $v(t'') - v(t) \geq M(p)$, then $\sigma, t'' \models \varphi_1$. Moreover, by hypothesis, also for all $t'' \geq t$, if $v(t'') - v(t) < M(p)$, then $\sigma, t'' \models \varphi_1$. Thus, $\sigma, t \models \varphi_1 \mathbf{U}_{[p,\infty)} \varphi_2$.

All cases with past operators are analogous to the future counterpart apart from the following two cases.
Equivalence (4.12)

- $(p = 0 \wedge \varphi_1 \mathbf{S}_{\neq 0} \varphi_2) \vee (p > 0 \wedge \mathit{time} > p \wedge H_{[0,p]}(\varphi_1 \wedge \varphi_1 \mathbf{S}_{\neq 0} \varphi_2)) \models_G \varphi_1 \mathbf{S}_{(p,\infty)} \varphi_2$.
  If $M(p) = 0$, then the equivalence is trivial as before. Suppose instead that $M(p) > 0$ and $\sigma, t \models \mathit{time} > p \wedge \mathbf{H}_{[0,p]}(\varphi_1 \wedge \varphi_1 \mathbf{S}_{(0,\infty)} \varphi_2)$. Since $M(p) > 0$ and $\mathit{time} > M(p)$, there exists $t' < t$ such that $v(t) - v(t') = M(p)$. Thus, $\sigma, t' \models \varphi_1 \mathbf{S}_{(0,\infty)} \varphi_2$. Moreover, for all t'', $t' \leq t'' \leq t$, $\sigma, t'' \models \varphi_1$. Thus, $\sigma, t \models \varphi_1 \mathbf{S}_{(p,\infty)} \varphi_2$.

  Equivalence (4.14)

- $(p = 0 \wedge \varphi_1 \mathbf{S} \varphi_2) \vee (p > 0 \wedge \mathit{time} \geq p \wedge \mathbf{H}_{[0,p)} \varphi_1 \wedge \mathbf{H}_{[0,p]} \mathbf{F}_{=0}(\tilde{\mathbf{G}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{S} \varphi_2)) \models_G \varphi_1 \mathbf{S}_{[p,\infty)} \varphi_2$.
  If $M(p) = 0$, then the equivalence is trivial as before. Suppose instead that $M(p) > 0$ and $\sigma, t \models \mathit{time} \geq p \wedge \mathbf{H}_{[0,p)} \varphi_1 \wedge \mathbf{H}_{[0,p]} \mathbf{F}_{=0}(\tilde{\mathbf{G}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{S} \varphi_2)$. Since $M(p) > 0$ and $\mathit{time} \geq M(p)$, then there exists a smallest point t_0 such that $t > t_0$ and $v(t) - v(t_0) = M(p)$. Thus, $\sigma, t_0 \models \mathbf{F}_{=0}(\tilde{\mathbf{G}}_{=0} \varphi_1 \wedge \varphi_1 \mathbf{S} \varphi_2)$. Thus, there exists t', with $v(t) - v(t') \geq M(p)$, such that $\sigma, t' \models \varphi_1 \mathbf{S} \varphi_2$, and for all $t'' > t'$, if $v(t) - v(t'') \leq M(p)$, then $\sigma, t'' \models \varphi_1$. Moreover, by hypothesis, also for all $t'' \leq t$, if $v(t) - v(t'') < M(p)$, then $\sigma, t'' \models \varphi_1$. Thus, $\sigma, t \models \varphi_1 \mathbf{S}_{[p,\infty)} \varphi_2$.

□

4.3.2 Discretization

We now show that given a XLTL-EF formula with dense, super-dense or super-discrete time model, we can build an equisatisfiable XLTL-EF formula with discrete time model.

The discretization approach is similar to the one described in [61]. The time evolution is split into a sequence of singular or open intervals in such a way that the trace is fine for the input formula on such intervals. In the case of dense time it is always possible to split an interval into two or more nonempty sub-intervals. This is not the case for super-discrete time because a time interval of 1 cannot be split.

Given a trace $\sigma \doteq \langle M, \tau, \overline{\mu} \rangle$ satisfying φ, we derive a sequence of time intervals $I_0, I_1, \ldots$ and use it to build a trace σ_D with discrete time model that satisfies a corresponding formula φ_D. The time interval sequence is obtained from the one of τ by splitting each interval finitely many times so that (i) σ is fine in each interval for all subformulae of φ, (ii) each interval in the sequence is singular or open, (iii) for every interval I_i and subformula $\varphi_1 \mathbf{U} \varphi_2$ of φ, if I_i is open and φ_2 holds in I_i, then φ_2 must also hold on I_{i-1} or I_{i+1} (this last requirement allows for a simpler encoding).

We encode the time interval sequence by relying on two extra Boolean variables *open* and *closed*. *closed* holds iff the current interval is singular. *open* holds iff the current interval is open or we are in super-discrete time and this is a singular interval reached from another singular interval via a time elapse of 1 time unit. Therefore, in the case of dense and super-dense time models, we build a corresponding discrete time model such that the two variables are mutually exclusive ($open \leftrightarrow \neg closed$), while this is not the case in super-discrete time. In this setting, for a time point $t \doteq \langle i, r \rangle$, its immediate successor along time $t' \doteq \langle i, r+1 \rangle$ corresponds to a singular interval in which both *open* and *closed* hold. Such interval represents the

immediate successors along time of t and $\tilde{\mathbf{X}}\varphi$ holds in t iff φ holds in t' (as discussed in Sec. 4.1).

We rewrite a formula φ over V into φ_D over $V \cup \{open, closed\}$ as

$$\varphi_D \doteq \mathcal{D}(\varphi) \wedge \psi_{int} \wedge \psi_{time};$$

where $\mathcal{D}$, ψ_{int} and ψ_{time} are defined as follows. $\mathcal{D}(\varphi)$ is defined recursively on the structure of φ and rewrites its temporal operators distinguishing the case in which the current interval is singular from the one in which it is open. $\mathcal{D}$ is homomorphic with respect to Boolean connectives, functions, constants and variables.

$$\begin{aligned} \mathcal{D}(p(u_0, \ldots, u_n)) &\doteq p(u_0, \ldots, u_n); \\ \mathcal{D}(\neg\varphi) &\doteq \neg\mathcal{D}(\varphi); \\ \mathcal{D}(\varphi_1 \wedge \varphi_2) &\doteq \mathcal{D}(\varphi_1) \wedge \mathcal{D}(\varphi_2). \end{aligned}$$

Consider $\varphi_1 \mathbf{U} \varphi_2$. The formula holds at time point t iff there exists a future state t' satisfying φ_2 and φ_1 always holds in between. However, in the discretized version if $t' > t$ and t' belongs to an open interval, then also φ_1 must hold in that interval. Therefore, φ_1 must hold until either φ_2 holds on a singular interval or $\varphi_2 \wedge \varphi_1$ holds in a open interval. Similar reasoning applies for the corresponding past formula $\varphi_1 \mathbf{S} \varphi_2$.

$$\begin{aligned} \mathcal{D}(\varphi_1 \mathbf{U} \varphi_2) &\doteq \mathcal{D}(\varphi_2) \vee \mathcal{D}(\varphi_1)\mathbf{U}(\mathcal{D}(\varphi_2) \wedge (\mathcal{D}(\varphi_1) \vee closed)); \\ \mathcal{D}(\varphi_1 \mathbf{S} \varphi_2) &\doteq \mathcal{D}(\varphi_2) \vee \mathcal{D}(\varphi_1)\mathbf{S}(\mathcal{D}(\varphi_2) \wedge (\mathcal{D}(\varphi_1) \vee closed)). \end{aligned}$$

Consider now $\mathbf{X}$ and $\tilde{\mathbf{X}}$. $\mathbf{X}$ requires to perform an instantaneous transition, hence a transition from a singular interval to another singular interval without elapse of time. In the super-dense case, there is always no time elapse between two singular intervals. Instead, in the super-discrete case, we need to consider that it is possible to perform a time elapse of 1 and reach a singular interval. Therefore, we need to state that the next state

does not satisfy *open*. In fact, *open* holds in all open intervals and the state corresponding to a singular interval reached via a time elapse of 1 satisfies both *open* and *closed*. Instead, $\tilde{\mathbf{X}}\varphi$ holds if either the current interval is open and φ holds now, or φ holds in the immediate time successors of the current interval. Therefore, the interval must be either an open interval or, in super-discrete time, a singular interval reached via a time elapse of 1 (i.e. a state in which *open* holds). Similar reasoning applies to the $\mathbf{Y}$ and $\tilde{\mathbf{Y}}$ cases.

$$\begin{aligned}
&\mathcal{D}(\mathbf{X}\varphi) \doteq \mathit{closed} \wedge \mathbf{X}(\neg \mathit{open} \wedge \mathcal{D}(\varphi));\\
&\mathcal{D}(\tilde{\mathbf{X}}\varphi) \doteq (\mathit{open} \wedge \mathcal{D}(\varphi)) \vee \mathbf{X}(\mathit{open} \wedge \mathcal{D}(\varphi));\\
&\mathcal{D}(\mathbf{Y}\varphi) \doteq \mathit{closed} \wedge \mathbf{Y}(\neg \mathit{open} \wedge \mathcal{D}(\varphi));\\
&\mathcal{D}(\tilde{\mathbf{Y}}\varphi) \doteq (\mathit{open} \wedge \mathcal{D}(\varphi)) \vee \mathbf{Y}(\mathit{open} \wedge \mathcal{D}(\varphi)).
\end{aligned}$$

Finally, consider the event-freezing functions; we discuss the future case and the symmetric argument holds for the past case. $u@\tilde{\mathbf{F}}(\varphi)$ evaluates to $\mathcal{D}(u)$ at the current time point if $\mathcal{D}(\tilde{\mathbf{X}}\varphi)$ holds now. Otherwise, we translate it into its discrete counterpart and retrieve the value of u at the next time point in which $\mathcal{D}(\varphi)$ holds in a singular interval or in a right open-interval.

$$\begin{aligned}
u@\tilde{\mathbf{F}}(\varphi) \doteq \mathbf{Ite}&((\mathit{open} \wedge \mathcal{D}(\varphi)) \vee \mathbf{X}(\mathit{open} \wedge \mathcal{D}(\varphi)),\\
&\mathcal{D}(u),\\
&\mathcal{D}(u)@\tilde{\mathbf{F}}(\mathcal{D}(\varphi) \vee \mathbf{X}(\mathit{open} \wedge \mathcal{D}(\varphi))));\\
u@\tilde{\mathbf{P}}(\varphi) \doteq \mathbf{Ite}&((\mathit{open} \wedge \mathcal{D}(\varphi)) \vee \mathbf{Y}(\mathit{open} \wedge \mathcal{D}(\varphi)),\\
&\mathcal{D}(u),\\
&\mathcal{D}(u)@\tilde{\mathbf{P}}(\mathcal{D}(\varphi) \vee \mathbf{Y}(\mathit{open} \wedge \mathcal{D}(\varphi)))).
\end{aligned}$$

ψ_{int} encodes the structure of the time model. It ensures that after every open interval there is a singular one and that *time* remains constant in

instantaneous transitions. In the following let δ be $next(time) - time$.

$$\psi_{int} \doteq closed \wedge \neg open \wedge \mathbf{G}((closed \wedge \delta = 0 \wedge \mathbf{X}(closed \wedge \neg open)) \vee (closed \wedge \delta > 0 \wedge \mathbf{X} open \wedge \psi_\tau) \vee (open \wedge \neg closed \wedge \delta > 0 \wedge \mathbf{X}(closed \wedge \neg open)));$$

where ψ_τ is defined depending on the time model τ as follows:

$$\psi_\tau \doteq \begin{cases} \mathbf{X}\neg closed & \text{if } \tau \in \{dense, super - dense\}; \\ \delta = 1 \leftrightarrow \mathbf{X} closed & \text{otherwise.} \end{cases}$$

Therefore, in the dense and super-dense cases there is a strict alternation between states in which $closed \wedge \neg open$ holds and states in which $\neg closed \wedge open$ holds: $open \leftrightarrow \neg closed$ and $open \leftrightarrow \mathbf{X}\neg open$. Instead, in the super-discrete case, from a state in which *closed* holds, by performing a time elapse of 1 time unit, we reach a state that is both *open* and *closed*. Therefore, states in which $\neg open$ holds correspond to singular intervals, while states where $\neg closed$ holds correspond to open intervals.

Finally, ψ_{time} forces the uniformity of predicates over *time* in open intervals:

$$\psi_{time} \doteq \bigwedge_{tu \bowtie cu \in Sub(\varphi)} \mathbf{G}(\neg closed \rightarrow ((\mathcal{D}(tu \leq cu) \rightarrow \mathbf{X}\mathcal{D}(tu \leq cu)) \wedge (\mathcal{D}(tu \geq cu) \rightarrow \mathbf{Y}\mathcal{D}(tu \geq cu))))$$

where $Sub(\varphi)$ denotes the set of subformulae of φ. ψ_{time} splits the time intervals such that for every constant cu occurring in a time constraint, $[cu, cu]$ is a singular interval in the sequence. In addition, the discretization described above generates formulae whose size is exponential in the size of the input. However, since we are interested in equisatisfiability, it is possible to rely on extra variables, one for each subformula, to obtain a linear-size discretized formula.

The following theorem ensures the correctness of the discretization procedure.

Theorem 2 - Discretization generates equisatisfiable formulae *φ and $\varphi_D \doteq \mathcal{D}(\varphi) \wedge \psi_{int} \wedge \psi_{time}$ are equisatisfiable.*

Proof. Given a trace $\sigma \doteq \langle M, \tau, \overline{\mu} \rangle$ satisfying φ we can build a trace σ_D with a discrete time model satisfying φ_D as follows. Let $I_0, I_1, \ldots$ be a sequence of time intervals such that i) σ is fine for all subformulas of φ in each interval I_i, ii) each interval I_i in the sequence is singular or open, and iii) in case of super-dense or super-discrete time, for all $i \geq 0$ and $t \in I_i$, there exists an integer n_i such that $\langle n_i, t \rangle \in \tau$.

We define the discrete time model by setting the value $v(i)$ for all $i \in \mathbb{N}$ based on the time interval sequence as follows:

$$v(i) \doteq \begin{cases} 0 & \text{if } i = 0; \\ v(i-1) + l(I_i) - r(I_{i-1}) & \text{if } i > 0 \text{ and both } I_{i-1}, I_i \text{ are singular;} \\ v(i-1) + \frac{r(I_i) - l(I_i)}{2} & \text{if } i > 0,\ I_{i-1} \text{ is singular and } I_i \text{ is open;} \\ v(i-1) + \frac{r(I_{i-1}) - l(I_{i-1})}{2} & \text{otherwise.} \end{cases}$$

Let $t_i = v(i)$, note that $t_i \in I_i$ for every i. Moreover, notice that if I_i and I_{i+1} are both singular then δ is the distance between the two time values. This δ is always 0 in the dense and super-dense cases, while it is either 0 or 1 in the super-discrete case. If I_i is singular and I_{i+1} is open then δ is equal to half of the length of I_{i+1} and if I_i is not singular then δ is equal to half of the length of I_i.

We build an assignment to *closed* and *open* also based on the time interval sequence. The value of *closed* is determined by the sequence of intervals as follows: $\sigma_D(i)(\mathit{closed}) = \top$ iff I_i is singular. While *open* holds in all open intervals and, in super-discrete time model, in the singular intervals reached via a time elapse of 1: $\sigma_D(i)(\mathit{open}) = \top$ iff I_i is open or

the time model is super-discrete, I_{i-1} is singular and $l(I_i) - r(I_{i-1}) = 1$. Thus, $\sigma_D \models \psi_{int}$.

Let $\mathbf{t}_i = t_i$ in case σ has a dense time and $\mathbf{t}_i = \langle n_i, t_i \rangle$ in case of super-dense or super-discrete time. Let us complete the definition of σ_D by saying that for all $i \geq 0$, $\sigma_D(i)(x) \doteq \sigma(\mathbf{t}_i)(x)$.

We now prove that, for all $i \geq 0$, for all subformulas ψ of φ, $\sigma, \mathbf{t}_i \models \psi$ iff $\sigma_D, i \models \mathcal{D}(\psi)$ and for all terms z in φ, $\sigma(\mathbf{t}_i)(z) = \sigma_D(i)(\mathcal{D}(z))$. The proof works by induction on the structure of ψ and z.

In the base cases, when z is equal to either a constant or a variable, $\sigma(\mathbf{t}_i)(z) = \sigma_D(i)(z)$ by the definition of σ_D.

In the recursive cases in which $\mathcal{D}$ is applied to Boolean connectives and functions, the proof follows immediately from the inductive hypothesis.

We detail the proof in the other cases focusing on the future operators, while the cases of past operators are similar.

- $\psi \doteq \varphi_1 \mathbf{U} \varphi_2$.
  - If $\sigma, \mathbf{t}_i \models \varphi_1 \mathbf{U} \varphi_2$, then either $\sigma, \mathbf{t}_i \models \varphi_2$ or there exists $t' > \mathbf{t}_i$ such that $\sigma, t \models \varphi_2$ and for all t'', $\mathbf{t}_i \leq t'' < t'$, $\sigma, t'' \models \varphi_1$. In the first case, $\sigma_D(i) \models \mathcal{D}(\varphi_2)$ by induction. In the second case, $t' \in I_j$ for some j. Since σ is fine, $\sigma, \mathbf{t}_j \models \varphi_2$ and, if I_j is open then $\sigma, \mathbf{t}_j \models \varphi_1$ too. Thus, by induction, $\sigma_D, j \models \varphi_2 \wedge (\varphi_1 \vee \mathit{closed})$, and thus $\sigma_D, i \models \mathcal{D}(\varphi_1 \mathbf{U} \varphi_2)$.
  - If $\sigma_D, i \models \mathcal{D}(\varphi_1 \mathbf{U} \varphi_2)$, then either $\sigma_D, i \models \mathcal{D}(\varphi_2)$ or there exists $j > i$ such that $\sigma_D, j \models \mathcal{D}(\varphi_2) \wedge (\mathcal{D}(\varphi_1) \vee \mathit{closed})$ and for all k, $i \leq k < j$, $\sigma_D, k \models \mathcal{D}(\varphi_1)$. By induction, either $\sigma, \mathbf{t}_i \models \varphi_2$ or $\sigma, \mathbf{t}_j \models \varphi_2$ and for all k, $i \leq k < j$, $\sigma, \mathbf{t}_k \models \varphi_1$. Moreover, if I_j is open, then $\sigma, \mathbf{t}_j \models \varphi_1$. Since σ is fine, $\sigma, t'' \models \varphi_1$ for all t'', $\mathbf{t}_i \leq t'' < \mathbf{t}_j$. Thus, $\sigma, t \models \varphi_1 \mathbf{U} \varphi_2$.
- $z \doteq u@\tilde{\mathbf{F}}(\phi)$.

– Suppose there exists $t' > \mathbf{t}_i$ such that $\sigma, t' \models \phi$ and for all t'', $\mathbf{t}_i < t'' < t'$, $\sigma, t'' \not\models \phi$. Thus $\sigma(t)(z) = \sigma(t')(u)$. Let $t' \in I_j$ for some $j \geq i$. Since σ is fine for ϕ and for all t'', $\mathbf{t}_i < t'' < t'$, $\sigma, t'' \not\models \phi$, then I_j is singular and $\mathbf{t}_j = t'$, thus $z = \sigma(\mathbf{t}_j)(u)$. Moreover, by induction, $\sigma_D, j \models \mathcal{D}(\phi)$ and $\sigma, k \not\models \mathcal{D}(\phi)$ for $i < k < j$. Thus, $\sigma_D(i)(\mathcal{D}(u)@\tilde{\mathbf{F}}(\mathcal{D}(\phi))) = \sigma_D(j)(\mathcal{D}(u)) = \sigma(\mathbf{t}_j)(u)$. Moreover, for k, $i \leq k \leq j$, $\sigma_D, k \not\models (\mathit{open} \wedge \mathcal{D}(\phi))$. Thus, $\sigma_D(i)(\mathcal{D}(z)) = \sigma_D(i)(\mathcal{D}(u)@\tilde{\mathbf{F}}(\mathcal{D}(\phi))) = \sigma(t)(z)$.

– Similarly, suppose there exists $t' \geq \mathbf{t}_i$ such that $\sigma, t' \models \tilde{\mathbf{X}}\phi$ and for all t'', $\mathbf{t}_i < t'' < t'$, $\sigma, t'' \not\models \phi$. Thus, $z = \sigma(t')(u)$. If $t' = \mathbf{t}_i$ and I_i is open, then, since σ is fine for ϕ, $\sigma, \mathbf{t}_i \models \phi$. By induction $\sigma_D, i \models \mathcal{D}(\phi)$ and thus $\sigma_D(i)(\mathcal{D}(z)) = \sigma(\mathbf{t}_i)(z)$. Similarly, if $t' = \mathbf{t}_i$ and I_i is singular, then I_{i+1} is open $\sigma, \mathbf{t}_{i+1} \models \phi$. By induction, $\sigma_D, i+1 \models \mathcal{D}(\phi)$ and thus $\sigma_D(i)(\mathcal{D}(z)) = \sigma(\mathbf{t}_i)(z)$. If instead $t' > \mathbf{t}_i$, let $t' \in I_j$ for some $j > i$. Since σ is fine for ϕ and for all t'', $\mathbf{t}_i < t'' < t'$, $\sigma, t'' \not\models \phi$, then I_j is singular, $\mathbf{t}_j = t'$, thus I_{j+1} is open, $\sigma, \mathbf{t}_{j+1} \models \phi$ and $\sigma(\mathbf{t}_i)(z) = \sigma(\mathbf{t}_j)(u)$. Moreover, by induction, $\sigma_D, j \models \mathcal{D}(\tilde{\mathbf{X}}\phi)$ and $\sigma, k \not\models \mathcal{D}(\phi)$ for $i < k < j$. Thus, $\sigma_D(i)(\mathcal{D}(u)@\tilde{\mathbf{F}}(\mathcal{D}(\phi) \vee \mathbf{X}(\mathit{open} \wedge \mathcal{D}(\phi)))) = \sigma_D(j)(\mathcal{D}(u)) = \sigma(\mathbf{t}_j)(u)$.

– Considering now the opposite direction, suppose $\sigma_D, i \models \mathit{open} \wedge \mathcal{D}(\phi)$. Then, $\sigma_D(i)(\mathcal{D}(z)) = \sigma_D(i)(\mathcal{D}(u))$, which by induction is equal to $\sigma(\mathbf{t}_i)(u)$. Also by induction, $\sigma, \mathbf{t}_i \models \phi$. Since I_i is open, or reached via a time elapse of 1 in super-discrete time, $\sigma, \mathbf{t}_i \models \tilde{\mathbf{X}}\phi$ and thus $\sigma(\mathbf{t}_i)(z) = \sigma(\mathbf{t}_i)(u)$.

– Similarly, suppose $\sigma_D, i \models \mathbf{X}(\mathit{open} \wedge \mathcal{D}(\phi))$. Then, $\sigma_D(i)(\mathcal{D}(z)) = \sigma_D(i)(\mathcal{D}(u))$, which by induction is equal to $\sigma(\mathbf{t}_i)(u)$. Also by induction, $\sigma, \mathbf{t}_{i+1} \models \phi$. Since I_{i+1} is open, $\sigma, \mathbf{t}_i \models \tilde{\mathbf{X}}\phi$ and thus $\sigma(\mathbf{t}_i)(z) = \sigma(\mathbf{t}_i)(u)$.

– Finally, suppose that we are not in the previous cases and there exists $j > i$ such that $\sigma_D, j \models \mathcal{D}(\phi) \vee \mathbf{X}(\mathit{closed} \wedge \mathcal{D}(\phi))$ and for all k, $i < k < j$, $\sigma_D, k \not\models \mathcal{D}(\phi)$. Then by induction, $\sigma, \mathbf{t}_j \models \tilde{\mathbf{X}}\phi$ and for all k, $i < k < j$, $\sigma, \mathbf{t}_k \not\models \phi$. Thus, since interval I_i is singular or $\mathbf{t}_i \not\models \phi$ (considered in the previous cases), for all t'', $\mathbf{t}_i < t'' < \mathbf{t}_j$, $\sigma, t'' \not\models \phi$. Thus, $\sigma(\mathbf{t}_i)(z) = \sigma(\mathbf{t}_j)(u)$.

It is routine to prove the cases of $\mathbf{X}$ and $\tilde{\mathbf{X}}$ and we can conclude that $\sigma_D \models \mathcal{D}(\varphi)$.

Finally, $\sigma_D \models \psi_{time}$. In fact, $\sigma_D \models \mathit{time} = 0 \wedge \mathbf{G}(\mathit{next}(\mathit{time}) - \mathit{time} = \delta)$ by definition of σ_D; the rest of ψ_{time} is trivially satisfied because σ is fine for φ.

In order to prove the other direction of the theorem, suppose that there exists σ with discrete time such that $\sigma \models \mathcal{D}(\varphi)$. Then, we can build a σ_C with super-dense or super-discrete time such that $\sigma_C \models \varphi$ as follows. Let $t_i = \sum_{h=0}^{i-1} \delta_h$, where δ_h is the value of δ in the h-th step of σ. $I_i \doteq [t_i, t_i]$ if $\sigma, i \models \mathit{closed}$; otherwise $I_i := (t_{i-1}, t_{i+1})$. Let $\sigma(t)(v) = \sigma(i)(v)$ for every $t \in I_i$.

We prove that σ_C is fine for φ. Let M be the first-order structure of σ_C and consider an open interval $I_i = (t_{i-1}, t_{i+1})$. For every $t, t' \in I_i$, $M(t)(x) = M(t')(x)$ and $M(t)(c) = M(t')(c)$, hence, by induction, for every term in form u according to the grammar of XLTL-EF, $M(t)(u) = M(t')(u)$. Thus, for every predicate in the form $p(u_1, \ldots, u_n)$, $\sigma_C, t \models p(u_1, \ldots, u_n)$ iff $\sigma_C, t' \models p(u_1, \ldots, u_n)$. Moreover, for every term cu, the interpretation $M(cu)$ is constant; since $\sigma \models \psi_{time}$, if $\sigma_C, t_i \models \mathit{time} \leq cu$ then $\sigma_C, t_{i+1} \models \mathit{time} \leq cu$ and if $\sigma_C, t_i \models \mathit{time} \geq cu$ then $\sigma_C, t_{i-1} \models \mathit{time} \geq cu$; since $t_{i-1} < t_i < t_{i+1}$, then either $t_i < M(cu)$ and so for all $t \in I_i$ $t_{i-1} < t < t_{i+1} \leq M(cu)$, or $t_i > M(cu)$ and thus so for all $t \in I_i$ $M(cu) \leq t_{i-1} < t < t_{i+1}$. Thus, for every predicate in the form $\mathit{time} \bowtie cu$, for all t, t', $\sigma_C, t \models \mathit{time} \bowtie cu$ iff $\sigma_C, t' \models \mathit{time} \bowtie cu$. The proof for the

remaining predicates in the form tu is similar, taking into account that $time@\tilde{\mathbf{F}}(\varphi)$ and $time@\tilde{\mathbf{P}}(\varphi)$ are either constant within I_i (when φ does not hold in I_i) or they are equivalent to $time$ (when φ holds in I_i).

Finally, it is routine to prove that $\sigma_C \models \varphi$. □

Notice that the discretization generates formulae without the $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ operators. In fact, in discrete time model we can replace every occurrence of $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ with the constant $\bot$. Therefore, we have obtained a XLTL-EF formula interpreted over the discrete time model that does not contain any $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ and the only remaining operators that are not in LTL are the event-freezing functions $@\tilde{\mathbf{F}}$ and $@\tilde{\mathbf{P}}$.

4.3.3 Removing Event-Freezing Functions

We now consider the satisfiability of XLTL-EF properties in the discrete time model. For every such property we construct a corresponding equi-satisfiable one that does not contain the event-freezing functions $@\tilde{\mathbf{F}}$ and $@\tilde{\mathbf{P}}$.

We replace every occurrence of $u@\tilde{\mathbf{F}}(\phi)$ and $u@\tilde{\mathbf{P}}(\phi)$ with a fresh variable $p_{u@\tilde{\mathbf{F}}(\phi)}$ and $p_{u@\tilde{\mathbf{P}}(\phi)}$ respectively. $p_{u@\tilde{\mathbf{F}}(\phi)}$ is a "monitor" for the value of u at the next time ϕ holds. The value of $p_{u@\tilde{\mathbf{F}}(\phi)}$ is chosen nondeterministically when ϕ holds in the next state and remains constant in all other transitions. Furthermore, the assignment to $p_{u@\tilde{\mathbf{F}}(\phi)}$ must be equal to $next(u)$ when ϕ holds in the next state. This ensures that, in every infinite path, the nondeterministic choice must correspond to an oracle that guesses the value u will have at the next occurrence of ϕ. We can apply the symmetric reasoning to constrain the assignment of $p_{u@\tilde{\mathbf{P}}(\phi)}$ and obtain an oracle for $u@\tilde{\mathbf{P}}(\phi)$. More formally, we define the rewriting $\mathcal{R}$ of $u@\tilde{\mathbf{F}}(\phi)$

and $u@\tilde{\mathbf{P}}(\phi)$ in a XLTL-EF formula φ as follows:

$$\begin{aligned}
\mathcal{R}(\varphi, u@\tilde{\mathbf{F}}(\phi)) \doteq\ & \phi[p_{u@\tilde{\mathbf{F}}(\phi)}/u@\tilde{\mathbf{F}}(\phi)] \wedge \\
& \mathbf{G}(\mathbf{X}\phi \rightarrow p_{u@\tilde{\mathbf{F}}(\phi)} = next(u)) \wedge \\
& \mathbf{G}(next(p_{u@\tilde{\mathbf{F}}(\phi)}) = p_{u@\tilde{\mathbf{F}}(\phi)} \vee \mathbf{X}\phi); \\
\mathcal{R}(\varphi, u@\tilde{\mathbf{P}}(\phi)) \doteq\ & \phi[p_{u@\tilde{\mathbf{P}}(\phi)}/u@\tilde{\mathbf{P}}(\phi)] \wedge \\
& \mathbf{G}(\phi \rightarrow next(p_{u@\tilde{\mathbf{P}}(\phi)}) = u) \wedge \\
& \mathbf{G}(next(p_{u@\tilde{\mathbf{F}}(\phi)}) = p_{u@\tilde{\mathbf{F}}(\phi)} \vee \phi).
\end{aligned}$$

The rewriting yields a formula over an extended set of variables. Given a formula ϕ over V, $\mathcal{R}(\varphi, u@\tilde{\mathbf{F}}(\phi))$ is a formula over $V \cup \{p_{u@\tilde{\mathbf{F}}(\phi)}\}$, where $p_{u@\tilde{\mathbf{F}}(\phi)}$ does not occur in ϕ. However, the value of the monitors introduced by the rewriting is uniquely determined by a trace over V. In fact, given a trace σ over the symbols V, we can define a trace $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))$ over $V \cup \{p_{u@\tilde{\mathbf{F}}(\phi)}\}$ such that $\sigma \models \varphi$ iff $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi)) \models \mathcal{R}(\varphi, u@\tilde{\mathbf{F}}(\phi))$. We define the trace corresponding to σ as follows:

$$\begin{aligned}
\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t)(x) &= \sigma(t)(x),\ x \in V; \\
\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t)(p_{u@\tilde{\mathbf{F}}(\phi)}) &= \sigma(t)(u@\tilde{\mathbf{F}}(\phi)); \\
\mathcal{R}(\sigma, u@\tilde{\mathbf{P}}(\phi))(t)(x) &= \sigma(t)(x),\ x \in V; \\
\mathcal{R}(\sigma, u@\tilde{\mathbf{P}}(\phi))(t)(p_{u@\tilde{\mathbf{P}}(\phi)}) &= \sigma(t)(u@\tilde{\mathbf{P}}(\phi)).
\end{aligned}$$

Theorem 3 - $\mathcal{R}$ yields equisatisfiable formulae
φ is equisatisfiable to both $\mathcal{R}(\varphi, u@\tilde{\mathbf{F}}(\phi))$ and $\mathcal{R}(\varphi, u@\tilde{\mathbf{P}}(\phi))$.

Proof. In the following we report the proof for the future case; the proof for the past case is symmetric.

- If $\sigma \models \varphi$ then $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi)) \models \mathcal{R}(\varphi, u@\tilde{\mathbf{F}}(\phi))$.
  Assume $\sigma \models \varphi$. Given the definition of $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))$, the prophecy variable $p_{u@\tilde{\mathbf{F}}(\phi)}$ is given the value of the term $u@\tilde{\mathbf{F}}(\phi)$, and thus $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi)) \models \varphi[p_{u@\tilde{\mathbf{F}}(\phi)}/u@\tilde{\mathbf{F}}(\phi)]$.

For every t, if $\sigma, t \models \mathbf{F}(\phi)$, then there exists $t' > t$ such that $\sigma, t' \models \phi$ and for all t'', $t < t'' < t'$, $\sigma, t'' \not\models \phi$. Thus, $\sigma(t')(u@\tilde{\mathbf{F}}(\phi)) = \sigma(t'')(u@\tilde{\mathbf{F}}(\phi)) = \sigma(t')(u)$. Therefore, $\sigma, t \models (\mathbf{X}\phi \rightarrow p_{u@\tilde{\mathbf{F}}(\phi)} = next(u)) \wedge (next(p_{u@\tilde{\mathbf{F}}(\phi)}) \neq p_{u@\tilde{\mathbf{F}}(\phi)} \rightarrow \mathbf{X}\phi)$.

- If $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi)) \models \mathcal{R}(\varphi, u@\tilde{\mathbf{F}}(\phi))$ then $\sigma \models \varphi$.
  Assume $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi)) \models \mathcal{R}(\varphi, u@\tilde{\mathbf{F}}(\phi))$.

  We need to show that $\sigma \models \varphi$, where σ is obtained from $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))$ by assigning $\sigma(def_{u@\tilde{\mathbf{F}}(\phi)}) \doteq \mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(p_{u@\tilde{\mathbf{F}}(\phi)})$.

  It is sufficient to prove that $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t)(p_{u@\tilde{\mathbf{F}}(\phi)}) = \sigma(t)(u@\tilde{\mathbf{F}}(\phi))$.

  Let us assume that there exists $t' > t$ such that, for all t'', $t < t'' < t'$, $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi)), t'' \not\models \phi$ and $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi)), t' \models \phi$. Thus, $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t)(u@\tilde{\mathbf{F}}(\phi)) = \mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t')(u)$. Since $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi)), t'' \models (\mathbf{X}\phi \rightarrow p_{u@\tilde{\mathbf{F}}(\phi)} = next(u)) \wedge (next(p_{u@\tilde{\mathbf{F}}(\phi)}) \neq p_{u@\tilde{\mathbf{F}}(\phi)} \rightarrow \mathbf{X}\phi)$ for all t'', $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t' - 1)(p_{u@\tilde{\mathbf{F}}(\phi)}) = \mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t')(u)$ and $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t'' - 1)(p_{u@\tilde{\mathbf{F}}(\phi)}) = \mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t')(p_{u@\tilde{\mathbf{F}}(\phi)})$ for all t'', $t < t'' < t'$. Therefore, $\mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t)(p_{u@\tilde{\mathbf{F}}(\phi)}) = \sigma(t')(u)$.

  If such t' does not exist, then $\sigma(t)(u@\tilde{\mathbf{F}}(\phi)) = \sigma(t)(def_{u@\tilde{\mathbf{F}}(\phi)}) = \mathcal{R}(\sigma, u@\tilde{\mathbf{F}}(\phi))(t)(p_{u@\tilde{\mathbf{F}}(\phi)})$.

  □

Notice that since we are considering the discrete time model the operator *next* is defined for LTL, hence $\mathcal{R}$ progressively removes freezing functions relying only on LTL operators.

Finally, the combination of the discretization $\mathcal{D}$ and the recursive application of $\mathcal{R}$ is a procedure that rewrites a $\mathrm{MTLC}_{0,\infty}$ formula on discrete, dense, super-dense or super-discrete time model into an equisatisfiable LTL formula over discrete time model.

4.4 Related work

Many different temporal logics have been defined and extended for timed systems. We will restrict our discussion to the ones related to metric temporal logics, hence to extensions of temporal logics that allow quantitative constraints with respect to the timing of events. We refer to [39] for a broader discussion on the specification languages that have been defined in this context.

The works [8, 78] present two alternatives for the @**F** and @**P** function symbols. In [8] TPTL is defined as the extension of propositional LTL with *freeze quantifiers*. Freeze quantifiers are used to freeze a time variable to the temporal context in which the quantified TPTL formula holds. These quantifiers allow metric temporal constraints by comparing the time at which different formulae held / will hold. A similar objective is achieved in [78]. This work uses registers to store the time at which a formula holds; LTL is extended with *store* and *load* operators to write and read from the registers. Our definition of LTL-EF adopts a more declarative style with functions that directly return the value of variables at the next or last state in which a formula will be/was true, resulting in a more natural specification.

Event-Clock Automata, introduced in [6], restrict the clock variables of TA such that they can only refer to the amount of time passed since the last time an event occurred in the past or required to reach the next time it will occur in the future. Therefore, an Event-Clock Automaton does not allow for arbitrary resets of clock symbols and corresponds to a TA with two clock variables for each event. One clock variable describes the time since the last occurrence of the event (event-recording clock) and the other represents the time until its next occurrence (event-predicting clock). A key aspect of this modelling formalism is that it is closed under complement

and the problem of deciding language inclusion is decidable, while TA are not closed under complement and language inclusion is undecidable. The logical counterpart of Event-Clock Automata, called Event-Clock Temporal Logic, has been studied in [148, 109, 149] and @**F** and @**P** of LTL-EF are a generalisation of event clocks. In fact, they allow the encoding of the fragment $\mathrm{MTL}_{0,\infty}$ [6] of MTL [127] extended with counting [111, 142]. Our semantics for LTL-EF is close to the one defined in [109] for event clocks and for the corresponding quantifiers in the equally-expressive monadic logic. However, [109] relies on nonstandard real numbers to handle left-open time intervals, while we do not require the use of any nonstandard real number.

The discretization of LTL-EF, described in §4.3.2, is similar to the one of [61]. In both cases the executions are represented as sequences of singular or open intervals. However, the considered temporal logics are different. In our case time points are quantified over a discrete, dense, super-dense or super-discrete time model, while [61] considers sequences of intervals. Therefore, the discretization of formulae differs even in the common fragment.

Finally, with respect to tool support, we are not aware of any other tool capable of proving the validity of $\mathrm{MTL}_{0,\infty}$ with parametric bounds in the case of dense or super-dense time. Instead, our implementation in nuXmv supports also first-order constraints. We are aware of only four related tools, namely MightyL [42], Zot [26], ATMOC [125] and CTAV [136]. However, MightyL uses a (discrete) pointwise semantics for MTL, while Zot and ATMOC only support bounded satisfiability of MTL properties. To the best of our knowledge CTAV is the only other tool capable of both proving and falsifying MTL specifications on timed automata. Therefore, all of the above implement techniques specific for timed automata, while our approach is applicable to the broader class of timed transition systems.

Part III

Falsification of Temporal Properties

Chapter 5

Representation of Fair Paths

In the first part of this thesis we have described how the verification of expressive temporal logics on systems with discrete, dense, super-dense and super-discrete time models can be reduced to the LTL model checking problem in the discrete time case. Furthermore, as described in §2.8.2, this problem can be reduced to deciding whether the language of an ITS is empty. Unfortunately, this problem is undecidable. Therefore, the failure of a procedure in identifying a proof for the emptiness of the language does not imply the existence of a fair path and, vice-versa, if we are unable to identify a fair path we cannot conclude that the language is empty. For this reason, we can split the verification problem into two complementary search problems: the search for a proof of emptiness of the language (the property holds) and the search for a fair path (a counterexample to the property).

Most of the techniques proposed in the literature focus on proving that the language is indeed empty, hence that the property holds, while the complementary falsification problem received relatively little attention.

Verification techniques in this context often rely on an abstraction-refinement loop (§2.8.4) to deal with the infinite system. Overapproximations can disregard many details of the model that are not significant

to prove the validity of the property at hand. The main idea is that if we are able to prove that some property holds for a superset of the paths of the system, then the property must hold in the system. However, if some path in the overapproximation does not satisfy the property, then we do not know if such path is an actual counterexample (a path of the original system) or it is a spurious execution. Therefore, we need a procedure to distinguish the two cases and in the second one we need to refine the abstraction.

The undecidability of the problem implies that the abstraction-refinement loop cannot be guaranteed to converge. Therefore, there will always be cases in which it generates an infinite sequence of refinements and spurious paths. This can happen both because the refinements are not "powerful" enough and fail to remove some spurious paths, but also because the spuriousness check fails to identify a path in the concrete system that corresponds to the abstract trace.

In this work we focus on this second case and propose techniques that allow the representation of more complex paths. In the finite-state case, as discussed in §2.7.3, we can always decide whether a fair transition system admits a fair path by looking for a lasso-shaped trace. This also gives a natural way to represent such executions: a finite sequence of states. However, in infinite-state systems there might be no lasso path and we need represent infinite sequences of possibly distinct states. We require a finite representation for an infinite set of states. FOL is a language suitable to represent infinite sets of states (i.e. interpretations to the variables) and SMT-solvers are powerful tools that enable automated reasoning over such formulae.

Recurrent sets (Sec. 2.3) are a FOL representation of infinite chains of states and have been used to represent infinite executions of software programs. However, apart from some trivial cases, they are not sufficient to

conclude that every infinite execution visits some fair state infinitely often. In fact, unless the set underapproximates the fair states, without additional information we cannot conclude that the infinite executions described by the recurrent set are fair.

For this reason, we split the recurrence set into at least two subsets S and D such that D is a subset of the fair states. The union of S and D must describe a closed recurrent set and, in addition, the left-total transition relation must not allow for infinite sequences of S states; every state in S must reach a state in D in a finite number of steps. Notice that this representation does not impose any upper bound on the distance between consecutive fair states and, in addition, it can describe multiple fair paths at the same time.

In this chapter we formally define the structures we use to represent fair paths and show that such representation is sound and also relatively complete. In more detail, we define *funnels* in Sec. 5.1 and in Sec. 5.2 we compose them to obtain a structure we call *funnel-loop* that represents a nonempty set of fair paths. Then, we discuss the soundness and expressiveness of this representation in Sec. 5.3. The chapter concludes in Sec. 5.4 by presenting an example.

5.1 Funnel

A *funnel* is a structure representing finite paths that start in a *source* region, remain in such region for a finite number of steps and finally reach a *destination* region. We define a funnel over a set of symbols V as the 5-tuple $\langle V, S(V), T(V, V'), D(V), \textsc{Rf}(V)\rangle$. S and D are formulae representing respectively the source and destination regions, T is the transition relation and Rf is a ranking function for S with respect to the transition relation T. A funnel can be seen as compact witnesses for universal and

existential reachability [10]. It represents a terminating loop over S where D are the end states of the loop. Depending on the shape of the ranking function, the loop might correspond to a simple loop or to more complex termination arguments such as nested loops.

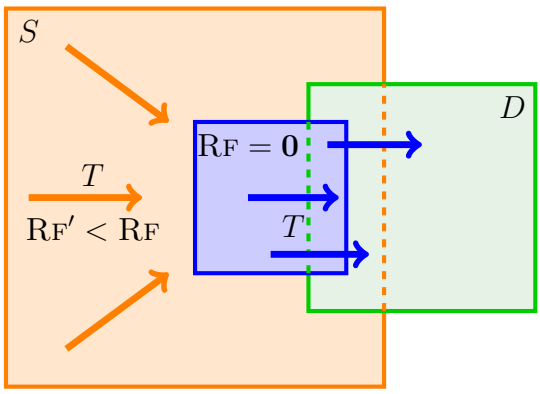

Figure 5.1: Funnel $\langle V, S, T, D, \mathrm{RF} \rangle$.

Fig. 5.1 depicts a funnel and highlights its components using different colours. The source region S is represented by the largest square in orange. The orange arrows inside this shape represent the transitions within the source region that progressively decrease the ranking function RF until it becomes equal to its minimal element $\mathbf{0}$ and the blue square is reached. This square corresponds to the states such that $S \wedge \mathrm{RF} = \mathbf{0}$ holds. Finally, the blue arrows represent the transitions that map every state in the blue region to some state in the destination region, depicted in green.

Every path through the funnel starts from a state in S. Then, it follows the transition relation T and remains in S while the ranking function RF is greater than the minimal element $\mathbf{0}$. Every such step decreases the ranking function, hence the path eventually reaches a state in $S \wedge \mathrm{RF} = \mathbf{0}$. Finally, from such state the path reaches a state in D in a single step. If we consider a trivial ranking function that is always equal to the minimal element $\mathbf{0}$ the 5-tuple simply asserts that every state in S is mapped into D by a single transition T.

Definition 18 - Funnel

A funnel fnl is a the 5-tuple

$$fnl \doteq \langle V, S(V), T(V, V'), D(V), \mathrm{RF}(V) \rangle$$

where: V is a set of symbols, RF *is a ranking function with minimal element* $\mathbf{0}$ *and, S, D and T are formulae representing respectively the source region, destination region and transition relation of fnl. Every funnel must satisfy the following hypotheses:*

F.1 $\forall V \exists V' : S \rightarrow T$;

F.2 $\forall V, V' : (S \wedge \mathbf{0} < \mathrm{RF} \wedge T) \rightarrow S'$;

F.3 $\forall V, V' : (S \wedge \mathbf{0} < \mathrm{RF} \wedge T) \rightarrow \mathrm{RF}' < \mathrm{RF}$;

F.4 $\forall V, V' : (S \wedge \mathrm{RF} = \mathbf{0} \wedge T) \rightarrow D'$.

For a funnel fnl_i we write S_i, T_i, D_i and RF_i to refer to its components.

Hyp. *F.1* requires the relation T to be left-total restricted to the source region S. Notice that we only care about the successors of the states in S, hence we can define a corresponding left-total relation by providing arbitrary successors for the states not in S. For example, given T satisfying all the hypotheses above, the relation $(S \wedge T) \vee (\neg S \wedge \bigwedge_{v \in V} v' = v)$ is left-total in both S and $\neg S$, and also satisfies all the conditions above. Therefore, for every funnel there exists a corresponding one with a left-total transition relation.

Hypotheses *F.2* and *F.3* share the same left-hand-side of the implication and could be easily written as a single formula. We keep them separated because they correspond to two different features of funnels that we will discuss separately also in the proofs. In fact, Hyp. *F.2* ensures that the source region is closed with respect to the transition relation for the states in which the ranking function is not equal to its minimal element. Instead, Hyp. *F.3* guarantees that there cannot be infinite chains of T transitions in

S where the ranking function is greater than its minimal element. Therefore, every path must eventually reach a state in S where the ranking function is equal to $\mathbf{0}$. Together they imply that any path starting from S will reach some state in $S \wedge \mathrm{RF} = \mathbf{0}$ via a finite number of T steps.

Finally, Hyp. *F.4* implies that T must map every state in $S \wedge \mathrm{RF} = \mathbf{0}$ into a state in the destination region D. The destination region may or may not intersect the source region, in particular, if $D \subseteq S$ it can be easily observed that the funnel describes infinite paths.

We define the transition system corresponding to a funnel *fnl* represented by the 5-tuple $\langle V, S, T, D, \mathrm{RF}\rangle$ as $M_{fnl} \doteq \langle V, S, (\neg D \wedge T) \vee (D \wedge D'), \top\rangle$. Every path in $\mathcal{L}(M_{fnl})$ starts from the source region S and follows the transition relation of *fnl* until it reaches the destination region D; from that point on we simply constrain it to remain within the set D. Therefore, if S is not empty then also D must be nonempty and, since all infinite paths are fair in M_{fnl}, $\mathcal{L}(M_{fnl})$ must contain at least one infinite path. For this reason we can write $fnl \models \varphi$ meaning that φ holds in every path in $\mathcal{L}(M_{fnl})$, interpreted using the infinite-trace semantics described for temporal logics (e.g. LTL in Sec. 2.7). From the definition it easily follows that every funnel *fnl* satisfies the following:

$$fnl \models S \ \mathbf{U} \ D.$$

In addition, we refer to the paths through a funnel *fnl* meaning the finite paths of M_{fnl} that end the first time they visit region D. Notice that the paths through a funnel are all finite and each of them is a prefix of some path in $\mathcal{L}(M_{fnl})$.

5.2 Funnel-loop

We define a funnel-loop as a circular chain of funnels $[fnl_i]_{i=0}^{n-1}$ such that the destination region of each funnel is included in the source region of the

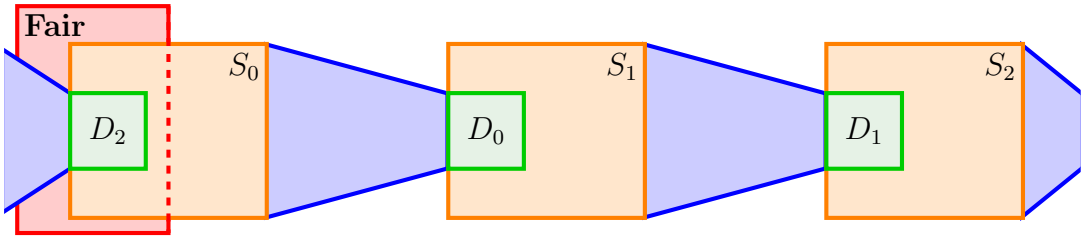

Figure 5.2: Funnel-loop of length 3.

following one and the destination region of the last funnel is included in the source region of the first one.

Fig. 5.2 shows a funnel-loop composed of 3 funnels. Each destination region (in green) is a subset of the source region (in orange) of the following funnel. In addition, the last destination region is also a subset of the rectangle representing the fair states.

Definition 19 - Funnel-loop

A sequence of $n \geq 1$ *funnels* $[fnl_i]_{i=0}^{n-1}$ *over symbols* V *is a funnel-loop iff the following holds.*

FL.1 $\forall i \in \{h\}_{h=0}^{n-1}, V : D_i \rightarrow S_{i+_n 1}$.

Hyp. *FL.1* requires the destination region of every funnel to be included in the source region of the following one, where the funnel following fnl_{n-1} is fnl_0.

We define the set of paths through a funnel-loop *floop*, written $\mathcal{L}(floop)$, as the infinite paths obtained by infinite concatenation of the paths through the funnels in the corresponding chain. We will write $floop \models \varphi$ meaning that φ holds in all such paths. For every funnel, Hyp. *FL.1* ensures that we can extend every path through such funnel, which ends in its destination region, by following the transition relation of the next funnel. Therefore, every path starting in any source region will eventually reach the destina-

tion region of the last funnel:

$$\mathit{floop} \models (\bigvee_{i=0}^{n-1} S_i) \ \mathbf{U} \ D_{n-1}.$$

In addition, Hyp. *FL.1* also implies that every time we reach the destination region of the last funnel in *floop* we are also in the source region of the first one. Therefore, we can extend the execution by appending another finite number of steps, i.e. a finite path starting from S_0 and ending in the last destination region D_{n-1}. We can do this infinitely many times obtaining infinite paths. Notice that, by construction, each path through a funnel must contain at least one transition. Therefore, the infinite concatenation of such finite paths leads to an infinitely long path, since each finite path adds at least one transition.

$$\mathit{floop} \models \mathbf{G}((\bigvee_{i=0}^{n-1} S_i) \ \mathbf{U} \ D_{n-1}).$$

5.2.1 Funnel-loop with Disjoint Regions

We now show that for every funnel-loop there exists a corresponding one that admits the same paths and whose regions are pairwise disjoint. Let $\mathit{floop} \doteq [\mathit{fnl}_i]_{i=0}^{n-1}$ be a funnel-loop of length n over symbols V. We define a corresponding funnel-loop $\widehat{\mathit{floop}} \doteq [\widehat{\mathit{fnl}}_i]_{i=0}^{n-1}$ over symbols $\widehat{V} \doteq V \cup \{l\}$, where l is a fresh integer variable with domain $\{i\}_{i=0}^{n-1}$. We construct $\widehat{\mathit{floop}}$ such that it admits the same set of paths of *floop* projected over the symbols in V and whose regions are pairwise disjoint. We achieve this by relying on l to keep track of the index of the current region. Therefore, every state in the same region of $\widehat{\mathit{floop}}$ prescribes the same assignment to l, while every pair of distinct regions assigns l to different values. This guarantees that the regions of $\widehat{\mathit{floop}}$ are pairwise disjoint.

More formally, we define each $\widehat{\mathit{fnl}_i} \doteq \langle \widehat{V}, \widehat{S}_i, \widehat{T}_i, \widehat{D}_i, \mathrm{RF}_i \rangle$ corresponding to fnl_i such that:

- $\widehat{S}_i \doteq S_i \wedge l = i$;
- $\widehat{D}_i \doteq D_i \wedge l = i +_n 1$;
- $\widehat{T}_i \doteq T_i \wedge (\mathbf{0} < \mathrm{RF}_i \wedge l' = l) \vee (\mathrm{RF}_i = \mathbf{0} \wedge l' = l +_n 1)$.

In the following we first show, in Th. 4, that the resulting structure $\widehat{\mathit{floop}}$ is still a funnel-loop. This requires us to show that each $\widehat{\mathit{fnl}_i}$ is a funnel and they are correctly chained in $\widehat{\mathit{floop}}$. Then, Th. 5 proves that the paths in the language of $\widehat{\mathit{floop}}$, projected over the symbols in V, are all and only the paths in $\mathcal{L}(\mathit{floop})$.

Theorem 4 - $\widehat{\mathit{floop}}$ is a funnel-loop
Let floop be a funnel-loop, then all $[\widehat{\mathit{fnl}_i}]_{i=0}^{n-1}$ *satisfy the hypotheses of Def. 18 and* $\widehat{\mathit{floop}}$ *satisfies the hypotheses of Def. 19.*

Proof. We first show that each $\widehat{\mathit{fnl}_i}$ in $[\widehat{\mathit{fnl}_i}]_{i=0}^{n-1}$ is a funnel and then show that they are correctly concatenated in $\widehat{\mathit{floop}}$.

- Consider first Hyp. *F.1*.
  By definition $\widehat{T}_i \doteq T_i \wedge (\mathbf{0} < \mathrm{RF}_i \wedge l' = l) \vee (\mathrm{RF}_i = \mathbf{0} \wedge l' = l +_n 1)$. In each state either $\mathrm{RF}_i = \mathbf{0}$ holds or $\mathbf{0} < \mathrm{RF}_i$ does. Therefore, in the first case $\widehat{T}_i$ admits a successor such that $l' = l +_n 1$, in the second case it admits a successor in which $l' = l$. Since Hyp. *F.1* holds for fnl_i, its transition relation $T_i(V, V')$ is left-total with respect to S_i. Therefore, $\widehat{T}_i$ is left-total over S_i and also over $\widehat{S}_i$. Therefore, Hyp. *F.1* holds for each $\widehat{\mathit{fnl}_i}$ in $\widehat{\mathit{floop}}$.

- We now consider Hyp. *F.2*.
  By definition $\widehat{T}_i \doteq T_i \wedge (\mathbf{0} < \mathrm{RF}_i \wedge l' = l) \vee (\mathrm{RF}_i = \mathbf{0} \wedge l' = l +_n 1)$. Therefore, every pair of states $\langle \widehat{\boldsymbol{v}}, \widehat{\boldsymbol{v}}' \rangle \in \widehat{T}_i$ such that $\widehat{\boldsymbol{v}} \models \widehat{S}_i \wedge \mathrm{RF}_i > \mathbf{0}$ must be such that $\widehat{\boldsymbol{v}}$ and $\widehat{\boldsymbol{v}}'$ assign the same value i to l. In order to

show that $\widehat{\boldsymbol{v}}' \models \widehat{S}'_i$ we need to show that $\widehat{\boldsymbol{v}}' \models S'_i \wedge l' = i$. However, we have already observed that $\widehat{\boldsymbol{v}}' \models l' = i$, hence we only need to prove that $\widehat{\boldsymbol{v}}' \models S'_i$. Let $\boldsymbol{v} \doteq \widehat{\boldsymbol{v}}_{\downarrow V}$ and $\boldsymbol{v}' \doteq \widehat{\boldsymbol{v}}'_{\downarrow V'}$ be the projections of the two states to the symbols V. Then, $\langle \boldsymbol{v}, \boldsymbol{v}' \rangle \in T_i$ and $\boldsymbol{v} \models S_i$. Since, Hyp. *F.2* holds for fnl_i, then $\boldsymbol{v}' \models S'_i$ also holds. $\widehat{\boldsymbol{v}}'$ agrees with $\boldsymbol{v}' \models S'_i$ on the assignment to all symbols in V, hence $\widehat{\boldsymbol{v}}' \models S'$. Therefore, Hyp. *F.2* holds for $\widehat{fnl}_i$.

- We now show that Hyp. *F.3* holds.
  By applying the same reasoning as above, for every such step in $\widehat{fnl}_i$ we obtain a corresponding step in fnl_i by projecting the assignments over the symbols in V. Hyp. *F.3* holds for fnl_i hence those assignments must decrease the value of the ranking function RF_i. Therefore, since RF_i does not depend on l its value must decrease also in all such steps of $\widehat{fnl}_i$ and Hyp. *F.3* must hold.

- Finally, consider Hyp. *F.4*.
  By applying the same reasoning as the previous two cases, for every step $\langle \widehat{\boldsymbol{v}}, \widehat{\boldsymbol{v}}' \rangle \in \widehat{T}_i$ in $\widehat{fnl}_i$, where $\widehat{\boldsymbol{v}} \models \widehat{S}_i \wedge \mathrm{RF}_i = \mathbf{0}$ we obtain a corresponding step in fnl_i by projecting the assignments over the symbols in V: $\boldsymbol{v} \doteq \widehat{\boldsymbol{v}}_{\downarrow V}$ and $\boldsymbol{v}' \doteq \widehat{\boldsymbol{v}}'_{\downarrow V'}$. $\boldsymbol{v} \models S_i \wedge \mathrm{RF}_i = \mathbf{0}$ since it agrees with $\widehat{\boldsymbol{v}}$ on the assignment of all symbols in V. Hyp. *F.4* holds for fnl_i hence the $\boldsymbol{v}'$ must be in D_i. By construction $\widehat{\boldsymbol{v}}'$ agrees with $\boldsymbol{v}'$ on the assignment of all symbols in V and, by definition of $\widehat{T}$, it must assign l to the index of the next region. Such an assignment agrees with the assignment required by $\widehat{D}_i$, hence Hyp. *F.4* holds for $\widehat{fnl}_i$.

We now show that $\widehat{floop}$ is a funnel-loop, hence that that Hyp. *FL.1* holds. We need to prove that $\forall \widehat{V} : \widehat{D}_i \rightarrow \widehat{S}_{i+_n 1}$ holds for every $0 \leq i < n$. For every i, by substituting the definitions we obtain: $\forall V, l : (D_i \wedge l = i +_n 1) \rightarrow (S_{i+_n 1} \wedge l = i +_n 1)$. There is only one possible assignment to

l making the left-hand-side true and it is exactly the assignment required on the right-hand-side. Therefore, in order for the formula to hold it is sufficient to prove: $\forall V : D_i \rightarrow S_{i+_n 1}$. This is exactly Hyp. *FL.1* for *floop* that holds by hypothesis. Therefore, Hyp. *FL.1* must hold for $\widehat{floop}$. □

Theorem 5 - *floop* and $\widehat{floop}$ admit the same language
The languages of floop and $\widehat{floop}$ *describe the same set of paths projected over the symbols* V*:* $\mathcal{L}(floop) = \mathcal{L}(\widehat{floop})_{\downarrow V}$.

Proof. We show that $\widehat{floop}$ admits all paths of $floop$ and vice-versa by induction on their funnels and the length of the path.

- Assume $floop$ admits a path starting from some state $\boldsymbol{v}$. Then by definition $\boldsymbol{v} \models S_i$ for some i. Let $\widehat{\boldsymbol{v}}$ be the state that assigns l to i and agrees with $\boldsymbol{v}$ on the assignment of all symbols in V. Then $\widehat{\boldsymbol{v}} \models \widehat{S}_i$ and $\widehat{\boldsymbol{v}}$ is an initial state for $\widehat{floop}$.

  Viceversa, assume $\widehat{floop}$ admits a path starting from some state $\widehat{\boldsymbol{v}}$. Then by definition $\widehat{\boldsymbol{v}} \models \widehat{S}_i$ for some i. Let $\boldsymbol{v} \doteq \widehat{\boldsymbol{v}}_{\downarrow V}$ be its projection over the symbols V, then $\boldsymbol{v} \models S_i$ and is an initial state for $floop$.

- By induction, assume σ and $\widehat{\sigma}$ are two corresponding paths ending in state $\boldsymbol{v}$ of region S_i of $floop$ and state $\widehat{\boldsymbol{v}}$ of region $\widehat{S}_i$ of $\widehat{floop}$ respectively.

  Assume $floop$ admits a successor state $\boldsymbol{v}'$ of $\boldsymbol{v}$. Then either $\boldsymbol{v}' \models S'_i$ or $\boldsymbol{v}' \models S'_{i+_n 1}$. Let $\widehat{\boldsymbol{v}}'$ be the assignment that extends $\boldsymbol{v}'$ with $l' = i$ in the first case and $l' = i +_n 1$ otherwise. $\widehat{\boldsymbol{v}}'$ is a successor of $\widehat{\boldsymbol{v}}$ corresponding to $\boldsymbol{v}'$ such that σ extended with $\boldsymbol{v}'$ corresponds to $\widehat{\sigma}$ extended with $\widehat{\boldsymbol{v}}'$.

  Viceversa, assume $\widehat{floop}$ admits a successor state $\widehat{\boldsymbol{v}}'$ of $\widehat{\boldsymbol{v}}$. Let $\boldsymbol{v}' \doteq \widehat{\boldsymbol{v}}'_{\downarrow V'}$ be the projection of $\widehat{\boldsymbol{v}}'$ to the symbols in V. Then, $\boldsymbol{v}'$ is a successor for $\boldsymbol{v}$ such that $\widehat{\sigma}$ extended with $\widehat{\boldsymbol{v}}'$ corresponds to σ extended with $\boldsymbol{v}'$.

□

These results allow us to assume, without loss of generality, the regions of a funnel-loop to be pairwise disjoint and we will exploit this result to simplify our proofs. However, in the definition of funnel-loop we allow for regions with non-empty intersections. This eases the construction of the structure in practical cases. It is possible to consider one funnel at a time and then chain them simply by checking the inclusion of each destination into the corresponding source region.

5.3 Soundness and Relative Completeness

We propose to identify a nonempty set of fair paths for a transition system M as a funnel-loop *floop*, hence every (infinite) path through *floop* must correspond to a fair execution of M. The totality of the transition relation of each funnel (Hyp. *F.1*) and their chaining (Hyp. *FL.1*) ensure that all the paths in $\mathcal{L}(\mathit{floop})$ are infinite. We need such paths to be fair paths, hence they must visit the fairness condition infinitely often. By construction of *floop* we know that every path goes through each S_i and each D_i infinitely many times. Since, by Hyp. *FL.1*, for every source region S_i, there exists a destination region D_j that is contained in it, it is sufficient to require one of the destination regions to contain only fair states. Without loss of generality we assume such a region to be the last one. These conditions ensure that *floop* represents a set of fair paths of M. However, such set might be empty or not reachable in M. Therefore, we finally require the union of the source regions to contain at least one state reachable in M. The existence of such state is sufficient to conclude nonemptiness of $\mathcal{L}(\mathit{floop})$ because the transition relation of each funnel always allows for a successor state (*F.1*) and, by induction, this ensures that every region and the language of *floop* are not empty.

Th. 6 shows that these requirements are sufficient for a funnel-loop

to prove the existence of a fair path in M and Th. 7 shows that if M admits a fair path then there exists a funnel-loop of length one for M. Therefore, funnel-loops composed of a single funnel are expressive enough to represent any fair path. However, funnel-loops of greater length can lead to a description easier to understand for a person and, in addition, could simplify the search procedure. We discuss the possible advantages of considering longer funnel-loops in Sec. 6.2.

Theorem 6 - Funnel-loops are sound
Let $M \doteq \langle V, I^M, T^M, F^M \rangle$ be a fair transition system. Let floop be a funnel-loop of length n over the symbols V and funnels $[fnl_i]_{i=0}^{n-1}$ that satisfy the following hypotheses.

FF.1 *There is at least one state in the union of the source regions of floop that is reachable in M:*

$$M \rightsquigarrow \bigvee_{i=0}^{n-1} S_i.$$

FF.2 *The destination region of the last funnel contains only fair states of M.*

$$\forall V \ : \ D_{n-1} \rightarrow F^M.$$

FF.3 *Every transition of every funnel underapproximates the transition relation of M. For every funnel fnl_i in $[fnl_i]_{i=0}^{n-1}$:*

$$\forall V, V' \ : \ S_i \wedge T_i \rightarrow T^M.$$

Then M admits at least one fair path.

Proof. We first prove that every path in $\mathcal{L}(floop)$ is infinite. Then we prove that every such path is fair with respect to the fairness condition F^M and that every step in every such path satisfies the transition relation T^M. Finally, we prove that $\mathcal{L}(floop)$ allows for at least one path which is a suffix of some path of M.

- Every path in $\mathcal{L}(\mathit{floop})$ is infinite.
  Consider a funnel $\mathit{fnl} \doteq \langle V, S, T, D, \text{RF} \rangle$ in *floop*. Hyp. *F.1* ensures that its transition relation T allows for a successor state for every state in S. Hyp. *F.2* ensures that every path of *fnl* remains in S while $\mathbf{0} < \text{RF}$. Hyp. *F.3* ensures that every such path will eventually reach a state in $S \wedge \text{RF} = \mathbf{0}$. Hyp. *F.4* ensures that every state in such region in one T step reaches a state in D. Therefore, every path starting from the source region S of each funnel can be extended until it reaches its destination region D and it must perform at least 1 transition. If fnl_{i-1} has a successor fnl_i in *floop*, by Hyp. *FL.1* the destination region D_{i-1} is included in S_i: every state in D_{i-1} is also in S_i. Therefore, the concatenation of fnl_{i-1} and fnl_i allows to extend every path starting from either S_{i-1} or S_i until it reaches D_i. By induction this shows that the funnel chain allows the extension of every path starting from the union of the source regions until it reaches the last destination region:
  $$\mathit{floop} \models (\bigvee_{i=0}^{n-1} S_i) \ \mathbf{U} \ D_{n-1}.$$
  In addition, Hyp. *FL.1* requires the last destination region D_{n-1} to be a subset of the first source region S_0. As stated above, we can extend every path starting in every region until it reaches D_{n-1}, hence from S_0 we reach D_{n-1} again in a finite number of steps and at least one. Therefore, since we can extend each path of a finite non-zero number of steps infinitely many times every path in $\mathcal{L}(\mathit{floop})$ is infinite.
- Every path in $\mathcal{L}(\mathit{floop})$ visits F^M infinitely often.
  Hyp. *FF.2* ensures that D_{n-1} underapproximates the fair states F^M. We have already shown above that every path of *floop* reaches a state in D_{n-1} infinitely often. Therefore, such paths visit F^M infinitely often.

- Every step of every path in $\mathcal{L}(\mathit{floop})$ satisfies T^M.
  Every step of every path in $\mathcal{L}(\mathit{floop})$, by definition, corresponds to a transition of some funnel *fnl*. By hypotheses *F.2*, *F.4* and *FL.1* every such path remains within the union of the regions and visits them following the order of the funnels. Therefore, every transition in every path of $floop$ must satisfy $S \wedge T$ for some funnel *fnl* in the sequence. Hyp. *FF.3* ensures that if $S \wedge T$ holds than also T^M is true. Therefore every step of every path of *floop* is also a step of M.

- $\mathcal{L}(\mathit{floop})$ allows for at least one path which is a suffix of some path of M.
  Hyp. *FF.1* ensures that there exists a finite path σ_{pref} of M starting in I^M and ending in some state $\boldsymbol{v}$ such that $\boldsymbol{v} \models \bigvee_{i=0}^{n-1} S_i$. Therefore, $\boldsymbol{v}$ must be in S_i for some $0 \leq i < n$. Then, in *floop* we can extend $\boldsymbol{v}$ to an infinite fair path σ_{suf} starting in $\boldsymbol{v}$. As shown above, every step of σ_{suf} satisfies the transition relation of M and visits the fairness condition F^M infinitely often. The concatenation σ of σ_{pref} and σ_{suf} without repetition of $\boldsymbol{v}$, starts from a state in I^M, every steps satisfies T^M and visits F^M infinitely often. Therefore, σ is a fair path for M: $\sigma \in \mathcal{L}(M)$.

□

Th. 7 ensures that if a transition system admits a fair path then there exists a corresponding funnel-loop. However, it may not be possible to represent it using finite formulae. In finite-state systems it is always possible to represent any set of states and relation between them via a finite formula. In fact, every relation and subset of the states is finite and can be represented as a finite quantifier-free formula, for example as the disjunction of the assignments in the set. However, this might not be the case in infinite-state systems. In such systems there could be an infinite set of

states which cannot be represented by a finite formula. This is possible iff there exists a finite formula equivalent to the disjunction of all states in the set. Therefore, Th. 7 guarantees completeness relative to the expressiveness of the logic used to represent the regions and the transition relation of the funnel. We remark that we are dealing with an undecidable problem, hence there exist no decision procedure that is both sound and complete. In addition, we highlight that the expressiveness of the considered logic is not the only source of incompleteness of an hypothetical search procedure that given a fair transition system tries to identify a corresponding funnel-loop. Any such procedure needs to explore the space of all possible formulae in order to find the ones that denote the desired region, transition and ranking function. This is achievable via a decision procedure if we consider, for example, propositional logic. However, the problem quickly becomes undecidable if we consider more expressive languages, such as the quantifier-free fragment of FOL. Therefore, the two main sources of incompleteness are the expressiveness of the considered logic and the capability of synthesising formulae in such logic. The following theorem concerns the existence of a funnel-loop and shows it to be dependent from the expressiveness of the logic, while it says nothing about the problem of actually identifying it, which we discuss in Chapter 7.

Theorem 7 - Funnel-loops are relatively complete
If a fair transition system M admits at least one fair path, then there exists a funnel-loop floop of length 1 *for M.*

Proof. In the following we define a predicate $\phi(V)$ as the set of assignments $\boldsymbol{v}$ such that $\boldsymbol{v} \models \phi$, meaning that $\phi(V)$ is a formula equivalent to the disjunction of the assignments in the set. Notice that there could be no finite representation of ϕ.

Let $M \doteq \langle V, I^M, T^M, F^M \rangle$ and assume there exists a fair path $\sigma \in \mathcal{L}(M)$. Without loss of generality we assume that σ visits every state at most once.

If this is not the case, it is sufficient to add an additional integer symbol whose assignment increases by one at every transition. In more detail, consider the fair transition system $\langle V \cup c, I^M \wedge c = 0, T^M \wedge c' = c+1, F^M \rangle$. If σ is a fair path of M then, we can obtain a fair path for the modified system by extending the assignment of every state of σ such that $c = 0$ in the first state and in all other states the value of c is its value in the previous state plus 1.

Let *floop* be a funnel-loop of length 1 and $fnl \doteq \langle V, S, T, D, \text{RF} \rangle$ be its funnel. We define the components of *fnl* as follows:

- S contains all and only states of σ: $S \doteq \{\boldsymbol{v} \mid \boldsymbol{v} \in \sigma\}$;
- D contains all and only the fair states of σ: $D \doteq \{\boldsymbol{v} \mid \boldsymbol{v} \in \sigma \wedge F^M(\boldsymbol{v})\}$;
- T is a relation containing all pairs of states $\langle \boldsymbol{v}, \boldsymbol{v}' \rangle$ such that $\boldsymbol{v}'$ is the successor state of $\boldsymbol{v}$ in σ: $T \doteq \{\langle \boldsymbol{v}, \boldsymbol{v}' \rangle \mid \langle \boldsymbol{v}, \boldsymbol{v}' \rangle \in \sigma\}$;
- RF associates to every state in σ the number of steps required to reach the next fair state in σ minus 1: $\forall k > 0, \forall V_1, \ldots, V_k : \text{RF}(V_1) = k - 1 \leftrightarrow (F^M(V_k) \wedge \bigwedge_{i=1}^{k-1} T(V_i, V_{i+1}))$; this is well-defined since each state appears only once in σ and, by construction, T allows for a single successor for each state. In addition, σ is a fair path by hypothesis, hence there can be at most a finite number of non-fair states between every pair of fair states.

We now show that *fnl* satisfies all hypotheses of Def. 18.

- Consider first Hyp. *F.1*.
  σ is an infinite sequence of states, all its states are in S and each pair of subsequent states is in T. Therefore, T must be left-total with respect to S and Hyp. *F.1* holds.
- Consider now Hyp. *F.2*.
  By construction, S contains all states of σ and T is a relation between states of σ. Thus, S is an inductive invariant and Hyp. *F.2* holds.

- We now prove Hyp. *F.3*.
  By construction, $\textsc{Rf}$ is greater than 0 in all states that require more than 1 transition to reach a fair state and T brings all such states 1 step closer to the next fair state in σ. Therefore, $(S(V) \wedge \textsc{Rf}(V) > 0 \wedge T(V, V')) \rightarrow \textsc{Rf}(V) = \textsc{Rf}(V') + 1$ is valid and implies Hyp. *F.3*.

- Finally, we show that Hyp. *F.4* holds.
  By construction, $\textsc{Rf}$ assigns the minimal value 0 to the states that reach a fair state in 1 step. Therefore, Hyp. *F.4* holds.

We now show that *fnl* corresponds to a funnel-loop *floop* of length one; we prove it satisfies the hypothesis of Def. 19.

- We need to show that Hyp. *FL.1* holds.
  floop contains a single funnel and we need to prove that its destination region D is included its source region S. By construction, S contains all states of σ while D contains the subset of states of σ that are also fair. Therefore, $D \rightarrow S$ is valid and Hyp. *FL.1* holds.

Finally, *floop* represents fair paths of M and Th. 6 applies.

- Consider first Hyp. *FF.1*.
  σ is a path of M, hence its first state is an initial state of M. All states of σ are in S. Therefore, S contains at least 1 initial state of M and Hyp. *FF.1* holds.

- Now we consider Hyp. *FF.2*.
  The last destination region of *floop* is D. By construction, D contains only fair states, hence Hyp. *FF.2* holds.

- Finally we prove that Hyp. *FF.3* holds.
  σ is a path of M and every pair of states $\langle \boldsymbol{v}, \boldsymbol{v}' \rangle$ such that $\boldsymbol{v}'$ is the successor state of $\boldsymbol{v}$ in σ satisfies $\boldsymbol{v}, \boldsymbol{v}' \models T^M$. T contains only such pairs, hence $T \rightarrow T^M$ is valid and Hyp. *FF.3* holds.

□

5.4 Funnel-loop Example

We exemplify funnel-loops by showing how they can be used to prove the nontermination of the imperative program reported in Fig. 5.3. In the procedure NONDET returns a nondeterministic value selected from the set provided as input. Assume we are interested in infinite runs in which c is equal to 0 infinitely often, i.e. fair nontermination. We can easily observe that one such run does indeed exist. It is sufficient to replace the nondeterministic assignments of c and n such that c is always set to 0 and n increases its value.

```
1: int c, n, old;
2: while n > old do
3:     while c < n do
4:         c ← c + 1
5:     end while
6:     old ← n
7:     c ← NONDET(ℤ)
8:     n ← NONDET(ℤ)
9: end while
```

Figure 5.3: Nonterminating procedure.

We first encode the software procedure as a transition system. Since we are interested in the fair nontermination of the procedure, we need to define a corresponding transition system whose language is empty iff the procedure does not admit any such path. In order to simplify our discussion, we do not model the program counter explicitly and define the transition relation such that all updates happen simultaneously. We consider the transition system $M \doteq \langle V, I^M, T^M, F^M \rangle$, where: (i) $V \doteq \{c, n, old\}$, $I^M \doteq \top$, (ii) $F^M \doteq c = 0$ and (iv) $T^M \doteq n > old \wedge ((c < n \wedge c' = c + 1 \wedge n' = n \wedge old' = old) \vee (c \geq n \wedge old' = n))$.

Fig. 5.4 reports the definition of M in SMV language. The VAR keyword introduces the state variables of the system. Each TRANS statement is a quantifier-free formula over the state variables that must hold in every transition of the system. In these formulae, the keyword NEXT refers to the next state assignment. Finally, FAIRNESS defines the quantifier-free formula representing the fair states.

```
VAR c: integer; n: integer; old: integer;
TRANS n > old;
TRANS
  (c < n ∧ next(c)=c+1 ∧ next(n)=n ∧ next(old)=old) ∨
  (c ≥ n ∧ next(old)=n);
FAIRNESS c = 0;
```

Figure 5.4: ITS corresponding to Fig. 5.3.

We now define a funnel-loop *floop* of length one proving the fair nontermination of the procedure. Let $floop \doteq [fnl]$ and $fnl \doteq \langle V, S, T, D, \mathrm{RF} \rangle$, where its components are defined as follows. The set of symbols V is the same as for the transition system M. We define the source and destination regions as $S \doteq 0 < old \wedge old < n$ and $D \doteq 0 < old \wedge old < n \wedge c = 0$ respectively. The ranking function ensures the termination of the inner loop of the procedure and is defined as $\mathrm{RF} \doteq n - c$ with minimal element 0. Finally, the transition relation T mimics T^M when $c < n$ and if $c \geq n$ it replaces the nondeterministic assignments by resetting c to 0 and increasing n by 1: $(c < n \wedge c' = c{+}1 \wedge n' = n \wedge old' = old) \vee (c \geq n \wedge old' = n \wedge c' = 0 \wedge n' = n{+}1)$. *floop* is represented in Fig. 5.5, where the outer rectangle corresponds to the source region S, while the two inner rectangles correspond to the region where $\mathrm{RF} = 0$ and the destination region D respectively.

Our goal is now to show that *floop* meets all requirements of Th. 6 for M, hence that it represents a nonempty set of fair paths of M. In order to do that we must first show that *fnl* is indeed a funnel, i.e. it satisfies all

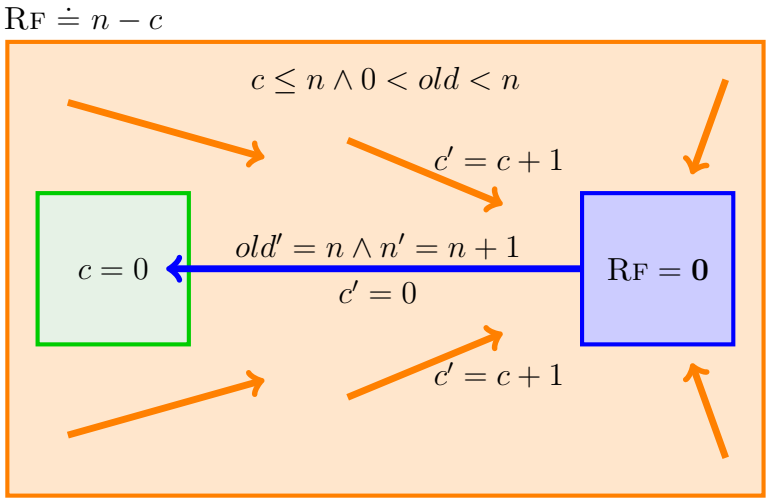

Figure 5.5: Funnel-loop proving the nontermination of Fig. 5.3.

requirements of Def. 18, and similarly *floop* needs to satisfy all conditions of Def. 19 to be a funnel-loop. Finally, we will show that also all hypothesis of Th. 6 are met.

***fnl* is a funnel.** We begin by showing that *fnl* satisfies the requirements of Def. 18. Hyp. *F.1* requires the transition relation of *fnl* to be left-total restricted to S. T is defined as the disjunction of two components and both prescribe a functional assignment to the state variables. The first disjunct applies to all states such that $c < n$, while the second one requires $c \geq n$. Therefore, they cannot contradict each other (they are complementary) and define a left-total relation. Hyp. *F.2* requires the transition relation to map states in $S \wedge \mathrm{RF} > 0$ into states in S. $\mathrm{RF} > 0$ implies $c < n$, hence the transition relation increases c by one, while all other symbols remain constant. Therefore, every such transition remains in S and RF decreases by one, proving that Hyp. *F.3* holds. Finally, Hyp. *F.4* requires T to map states in $S \wedge \mathrm{RF} = 0$ into D. This holds since in such states T assigns c to 0, hence it maps them into states where D holds.

***floop* is a funnel-loop.** Def. 19 of funnel-loop requires Hyp. *FL.1* to hold, hence to prove that the funnels of *floop* are correctly concatenated. *floop* contains a single funnel *fnl* and we need to show that its destination region D is included in the source region S. This trivially follows from their definition, D contains the subset of states of S where $c = 0$.

Th. 6 applies. Finally, we show that *floop* represents a nonempty set of fair paths of M. The assignment $c = 0 \wedge old = 0 \wedge n = 1$ is an initial state of M and is also in S, hence Hyp. *FF.1* holds. Hyp. *FF.2* trivially holds since D implies $c = 0$, hence it contains only fair states. T is an underapproximation of T^M, it removes the nondeterminism of T^M by prescribing a specific next value for c and n. Therefore, $S \wedge T \rightarrow T^M$ and Hyp. *FF.3* holds. Finally, by Th. 6, we conclude that *floop* is a witness for the existence of at least one fair path in the language of M.

Chapter 6

Partitioning the search space of funnel-loops

Chapter 5 and, in particular, Theorems 6 and 7, show that any fair path can be represented as a funnel-loop of length one. However, they do not guarantee the existence of one described by finite formulae and, in addition, such formulae could be arbitrarily complex.

In order to better support the exploration of the space of all possible funnel-loops, this chapter organises the search space along two orthogonal directions. We first *segment* fair paths into a finite sequence of funnels. The infinite path is represented as a concatenation of finite paths. This often allows for funnels described by formulae with a simpler structure and also less complex ranking functions. The other direction is *decomposition*. In this case, a fair transition system is decomposed with respect to a partitioning of its symbols. For each subset of the symbols, we identify components, called *existential components* or E-comps, that represent their behaviour with respect to a sequence of regions. E-comps distinguish three kinds of transitions between their regions and all states in the same region must exhibit transitions of the same kind. In this sense, the regions of an E-comp group states with similar behaviour. We define two operators over these structures. The first operation, called *projection*, shrinks

the set of paths described by an E-comp by considering only a subset of its regions. The second operation, called *composition*, defines how E-comps can be composed to obtain a description of the behaviour of a larger set of symbols, given by the union of the symbols of the composed elements. In this setting we represent fair paths as the composition and projection of a finite set of E-comps and show their correspondence to funnel-loops.

First, in Sec. 6.1, we introduce the running example we will use to exemplify the structures, operations and observations reported in this chapter. Then, we discuss the potential benefits of segmentation in Sec. 6.2. Instead, Sec. 6.3 formally defines E-comps and the two operators, while theorems 11 and 12 highlight the relationship between funnel-loops and E-comps. In more detail, Th. 11 shows that a funnel-loop also defines a corresponding E-comp and, viceversa, Th. 12 details the conditions under which an E-comp corresponds to a funnel-loop that proves the existence of at least one fair path for some fair transition system.

6.1 Running Example

This section introduces a simple LTL verification problem on a software program that will be used as running example throughout this chapter.

Consider the simple program described by Fig. 6.1, where NONDET is a function that nondeterministically selects a value from the set provided as input. Our objective is to check whether in every infinite execution of such program the value of y will eventually remain always positive or always negative. This statement can be written in LTL as $(\mathbf{FG}y \geq 0) \vee (\mathbf{FG}y \leq 0)$. Intuitively, any counterexample to such specification must be a nonterminating execution of the program in which both $y > 0$ and $y < 0$ hold infinitely often.

We encode the software program as an infinite-state transition system

```
1: int x ← NONDET(ℤ)
2: real y ← NONDET(ℝ)
3: while x² ≥ xy do
4:     y ← NONDET(ℝ)
5:     x ← x + 1
6: end while
```

Figure 6.1: Running example.

using an additional variable pc to model the program counter. Then, we employ the reduction from LTL model checking to the existence of a fair path. The resulting infinite-state transition system is $Ex \doteq \langle V, I^{Ex}, T^{Ex}, F^{Ex} \rangle$. Its components are defined as follows. $V \doteq \{x, y, pc, f_0, f_1\}$ is the set of variables, pc and x are two integer variables, y is a real variable, f_0 and f_1 are two Boolean symbols. The initial states are all the states where $pc = 3$ holds: $I^{Ex} \doteq pc = 3$. The fair states are those in which both f_0 and f_1 hold: $F^{Ex} \doteq f_0 \wedge f_1$. f_0 and f_1 have been introduced by the reduction to keep track of whether in the current path we visited a state in $y > 0$ and $y < 0$ respectively. Finally, the transition relation is defined as follows.

$$\begin{aligned} T^{Ex} \doteq\ & (pc = 3 \rightarrow (x^2 \geq xy \wedge pc' = 4 \wedge x' = x \wedge y' = y)) \wedge \\ & (pc = 4 \rightarrow (pc' = 5 \wedge x' = x)) \wedge \\ & (pc = 5 \rightarrow (pc' = 3 \wedge x' = x + 1 \wedge y' = y)) \wedge \\ & ((f_0 \wedge f_1) \rightarrow (\neg f_0' \wedge \neg f_1')) \wedge \\ & (f_0' \rightarrow (f_0 \vee y > 0)) \ \wedge \ (f_1' \rightarrow (f_1 \vee y < 0)). \end{aligned}$$

The first three lines of the formula encode the transition relation of the program. Notice that every state such that $pc = 3$ and $\neg(x^2 \geq xy)$ hold is a deadlock for Ex, i.e. the relation admits no successor state. Finally, the last two lines of the formula ensure that in every path in which $f_0 \wedge f_1$ holds infinitely often also $y > 0$ and $y < 0$ hold infinitely often.

6.2 Segmentation

Th. 7 shows that it is always possible to find a funnel-loop of length one as a witness for some fair path of any transition system. Conversely, one could also represent a fair path as an infinite sequence of funnels such that each source region corresponds to a single state in the sequence. While such sequence does not fit into our definition of funnel-loop, since it involves an infinite number of funnels, it still represents a fair path. Moreover, every ranking function is always equal to $\mathbf{0}$ and all source and destination regions can be represented as quantifier-free formulae written as the conjunction of a constant number of terms. In fact, every state is a total assignment over V and can be represented as the conjunction of $|V|$ equalities. Therefore, the two extremes are given by a funnel-loop of length one whose funnel could require infinite formulae and a structure represented by an infinite sequence of funnels, each of which is defined by formulae with constant size. However, unsurprisingly, it is not the case that the size of the formulae always decreases as the number of funnel increases.

Th. 8 shows that any sequence of two funnels can be represented as a single funnel corresponding to their concatenation. In addition, the recursive application of this transformation constructs, from a funnel-loop of arbitrary length, a corresponding one of length 1. The construction builds an increasingly complex representation for the funnel, hence it outlines a correspondence between considering longer funnel-loops and shorter ones with more complex descriptions. For this reason, funnel-loops of greater length could simplify the search procedure and might not require complex disjunctive representations of the regions, ranking functions and transition relations. In addition, funnel-loops of greater length lead to a description easier to understand for a person, since it clearly defines the ordered sequence of regions that the system must visit.

Theorem 8 - Shrink funnel-loop

Given two funnels fnl_0 and fnl_1 such that $D_0 \to S_1$, there exists a funnel fnl whose paths are obtained as the concatenation of the paths of fnl_0 and fnl_1.

Proof. Let $fnl_0 \doteq \langle V, S_0, T_0, D_0, \mathrm{RF}_0 \rangle$ and $fnl_1 \doteq \langle V, S_1, T_1, D_1, \mathrm{RF}_1 \rangle$ be the two funnels. Theorems 4 and 5 allow us to assume S_0 and S_1 to be disjoint. We define a funnel $fnl \doteq \langle V, S, T, D, \mathrm{RF} \rangle$ where:

$$S \doteq S_0 \vee S_1;$$
$$T \doteq (S_0 \wedge T_0) \vee (S_1 \wedge T_1); \quad \mathrm{RF}(V) \doteq \begin{cases} \langle 1, \mathbf{0}_1, \mathrm{RF}_0(V) \rangle & \text{if } S_0(V); \\ \langle 0, \mathrm{RF}_1(V), \mathbf{0}_0 \rangle & \text{otherwise;} \end{cases}$$
$$D \doteq D_1;$$

with minimal element $\langle 0, \mathbf{0}_1, \mathbf{0}_0 \rangle$ and lexicographic ordering.

We now show that *fnl* is a funnel: it satifies all hypotheses of Def. 18.

- Hyp. *F.1* requires T to be left-total in S. S is defined as the disjunction of S_0 and S_1. $S_0 \wedge T$ is equivalent to T_0 since $S_0 \wedge S_1$ is unsatisfiable. T_0 is left-total relative to S_0 since Hyp. *F.1* holds for fnl_0, hence T is left-total relative to S_0. Similarly $S_1 \wedge T$ is equivalent to T_1 and T_1 is left-total relative to S_1. Therefore, T is left-total relative to $S_0 \vee S_1$, hence also relative to S.

- Hyp. *F.2* requires T to map every state in which the ranking function is greater than the minimal element into states in S. First we show that every state in S_1 with $\mathrm{RF} > \mathbf{0}$ is mapped into S. $S_1 \wedge \mathrm{RF} > \mathbf{0}$, by definition of RF, implies $\mathrm{RF}_1 > \mathbf{0}$. $S_1 \wedge T$ is equivalent to T_1, and Hyp. *F.2* guarantees that T_1 maps every state in $S_1 \wedge \mathrm{RF}_1 > \mathbf{0}$ into S_1, hence in S. Therefore, T maps every such state in S. Consider now a state in S_0. If $\mathrm{RF}_0 > \mathbf{0}$ then by the same reasoning of the previous case, we conclude that T maps such state into S. Otherwise, consider a state in $S_0 \wedge \mathrm{RF}_0 = \mathbf{0}$, hence $\mathrm{RF} = \langle 1, 0, 0 \rangle$. In S_0 the transition relation T is equivalent to T_0. Since Hyp. *F.4* holds for fnl_0, T_0 maps

every such state into D_0. By hypothesis $D_0 \rightarrow S_1$ is valid, hence every such state is in S_1 and also in S. Therefore, T maps every state in $S \wedge \mathrm{RF} > \mathbf{0}$ into a state in S.

- Hyp. *F.3* requires RF to decrease at every transition T. For states in S_1 and $S_0 \wedge \mathrm{RF}_0 > \mathbf{0}$ it directly follows from the fact that Hyp. *F.3* holds for both fnl_0 and fnl_1 and the fact that $S_i \wedge T$ is equivalent to T_i, for $i \in \{0, 1\}$. Consider states in $S_0 \wedge \mathrm{RF}_0 = \mathbf{0}$. By definition, this implies $\mathrm{RF} = \langle 1, \mathbf{0}_1, \mathbf{0}_0 \rangle$. Hyp. *F.4* ensures that every such state is mapped by T_0 into some state in D_0. As stated above, this implies that T maps every such state in some state into S_1. Therefore, in every successor state the ranking function RF evaluates to $\langle 0, \mathrm{RF}_1, \mathbf{0}_0 \rangle$. Since we are considering the lexicographic ordering we have that $\langle 0, \mathrm{RF}_1, \mathbf{0}_0 \rangle < \langle 1, \mathbf{0}_1, \mathbf{0}_0 \rangle$, hence Hyp. *F.3* holds for *fnl*.
- Hyp. *F.4* requires *fnl* to reach its destination region D once the ranking function becomes equal to its minimal element. $\mathrm{RF} = \mathbf{0} = \langle 0, \mathbf{0}_1, \mathbf{0}_0 \rangle$ implies we are in some state in region S_1 and $\mathrm{RF}_1 = \mathbf{0}_1$. Again, $S_1 \wedge T$ is equivalent to T_1 and by Hyp. *F.4* for fnl_1, $S_1 \wedge \mathrm{RF}_1 = \mathbf{0}_1 \wedge T_1 \rightarrow D'_1$ is valid. Therefore, $S \wedge \mathrm{RF} = \mathbf{0} \rightarrow D'$ is also valid.

Then, it is sufficient to observe that any path through *fnl* is the concatenation of a possibly empty path through fnl_0 and one of fnl_1. In addition, every path through fnl_0 [resp. fnl_1] is a prefix [resp. suffix] of some path through *fnl*. □

6.2.1 Example Segmentation

We now define two funnel-loops, of length respectively 6 and 1, for the fair transition system $Ex \doteq \langle V, I^{Ex}, T^{Ex}, F^{Ex} \rangle$ we defined in the running example introduced in Sec. 6.1. Both funnel-loops are sufficient to conclude the existence of a fair path for *Ex*. We first describe the funnel-loop $floop \doteq [fnl_i]_{i=0}^{5}$

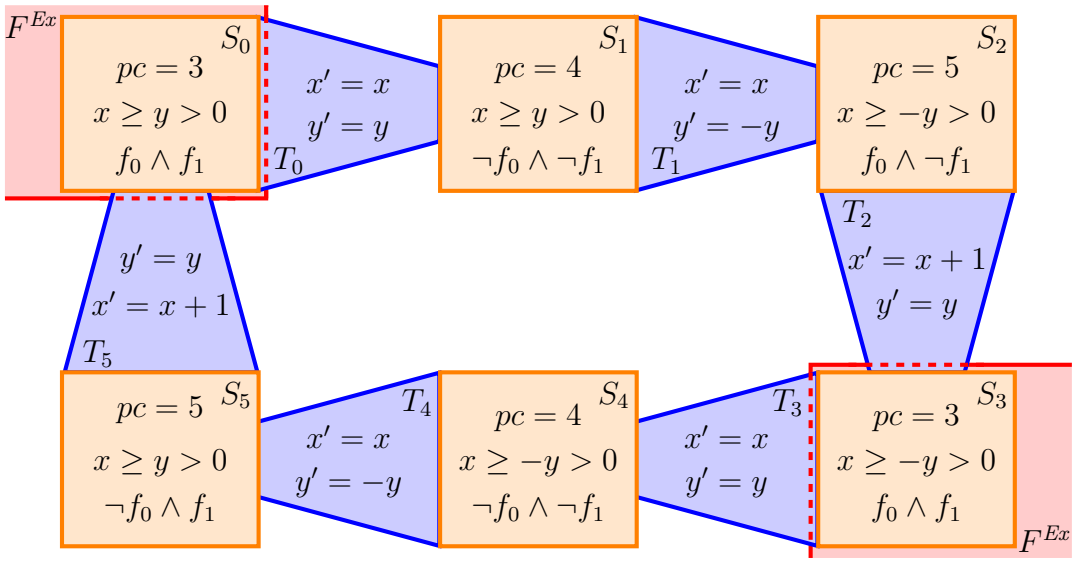

Figure 6.2: funnel-loop *floop* of length 6.

depicted in Fig. 6.2. The figure reports the source regions and transition relations of each funnel. The transitions in the figure report only the constraints for x and y, while the ones for pc, f_0 and f_1 can be trivially inferred by the assignments in the regions. More formally, each funnel fnl_i is the tuple $\langle V, S_i, T_i, D_i, \mathrm{RF}_i \rangle$. We define each ranking function such that it is always equal to its minimal element, $\forall V : \mathrm{RF}_i(V) = \mathbf{0}_i$, and each destination region as the corresponding source region, $D_i \doteq S_{i+_6 1}$. We define the remaining components, source regions and transition relations, as follows. The first funnel fnl_0 represents the step from location 3 to location 4 of Fig. 6.1. In S_0 both f_0 and f_1 are true, hence S_0 contains only fair states and also $D_5 \doteq S_0$ does. Notice that $x \geq y \wedge y > 0$ implies $x^2 \geq xy$. Therefore, the condition of the while loop is satisfied.

$$S_0 \doteq pc = 3 \wedge x \geq y \wedge y > 0 \wedge f_0 \wedge f_1;$$
$$T_0 \doteq pc' = 4 \wedge x' = x \wedge y' = y \wedge \neg f_0' \wedge \neg f_1'.$$

The second funnel fnl_1 performs the step from $pc = 4$ to $pc = 5$. In this step, the program of Fig. 6.1 assigns a nondeterministic value to y.

The funnel underapproximates this transition by always assigning to y the opposite of its current value. In addition, since $y > 0$ in S_1, the transition relation assigns f_0' to true.

$$S_1 \doteq pc = 4 \wedge x \geq y \wedge y > 0 \wedge \neg f_0 \wedge \neg f_1;$$
$$T_1 \doteq pc' = 5 \wedge x' = x \wedge y' = -y \wedge f_0' \wedge \neg f_1'.$$

The third funnel fnl_2 performs the last step of the first iteration of the while loop. Its transition relation increases the value of x by one and, since $y < 0$ holds in the current state, f_1 is true in the next one.

$$S_2 \doteq pc = 5 \wedge x \geq -y \wedge y < 0 \wedge f_0 \wedge \neg f_1;$$
$$T_2 \doteq pc' = 3 \wedge x' = x + 1 \wedge y' = y \wedge f_0' \wedge f_1'.$$

The fourth funnel fnl_3 represents the first step of the loop of Fig. 6.1 as fnl_0. However, in this case y is negative.

$$S_3 \doteq pc = 3 \wedge x \geq -y \wedge y < 0 \wedge f_0 \wedge f_1;$$
$$T_3 \doteq pc' = 4 \wedge x' = x \wedge y' = y \wedge \neg f_0' \wedge \neg f_1'.$$

The fifth funnel fnl_4 is analogous to fnl_1, but has negative value of y.

$$S_4 \doteq pc = 4 \wedge x \geq -y \wedge y < 0 \wedge \neg f_0 \wedge \neg f_1;$$
$$T_4 \doteq pc' = 5 \wedge x' = x \wedge y' = -y \wedge \neg f_0' \wedge f_1'.$$

Finally, funnel fnl_5 is analogous to fnl_2, but has positive value of y.

$$S_5 \doteq pc = 5 \wedge x \geq y \wedge y > 0 \wedge \neg f_0 \wedge f_1;$$
$$T_5 \doteq pc' = 3 \wedge x' = x + 1 \wedge y' = y \wedge f_0' \wedge f_1'.$$

It can be easily observed that each funnel satisfies all hypotheses of Def. 18 and the funnels are correctly chained (Def. 19) by definition of the destination regions. Notice that every region and transition of *floop* is a

purely conjunctive formula and both S_0 and S_3 underapproximate the fair states. Therefore, in every iteration through *floop* we visit the fair states twice, in S_0 with positive y and in S_3 with negative y. *floop* satisfies all hypotheses of Th. 6 and represents at least one counterexample for our LTL model checking problem.

Th. 7 ensures the existence of a funnel-loop of length one. In particular, one such funnel-loop can be obtained via the recursive application of the transformation of Th. 8. In the following we describe the resulting funnel $fnl \doteq \langle V, S, T, D, \mathrm{RF} \rangle$. Its components can be defined in terms of the funnels we defined above as follows. The source region is the union of the source regions of the $\{fnl_i\}_{i=0}^{5}$: $S \doteq \bigvee_{i=0}^{5} S_i$. The destination region is the last destination region of *floop*: $D \doteq D_5$. The transition relation can be defined as $T \doteq \bigvee_{i=0}^{5}(S_i \wedge T_i)$ by observing that the source regions $\{S_i\}_{i=0}^{5}$ are pairwise-disjoint. Finally, the ranking function RF is defined as a function that maps every assignment to the symbols in V to a number in $\mathbb{N}$ such that it assigns decreasing values to states in the regions $S_0, \ldots, S_4$ and assigns the constant 0 to states in S_5:

$$\mathrm{RF}(V) \doteq \begin{cases} 0 & \text{if } S_5(V), \\ 1 & \text{if } S_4(V), \\ 2 & \text{if } S_3(V), \\ 3 & \text{if } S_2(V), \\ 4 & \text{if } S_1(V), \\ 5 & \text{otherwise.} \end{cases}$$

By construction, the transition relation maps every state in S_0 to some state in S_1, which is in turn mapped into S_2 and so on. Therefore, every state in $S \wedge \mathrm{RF} > 0$ is mapped to some other state in S in which the ranking function has lower value. $S \wedge \mathrm{RF} = 0$ is equivalent to S_5 and in such region T corresponds to T_5. Therefore, in a single transition we reach

D_5 that, by definition, is equivalent to D and contained in S_0.

Notice that *floop* is described by purely conjunctive formulae and all its ranking function are always equal to their minimal element. Instead, the description of *fnl* requires longer disjunctive formulae and the ranking function represents the sequence of regions that was explicit in *floop*.

6.3 Decomposition

In the previous section we segmented the paths of a fair transition system into funnels representing finite paths. In the following we adopt an orthogonal view and decompose the system with respect to a partitioning of its symbols. A component, called *existential component* or E-comp, describes the behaviour of all the symbols in a partition with respect to a set of regions and defines a set of loops over such regions. This is done under the assumption that the symbols in the other partitions satisfy some conditions. Therefore, while funnels characterise sets of finite paths, E-comps describe (possibly empty) sets of infinite paths.

We will show how E-comps can be obtained from funnel-loops with an additional restriction on their transition relation, hence how an E-comp can be constructed by concatenating funnels (Th. 11).

Vice-versa we obtain a funnel-loop from a set of E-comps via the following steps. First, we compose E-comps to obtain another E-comp whose loops consider the union of the symbols of the smaller ones. We do this until we obtain a component that considers all the symbols of the system and then, among all its loops we search for one that is also fair. Finally, we restrict its language to only the fair paths by projecting the E-comp over the regions of the fair loop. We show that such E-comp corresponds to a funnel-loop for the transition system, hence proving the existence of at least one fair path (Th. 12).

The section is organised of follows. First, §6.3.1 defines the structure and properties of E-comps. Then, in §6.3.3, we describe the conditions under which a funnel-loop corresponds to an E-comp and defines in which case an E-comp implies the existence of a funnel-loop for a transition system. Finally, §6.3.4 defines the composition and projection operators for E-comps and show that the set of E-comps is closed under such operations.

6.3.1 Existential Components

An *existential component*, or E-comp, is a transition system associated with a set of regions, assumptions and ranking functions. We call the conjunction of a region and its corresponding assumption *restricted region* and, in addition, E-comps associate to each restricted region a ranking function. Restricted regions group states that have "similar behaviour" with respect to the transition relation. If some state in a restricted region allows for a transition with certain characteristics, then a transition with the same characteristics must exist for all states in the restricted region, hence the name existential components. In the following, we first describe what we mean by similar behaviour via the definition of three predicates that classify the transitions. Then, we employ these predicates to formally define E-comps. Finally, we characterise the language of such components.

We are interested in transitions representing self-loops over the restricted regions of two types: self-loops in which the ranking function decreases, called *ranked* transitions, and self-loops in which the ranking function remains constant, called *stutter* transitions. We characterise them using two relations $rankedT_j(V, V')$ and $stutterT_j(V, V')$ over symbols V and V' such that a transition in the restricted region with index j is a ranked transition iff $rankedT_j$ holds and it is a stutter transition iff $stutterT_j$ does. Finally, we consider transitions between possibly distinct restricted regions, starting from a state in which the ranking function is $\mathbf{0}$ and reaching some state

in the second region. We call them *progress* transitions and characterise them using the relation $progressT_{j,j'}(V, V')$. We call a transition a progress transition from region j to region j' iff $progressT_{j,j'}$ holds. Therefore, we distinguish three kinds of transitions between regions and require that either no state allows for a transition of a given kind or all states in the same restricted region admit such a transition. Fig. 6.3 depicts an E-comp

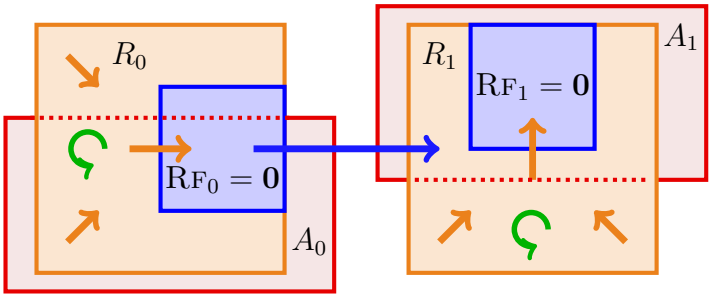

Figure 6.3: E-comp with two regions; transitions of the three types are highlighted with different colors: orange for ranked, green for stutter and blue for progress transitions.

with two regions R_0 and R_1, assumptions A_0 and A_1 and ranking functions RF_0 and RF_1. The orange arrows within the regions represent the ranked transitions, the green circular arrows correspond to the stutter transitions and, finally, the blue arrow from R_0 to R_1 corresponds to the progress transition. Notice that this transition starts from the states in R_0 in which the ranking function RF_0 is equal to its minimal element $\mathbf{0}$.

We now introduce the formal definitions of the predicates classifying the transitions and of E-comps. For a set of symbols V, let $\mathcal{R}^i \doteq \{R^i_j(V)\}_{j=0}^{m-1}$, $\mathcal{A}^i \doteq \{A^i_j(V)\}_{j=0}^{m-1}$ and $\mathcal{W}^i \doteq \{\mathrm{RF}^i_j(V)\}_{j=0}^{m-1}$ be, respectively, the set of regions, assumptions and ranking functions of an E-comp H^i. Then, $R^i_j \wedge A^i_j$ is the j^{th} restricted region and RF^i_j is the ranking function associated to it. We define the three relations that classify the transitions as follows.

$$
\begin{aligned}
rankedT^i_j(V, V') &\doteq R^i_j \wedge A^i_j \wedge \mathbf{0}^i_j <^i_j \mathrm{RF}^i_j \wedge R^{i\prime}_j \wedge A^{i\prime}_j \wedge \mathrm{RF}^{i\prime}_j < \mathrm{RF}^i_j;\\
stutterT^i_j(V, V') &\doteq R^i_j \wedge A^i_j \wedge R^{i\prime}_j \wedge A^{i\prime}_j \wedge \mathrm{RF}^{i\prime}_j = \mathrm{RF}^i_j;\\
progressT^i_{j,j'}(V, V') &\doteq R^i_j \wedge A^i_j \wedge \mathbf{0}^i_j = \mathrm{RF}^i_j \wedge R^{i\prime}_{j'} \wedge A^{i\prime}_{j'}.
\end{aligned}
$$

Notice that, for every j, the relations $\mathit{rankedT}^i_j$ and $\mathit{sutterT}^i_j$ are always disjoint. In the first case the ranking function strictly decreases, while in the second one it must remain constant. However, they are not a partitioning of all possible transitions. In fact, transitions in which the ranking function increases or that move to another region are in neither of the two sets of transitions. In addition, $\mathit{progressT}^i_{j,j'}$ and $\mathit{rankedT}^i_j$ are always disjoint by definition, while the first one could have a non-empty intersection with $\mathit{sutterT}^i_j$ if $j = j'$. In particular, all transitions that both start and end in a state satisfying $R^i_j \wedge A^i_j \wedge \mathrm{RF}^i_j = \mathbf{0}^i_j$ are in the intersection of $\mathit{stutterT}^i_j$ and $\mathit{progressT}^i_{j,j}$. Therefore, the existence of one such transition implies that all states in the restricted region must allow for at least one stutter transition. In addition, for the states in which $\mathrm{RF}_j = \mathbf{0}_j$, this transition is also a progress transition, hence they all admit at least one progress transition that remains in the same region.

We remark that E-comps represent the possibility of performing such transitions and group states for which there exists a successor along the same transition types. Given a partitioning $\{V^i\}_{i=0}^n$ of the symbols V, we want to define the restricted regions such that they allow a set of next assignments to the symbols in a single partition V^i, while the assignment to the symbols in $V^{\neq i} \doteq V \setminus V^i$ is abstracted and only the assumptions are retained. For this reason, we introduce a quantifier alternation $(\exists V^{i'} \forall V^{\neq i'})$, and require the existence of a transition of the given type for every assignment to the $V^{\neq i'}$ satisfying the corresponding assumptions. We apply this reasoning to each of the three types of transitions (stutter, ranked and progress) and formally define E-comps as follows.

Definition 20 - *E*-comp

Given a set of symbols V such that $\{V^i\}_{i=0}^n$ is a partitioning of V for some $n \in \mathbb{N}$. An E*-comp H^i of length $m^i \in \mathbb{N}$ and responsible for V^i is a transition system $\langle V, I^i(V), T^i(V, V')\rangle$ associated with:*

- *a set of regions $\mathcal{R}^i \doteq \{R^i_j(V) \mid 0 \leq j < m^i\}$;*
- *a set of assumptions $\mathcal{A}^i \doteq \{A^i_j(V^{\neq i}) \mid 0 \leq j < m^i\}$, where $V^{\neq i} \doteq \bigcup_{0\leq k<n, k\neq i} V^k$ and $A^i_j(V^{\neq i}) \doteq \bigwedge_{0\leq k<n, k\neq i} A^{i,k}_j(V^k)$;*
- *a set of functions $\mathcal{W}^i \doteq \{\mathrm{RF}^i_j(V) \mid 0 \leq j < m^i\}$ such that each RF^i_j is a ranking function with respect to a well-founded relation $<^i_j$ and minimal element $\mathbf{0}^i_j$;*

such that the following hold:

EC.1 $H^i \models \bigvee_{j=0}^{m^i-1} R^i_j \wedge A^i_j$;

EC.2 $\forall j : 0 \leq j < m^i \rightarrow$

$$\exists V, V' : \mathit{ranked} T^i_j(V, V') \quad \models \quad \forall V \exists V^{i\prime} \forall V^{\neq i\prime} :$$
$$R^i_j \wedge A^i_j \wedge \mathbf{0}^i_j <^i_j \mathrm{RF}^i_j \wedge A^{i\prime}_j \rightarrow R^{i\prime}_j \wedge T^i \wedge \mathrm{RF}^{i\prime}_j <^i_j \mathrm{RF}^i_j;$$

EC.3 $\forall j : 0 \leq j < m^i \rightarrow$

$$\exists V, V' : \mathit{stutter} T^i_j(V, V') \quad \models \quad \forall V \exists V^{i\prime} \forall V^{\neq i\prime} :$$
$$R^i_j \wedge A^i_j \wedge A^{i\prime}_j \rightarrow R^{i\prime}_j \wedge T^i \wedge \mathrm{RF}^{i\prime}_j = \mathrm{RF}^i_j;$$

EC.4 $\forall j, j' : 0 \leq j < m^i \wedge 0 \leq j' < m^i \rightarrow$

$$\exists V, V' : \mathit{progress} T^i_{j,j'}(V, V') \quad \models \quad \forall V \exists V^{i\prime} \forall V^{\neq i\prime} :$$
$$R^i_j \wedge A^i_j \wedge \mathrm{RF}^i_j = \mathbf{0}^i_j \wedge A^{i}_{j'}{}' \rightarrow R^{i}_{j'}{}' \wedge T.$$

In the definition, each assumption $A^i_j(V^{\neq i})$ of E-comp i at index j is composed of n conjuncts $\{A^{i,k}_j(V^k)\}_{0\leq k<n, k\neq i}$, where each conjunct is a formula over the symbols in a single partition V^k different from V^i. In

addition, when clear from the context we will simply write $\mathbf{0}$ and $<$ for $\mathbf{0}^i_j$ and $<^i_j$ respectively.

Hyp. *EC.1* requires all initial states of H^i to be in the union of its restricted regions. Hypotheses *EC.2*, *EC.3* and *EC.4* require that if there exists, respectively, a ranked, stutter or progress transition from the j^{th} restricted region, then every state in the region allows for a successor via a transition of the same kind, provided the assumptions are met. In the case of Hypotheses *EC.2* and *EC.3* every such transition must remain in the j^{th} restricted region (a self-loop). Finally, Hyp. *EC.4* requires that for every j' either no state in the j^{th} restricted region admits a successor in the j'^{th} restricted region or all of them admit at least one.

We define the language of an E-comp $H \doteq \langle V, I, T\rangle$ over $\mathcal{R}$, $\mathcal{A}$ and $\mathcal{W}$, written $\mathcal{L}(H)$, as the language of the corresponding transition system $M \doteq \langle V, I, T^M, \top\rangle$, where T^M is defined as follows:

$$T^M \doteq T \wedge \Big(\bigvee_{j=0}^{m-1} R'_j \wedge A'_j\Big) \wedge \bigwedge_{j=0}^{m-1} (R_j \wedge A_j \wedge \mathbf{0} < \mathrm{RF}_j) \rightarrow (R'_j \wedge A'_j \wedge \mathrm{RF}'_j \leq \mathrm{RF}_j).$$

Therefore, we consider only paths that remain within the set of restricted regions and move from one region to another only if the corresponding ranking function is equal to the minimal element. In fact, as long as the ranking function of the current region is greater than its minimal element, T^M allows only ranked or stutter transitions.

The following two paragraphs highlight the differences between E-comps and two well-known concepts: equivalence relations and bisimulating abstractions. We do this to better motivate the need for this novel structure and the reason why we did not resort those concepts in its definition.

***E*-comp and equivalence relation.** For some equivalence relation, defined in terms of the three transition predicates (*stutterT*, *rankedT* and *progressT*), one might want to define the restricted regions as equivalence classes. However, equivalence classes are required to be maximal; they contain the closure of the equivalence relation. This is not the case for the restricted regions that represent a weaker notion. Two restricted regions could have a non-empty intersection without being the very same restricted region, but this is not possible in the case of equivalence classes; in fact, they are either disjoint or the same class. However, any subset of our hypothetical equivalence class is a valid restricted region. In addition, we are interested in building an underapproximation for a fair transition system, hence the capability of shrinking the regions as much as needed is welcome and simplifies the definition of E-comps.

***E*-comp and (bi)simulation.** The representation of a transition system as an E-comp can be seen as a simulation relation between two transition systems by introducing an additional assumption on the ranking functions.[1] Assume that every ranking function maps states into some natural number and, when decreasing, decreases of exactly 1. Then, given an E-comp H of size m, we define a transition system that has a location corresponding to $R_j \wedge A_j \wedge \mathrm{RF}_j = k$, for every restricted region j and $k \in \mathbb{N}$. We define its transitions such that: (i) a location admits a self-loop iff the corresponding restricted region in H admits a stutter transition, (ii) from every location corresponding to $R_j \wedge A_j \wedge \mathrm{RF}_j = k$ with $k > 0$ there is a transition to the location that corresponds to $R_j \wedge A_j \wedge \mathrm{RF}_j = k - 1$ iff the restricted region j of H admits ranked transitions and (iii) finally, for every

[1] Given two transition systems $A \doteq \langle V^A, I^A, T^A, \top \rangle$ and $B \doteq \langle V^B, I^B, T^B, \top \rangle$ and a relation $rel(V^A, V^B)$ between states of A and states of B. B *simulates* A with respect to *rel* iff:
$\forall V^A, V^B, V^{A\prime} : rel(V^A, V^B) \wedge T^A(V^A, V^{A\prime}) \rightarrow \exists V^{B\prime} : T^B(V^B, V^{B\prime}) \wedge rel(V^{A\prime}, V^{B\prime})$.
In addition, there is a *bisimulation* between A and B if both B simulates A and A simulates B.

$0 \leq j, j' < m$ there are the transitions from the location corresponding to $R_j \wedge A_j \wedge \mathrm{RF}_j = 0$ to each of the locations $R_{j'} \wedge A_{j'} \wedge \mathrm{RF}_{j'} = k$ for all k iff H admits a progress transition from the restricted region with index j to the one with index j'. It is easy to see that such a transition system simulates the E-comp. However, this is not guaranteed to be a bisimulation. In fact, in the progress transitions there could be a mismatch between the value of the ranking function in H with respect to the one required by the location of the transition system.

E-comp with disjoint regions

We now show that every E-comp admits a corresponding one with the same language, projected over the common symbols, and whose regions are pairwise disjoint. Given an E-comp $H \doteq \langle V, I(V), T(V, V') \rangle$ of length m over regions $\mathcal{R}$, assumptions $\mathcal{A}$, ranking functions $\mathcal{W}$ and responsible for $V_r \subseteq V$, we define a corresponding E-comp $\widehat{H} \doteq \langle \widehat{V}, \widehat{I}(\widehat{V}), \widehat{T}(\widehat{V}, \widehat{V}') \rangle$ over regions $\widehat{\mathcal{R}}$, assumptions $\mathcal{A}$, ranking functions $\mathcal{W}$ and responsible for $\widehat{V}_r$ whose regions and pairwise disjoint. $\widehat{H}$, with respect to H, has an additional symbol l: $\widehat{V} \doteq V \cup \{l\}$. The fresh variable is used to keep track of the index of the current region and each region of H is strengthened by requiring the correct assignment for such symbol, while the sets of assumptions and ranking functions remain the same. More formally we define the components of $\widehat{H}$ as follows:

- $\widehat{V} \doteq V \cup \{l\}$, where $l \notin V$ is a fresh symbol whose domain are the integers from 0 to $m - 1$;
- $\widehat{V}_r \doteq V_r \cup \{l\}$;
- $\widehat{\mathcal{R}} \doteq \{R_j \wedge l = j \mid R_j \in \mathcal{R}\}$;
- $\widehat{I}(\widehat{V}) \doteq I(V) \wedge \bigvee_{j=0}^{m-1} (l = j \wedge R_j(V) \wedge A_j(V))$;
- $\widehat{T}(\widehat{V}, \widehat{V}') \doteq T(V, V') \wedge \bigvee_{j=0}^{m-1} (l' = j \wedge R_j(V'))$.

Notice that, by construction, the regions in $\widehat{\mathcal{R}}$ are pairwise disjoint. We now show that H and $\widehat{H}$ admit the same paths with respect to the assignments over the common symbols V and that $\widehat{H}$ is in fact an E-comp.

In the following we first show, in Th. 9, that the resulting structure $\widehat{H}$ is still an E-comp. Then, Th. 10 proves that the paths in the language of $\widehat{H}$ projected over the symbols V are all and only the paths in $\mathcal{L}(H)$.

Theorem 9 - $\widehat{H}$ is an E-comp
If H satisfies all hypotheses of Def. 20 then so does $\widehat{H}$.

Proof. We need to show that $\widehat{H}$ satisfies all hypotheses of Def. 20.

- We first consider Hyp. *EC.1*.
  The initial states of $\widehat{H}$ are a subset of its restricted regions iff:

$$(I \wedge \bigvee_{j=0}^{m-1} (l = j \wedge R_j \wedge A_j)) \rightarrow \bigvee_{j=0}^{m-1} (l = j \wedge R_j \wedge A_j).$$

  The left-hand-side of the implication contains the right-hand-side as a conjunction, hence the formula is valid.

- Consider now Hypotheses *EC.2*, *EC.3* and *EC.4*.
  If $\widehat{H}$ admits a transition between two restricted regions $\widehat{R}_{j_0} \wedge A_{j_0}$ and $\widehat{R}_{j_1} \wedge A_{j_1}$ of one of the 3 kinds then, by construction of $\widehat{T}$, the projection of the two assignments over the symbols V satisfies T. Thus, H must admit a transition of the same kind between its restricted regions $R_{j_0} \wedge A_{j_0}$ and $R_{j_1} \wedge A_{j_1}$. Let t be the kind of the transition. All three hypotheses hold for H, hence every state in $R_{j_0} \wedge A_{j_0}$ admits at least one successor in R_{j_1} via a t-transition, provided A_{j_1} holds. For every state $\widehat{\boldsymbol{v}}$ in $\widehat{R}_{j_0} \wedge A_{j_0}$, let $\boldsymbol{v} \doteq \widehat{\boldsymbol{v}}_{\downarrow V}$ be its restriction to the symbols in V. $\boldsymbol{v}$ is in $R_{j_0} \wedge A_{j_0}$ and it admits a successor $\boldsymbol{v}'$ via a t-transition. Then, $\widehat{\boldsymbol{v}}'$ defined by extending $\boldsymbol{v}'$ with $l' = j_1$, is a t-successor for $\widehat{\boldsymbol{v}}$ in $\widehat{H}$.

□

Theorem 10 - $\widehat{H}$ and H recognise the same language
The languages of H and $\widehat{H}$ admit the same set of paths projected over the symbols V: $\mathcal{L}(H) = \mathcal{L}(\widehat{H})_{\downarrow V}$.

Proof. We proceed by induction on the length of the path. We first show that there is a one-to-one correspondence between the initial states and then that a one-to-one correspondence exists also between the transitions.

- For every initial state $\boldsymbol{v}$ of H, $\boldsymbol{v} \models I$ holds and, by Hyp. *EC.1*, $\boldsymbol{v} \models R_j \wedge A_j$ for some $0 \leq j < m$. Define $\widehat{\boldsymbol{v}}$ over $\widehat{V}$ as the extension of $\boldsymbol{v}$ with the assignment $l = j$. By construction $\widehat{\boldsymbol{v}} \models \widehat{R}_j \wedge A_j \wedge l = j$, hence $\widehat{\boldsymbol{v}} \models \widehat{I}$ and $\widehat{\boldsymbol{v}}$ is an initial state for some path in $\mathcal{L}(\widehat{H})$.

  Viceversa, given an initial state $\widehat{\boldsymbol{v}}$ of $\widehat{H}$, define $\boldsymbol{v} \doteq \widehat{\boldsymbol{v}}_{\downarrow V}$ as the restriction of $\widehat{\boldsymbol{v}}$ to V. By construction, $\boldsymbol{v} \models I$ and $\widehat{\boldsymbol{v}} \models \widehat{R}_j \wedge A_j$ for some $0 \leq j < m$. Thus, by definition of $\widehat{R}_j$, $\widehat{\boldsymbol{v}} \models R_j \wedge A_j$ holds. R_j and A_j do not contain l and $\boldsymbol{v} \models R_j \wedge A_j$ holds, hence $\boldsymbol{v}$ is an initial state of H.

- Consider a transition of H from $\boldsymbol{v}$ to $\boldsymbol{v}'$: $\boldsymbol{v}, \boldsymbol{v}' \models R_j \wedge A_j \wedge T \wedge R_{j'} \wedge A_{j'}$ for some $0 \leq j < m$ and $0 \leq j' < m$. By inductive hypothesis, there is an assignment $\widehat{\boldsymbol{v}}$ for the symbols $\widehat{V}$ corresponding to $\boldsymbol{v}$. We show that $\widehat{H}$ admits a successor $\widehat{\boldsymbol{v}}'$ for $\widehat{\boldsymbol{v}}$ that corresponds to $\boldsymbol{v}'$. By hypothesis, $\boldsymbol{v}' \models R_{j'} \wedge A_{j'}$. We define $\widehat{\boldsymbol{v}}'$ by extending the assignment $\boldsymbol{v}'$ with $l = j'$. Then, $\widehat{\boldsymbol{v}}'$ corresponds to $\boldsymbol{v}'$ and $\widehat{\boldsymbol{v}}, \widehat{\boldsymbol{v}}' \models \widehat{R}_j \wedge A_j \wedge \widehat{T} \wedge \widehat{R}_{j'} \wedge \widehat{A}_{j'}$.

  Viceversa, consider a transition of $\widehat{H}$ from $\widehat{\boldsymbol{v}}$ to $\widehat{\boldsymbol{v}}'$ and an assignment $\boldsymbol{v} \doteq \widehat{\boldsymbol{v}}_{\downarrow V}$ for the symbols V corresponding to $\widehat{\boldsymbol{v}}$. By hypothesis, $\widehat{\boldsymbol{v}} \models \widehat{R}_j \wedge A_j$ and $\widehat{\boldsymbol{v}}' \models \widehat{R}_{j'} \wedge A_{j'}$ for some j and j'. By definition of $\widehat{R}_{j'}$ the following holds: $\widehat{\boldsymbol{v}}' \models R_{j'}$. $\boldsymbol{v}' \doteq \widehat{\boldsymbol{v}}'_{\downarrow V}$ is an assignment over the symbols V corresponding to $\widehat{\boldsymbol{v}}'$. Since $R_{j'}$ and $A_{j'}$ do not depend on l and $\widehat{\boldsymbol{v}}' \models R_{j'} \wedge A_{j'}$, then $\boldsymbol{v}' \models R_{j'} \wedge A_{j'}$. Hence, $\boldsymbol{v}, \boldsymbol{v}' \models R_j \wedge A_j \wedge T \wedge R_{j'} \wedge A_{j'}$. Therefore, $\boldsymbol{v}'$ is a successor for $\boldsymbol{v}$ in H corresponding to $\widehat{\boldsymbol{v}}'$.

□

Similarly to funnel-loops, the definition of E-comps allows for regions with non-empty intersection. This eases their construction and, we exploit the results above to simplify proofs and formal arguments about the language of E-comps.

6.3.2 Example Decomposition: *E*-comps Definition

We now describe a possible strategy to decompose the fair transition system Ex defined in Sec. 6.1. Ex is defined over the set of variables $\{x, y, pc, f_0, f_1\}$. We consider one variable at a time and define a component representing some of its possible behaviours in the system. It is possible to define many different components for every subset of the symbols, for the sake of brevity and clarity we only describe one for each symbol. In the following E-comps we implicitly define every set of initial states as the disjunction of the regions and every ranking function as always equal to its minimal element, hence the E-comps will admit no ranked transition.

Consider first the program counter pc. From the transition relation of Ex the variable will keep assuming the values $[3, 4, 5]$ in this order. For this reason, we define an E-comp H^{pc} with three regions as depicted in Fig. 6.4. H^{pc} is responsible for pc and its three regions are defined as $R_0^{pc} \doteq pc = 3$, $R_1^{pc} \doteq pc = 4$ and $R_2^{pc} \doteq pc = 5$. Then, its transition relation is the disjunction of the three progress transitions between the regions: $T^{pc} \doteq (pc = 3 \wedge pc' = 4) \vee (pc = 4 \wedge pc' = 5) \vee (pc = 5 \wedge pc' = 3)$. We do not introduce any self-loop on the regions, since none exists in the transition relation of Ex. Finally, this behaviour does not require any assumption. In fact, the transition relation

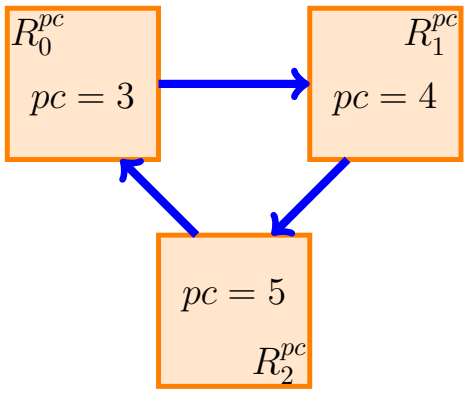

Figure 6.4: E-comp responsible for pc.

T^{pc} is sufficient to ensure that we move from one region to another without having to assume anything about the other symbols.

Consider now the Boolean symbols f_0 and f_1 and define two E-comps: H^{f_0} for f_0 and H^{f_1} for f_1. The E-comps are shown in Fig. 6.5 for $i \in \{0, 1\}$. In both E-comps we need to distinguish the truth value of the two symbols in order to identify the fair states, hence we define each E-comp using two regions. For $i \in \{0, 1\}$, let $R_0^{f_i}$, $R_1^{f_i}$ be the regions of H^{f_i} and T^{f_i} its transition relation. We define the two regions such that one corresponds to the case in which the variable is assigned to true ($R_0^{f_i}$) and the other to the case in which the variable is false ($R_1^{f_i}$). In Ex the two variables can remain constant for any number of steps and toggle their truth value when a certain condition is met. The simplest components we can define in this case are defined as $R_0^{f_i} \doteq f_i$, $R_1^{f_i} \doteq \neg f_i$ and $T^{f_i} \doteq \top$, for $i \in \{0, 1\}$, with no assumptions on the other symbols.

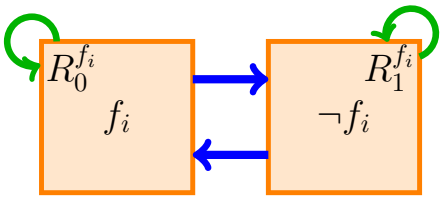

Figure 6.5: E-comp responsible for f_i.

Consider now the variable y and we define H^y as the E-comp responsible for it. In the transition relation of Ex the variable appears in the following predicates $\{y < 0, y > 0, x^2 \geq xy, y' = y\}$. In only one case it appears together with another symbol: $x^2 \geq xy$. We can observe that if $|x| \geq |y|$ then the predicate must hold. This suggests a dependency between x and y and for this reason we could define a single E-comp that considers both symbols together. However, we would like to keep them separated for this example. We break the dependency between the two by considering the stronger conditions $x \geq 1$ and $y \leq 1$. Then, the presence of $y < 0$ and $y > 0$ suggests the need for two regions to distinguish the sign of the variable.

Fig. 6.6 depicts H^y. The E-comp has two regions: $R_0^y \doteq y = -1$ and $R_1^y \doteq y = 1$. The regions differentiate the two cases and we introduce two corresponding assumptions $A_0^y \doteq x \geq 1$ and $A_1^y \doteq x \geq 1$. Finally, we define the transition relation T^y of H^y such that it allows stutter transitions in both regions and also progress transitions to move from one region to the other: $T^y \doteq y' = y \vee y' = -y$.

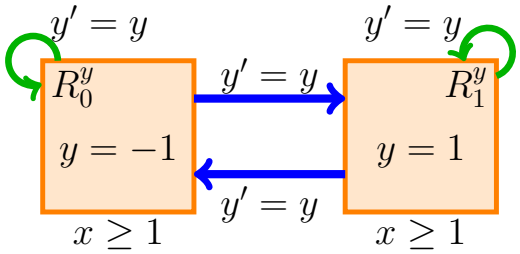

Figure 6.6: E-comp responsible for y.

The only remaining symbol is x, for which we define the E-comp H^x depicted in Fig 6.7. In the transition relation of Ex the variable appears in the following predicates $\{x^2 \geq xy, x' = x, x' = x + 1\}$. We apply the same reasoning as above to analyse the predicate $x^2 \geq xy$ and obtain a single region $R_0^x \doteq x \geq 1$ with assumption $A_0^x \doteq y \leq 1$ for H^x. We define the transition relation T^x of H^x as the disjunction of the two remaining predicates, $T^x \doteq x' = x \vee x' = x + 1$.

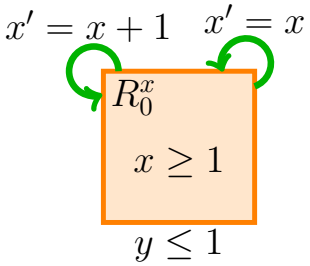

Figure 6.7: E-comp responsible for x.

The purpose of E-comps is to split the process of identifying some fair path into two phases. In the first phase, one symbol or one group of closely related symbols should be considered at a time to identify possible infinite behaviours over them, as exemplified above. The successive step requires to identify how they should be composed in order to obtain a structure that represents fair paths of the transition system. For this reason, we first formally define the conditions required for an E-comp to correspond to a funnel-loop, in §6.3.3. Then, in §6.3.4 we introduce two operators over E-comps. The operators allow the search for an E-comp satisfying the conditions mentioned above. They need to ensure that the components to

be combined are compatible and preserve the existence of the infinite behaviours. We achieve this by combining E-comps such that the respective assumptions are met. §6.3.5 shows how the E-comps we defined above can be composed to prove the existence of a fair path in Ex.

6.3.3 Correspondence between Funnel-loops and *E*-comps

We now characterise the correspondence between funnel-loops and E-comps. The first result, formally stated in Th. 11, shows how a funnel-loop can be used in the decomposition of a system by defining a corresponding E-comp. Then, Th. 12 describes the sufficient conditions for an E-comp to correspond to a funnel-loop for a transition system, hence representing a nonempty set of fair path of such model. These two results provide the formal basis to exploit the segmentation and decomposition of the system for the search of a fair path.

Theorem 11 - Funnel-loop to *E*-comp
Given a set of symbols $\widehat{V} \subseteq V$, a funnel-loop $floop$ composed of the funnels $[fnl_j]_{j=0}^{m-1}$ such that all its transition relations are of the form $T_j(V, \widehat{V}')$ corresponds to an E*-comp $H \doteq \langle V, \bigvee_{j=0}^{m-1} S_j, \bigvee_{j=0}^{m-1} S_j \wedge T_j \rangle$ responsible for symbols $\widehat{V}$ and associated with regions $\{S_j\}_{j=0}^{m-1}$, ranking functions $\{\mathrm{RF}_j\}_{j=0}^{m-1}$ and assumptions $\{\top\}_{j=0}^{m-1}$.*

Proof. We show that H satisfies all hypotheses of Def. 20.

- Consider first Hyp. *EC.1*.
  By definition all assumptions are $\top$ and the initial states are defined as the union of the regions. Therefore, Hyp. *EC.1* holds.
- We now prove that Hyp. *EC.2* holds.
  Hyp. *F.1* ensures that in every region S_j, T_j always allows for a successor state. Therefore, also $\bigvee_{j=0}^{m-1} S_j \wedge T_j$ is left-total in the union of the regions. Hyp. *F.3* ensures that every self-loop on S_j decreases the

associated ranking function RF_j. If a self-loop exists, the transition is a ranked transition and all such transitions are ranked. All such states admit a successor and the successor must decrease the value of the ranking function. Therefore, Hyp. *EC.2* holds.

- Consider now Hyp. *EC.3*.
  As observed in the previous case, all self-loops on a region must decrease the corresponding transition relation. Therefore, H admits no stutter transitions and Hyp. *EC.3* holds.

- Finally we show that Hyp. *EC.4* holds.
  Hyp. *F.4* ensures that from every region S_j if $\mathrm{RF}_j = \mathbf{0}$, then in one transition T_j we always reach a state in D_j and, by Hyp. *FL.1*, such state is in the following region $S_{j+_m 1}$. Since, the transition relation is left-total by Hyp. *F.1*, all states in $S_j \wedge \mathrm{RF}_j = \mathbf{0}$ admit at least one and only successors in $S_{j+_m 1}$, hence Hyp. *EC.4* holds.

□

We now describe the conditions under which an E-comp H represents a nonempty set of fair paths for a transition system M. Th. 12 shows this by defining a corresponding funnel-loop for M that satisfies all hypotheses of Th. 6. H needs to be responsible for all symbols in V and describe a reachable fair lasso over its regions. H must allow for transitions moving from one region to the following one and also transitions remaining in the same region that decrease the corresponding ranking functions. Therefore, H must allow for infinite paths along the loop described by its regions. In order for such paths to be fair, one of the regions must underapproximate the fair states of M and, without loss of generality, we assume such region to be the last one. Finally, every such path must also be a path of M, hence every transition of H within and between its regions must underapproximate the transition relation of M.

Theorem 12 - *E*-comp to Funnel-loop

Let M be a fair transition system $M \doteq \langle V, I^M, T^M, F^M \rangle$. The existence of an E*-comp $H \doteq \langle V, I, T \rangle$ responsible for all symbols V over regions $\mathcal{R}$ and ranking functions $\mathcal{W}$ of length $m \in \mathbb{N}$ satisfying all the following conditions, implies the existence of a funnel-loop for M.*

H.1 $M \rightsquigarrow I$;

H.2 $\forall j \in \{i\}_{i=0}^{m-1}, V, V' : R_j \wedge T \wedge ((\mathrm{RF}'_j < \mathrm{RF} \wedge R'_j) \vee (\mathrm{RF}_j = \mathbf{0} \wedge R'_{j+_m 1})) \rightarrow T^M$;

H.3 $\forall V, V' : R_{m-1} \wedge \mathrm{RF}_{m-1} = \mathbf{0} \wedge T \wedge R'_0 \rightarrow F^{M'}$;

H.4 $\forall j \in \{i\}_{i=0}^{m-1} \exists V, V' : R_j \wedge \mathrm{RF}_j = \mathbf{0} \wedge T \wedge R'_{j+_m 1}$;

H.5 $\forall j \in \{i\}_{i=0}^{m-1} : (\forall V : \mathrm{RF}_j = \mathbf{0}) \vee (\exists V, V' : R_j \wedge T \wedge R'_j \wedge \mathrm{RF}'_j < \mathrm{RF})$.

Proof. Since H is responsible for all symbols V, then all assumptions in $\mathcal{A}$ are empty. We first define the funnel-loop *floop* corresponding to the E-comp H and then prove the following: (i) all of its funnels meet the hypotheses of Def. 18, (ii) *floop* is indeed a funnel-loop (Def. 19) and (iii) *floop* meets all the hypotheses of Th. 6.

Let $\mathit{floop} \doteq [\mathit{fnl}_j]_{j=0}^{m-1}$, where for all j, $\mathit{fnl}_j \doteq \langle V, S_j, T_j, D_j, \mathrm{RF}_j \rangle$ such that: (i) $S_j \doteq R_j$ for $R_j \in \mathcal{R}$; (ii) $T_j \doteq T \wedge ((\mathrm{RF}'_j < \mathrm{RF}_j \wedge R'_j) \vee (\mathrm{RF}_j = \mathbf{0} \wedge R'_{j+_m 1}))$; (iii) $D_j \doteq \exists V' : R_j \wedge \mathrm{RF}_j = \mathbf{0} \wedge T \wedge R'_{j+_m 1}$; and (iv) $\mathrm{RF}_j \in \mathcal{W}$.

We show that each fnl_j is a funnel (Def. 18).

- Consider Hyp. *F.1*.

  T_j is left-total with respect to S_j. In fact, it always allows for at least one successor that is either in the same region with decreasing ranking function or in the following region. H is an E-comp, hence it satisfies Hyp. *EC.2* and Hyp. *EC.4*. Hypotheses *H.4* and *H.5* ensure that at least one transition of both kinds exists in H. Thus, from every state in $S_j \wedge \mathbf{0} < \mathrm{RF}_j$ there exists a successor in the same region

with $\mathrm{RF}'_j < \mathrm{RF}_j$ and from every state in $S_j \wedge \mathrm{RF}_j = \mathbf{0}$, T admits a successor in $S_{j+_m 1}$.

- Hyp. *F.2* holds by construction of T_j.
  $\mathbf{0} < \mathrm{RF}_j$ implies that the second component of the disjunction in T_j is false and T_j becomes equivalent to $T \wedge \mathrm{RF}'_j < \mathrm{RF}_j \wedge R'_j$, hence R'_j.
- Also Hyp. *F.3* holds by construction of T_j.
  $\mathbf{0} < \mathrm{RF}_j$ implies that the second component of the disjunction in T_j is false and T_j is equivalent to $T \wedge \mathrm{RF}'_j < \mathrm{RF}_j \wedge R'_j$, hence $\mathrm{RF}'_j < \mathrm{RF}_j$.
- Consider now Hyp. *F.4*.
  Hyp. *F.4* holds by construction of D_i; in fact, we defined it as the existential image of $R_j \wedge \mathrm{RF}_j = \mathbf{0}$ with respect to $T \wedge R'_{j+_m 1}$.

We now show that *floop* is a funnel-loop (Def. 19), by proving that Hyp. *FL.1* holds. By construction, each T_j, from a state in $R_j \wedge \mathrm{RF}_j = \mathbf{0}$ with $j < m$ can only reach states that are in $R_{j+_m 1}$. Therefore, by construction of D_j Hyp. *FL.1* holds.

Finally, we show that *floop* meets all hypotheses of Th. 6.

- Consider first Hyp. *FF.1*.
  Hyp. *EC.1* ensures that the initial states of H underapproximate the union of its regions. Hyp. *H.1* ensures that there exists a reachable initial state in H. Therefore, there is a reachable state in the union of the regions and Hyp. *FF.1* holds.
- Consider now Hyp. *FF.2*.
  D_{m-1} is the existential image of $R_{m-1} \wedge \mathrm{RF}_{m-1} = \mathbf{0}$ with respect to $T \wedge R'_0$. By Hyp. *H.3*, all such states are fair and Hyp. *FF.2* holds.
- Finally, we show that Hyp. *FF.3* holds.
  By construction, each $S_j \wedge T_j$ underapproximates T and, by Hyp. *H.2*, T underapproximates T^M. Therefore, $S_j \wedge T_j$ underapproximates T^M.

□

6.3.4 Operators over *E*-comps

We now define the *projection* and *composition* operators for E-comps. The first operator shrinks an E-comp by considering only a subset of its regions, while the second operator computes the product of n E-comps. The operators define the search space that needs to be explored to find an E-comp satisfying all hypotheses of Th. 12, i.e. corresponding to a funnel-loop.

E-comp projection

The projection of a E-comp is a smaller E-comp describing a subset of the paths of the original structure. We project an E-comp over an ordered subset of its regions. We restrict the transition relation by removing all stuttering transitions and by requiring the progress transitions to follow the order of the regions. Therefore, projection restricts the language of an E-comp to the paths that visit only regions in the sequence in order and are either finite or reach the last region infinitely often.

Definition 21 - *E*-comp projection
Given an E*-comp* $H \doteq \langle V, I, T\rangle$ *over* m *regions* $\mathcal{R}$*, assumptions* $\mathcal{A}$ *and ranking functions* $\mathcal{W}$*, we define its projection to a sequence of* k *indexes* $idxs \doteq \langle j_0^{\downarrow}, \ldots, j_{k-1}^{\downarrow}\rangle \subseteq \{j\}_{j=0}^{m-1}$ *as the* E*-comp* $H^{\downarrow} \doteq \langle V, I^{\downarrow}, T^{\downarrow}\rangle$ *associated with regions* $\mathcal{R}^{\downarrow}$*, assumptions* $\mathcal{A}^{\downarrow}$ *and ranking functions* $\mathcal{W}^{\downarrow}$ *such that:*

- $I^{\downarrow} \doteq I \wedge \bigvee_{j \in idxs}(R_j \wedge A_j)$;
- $T^{\downarrow} \doteq T \wedge \bigwedge_{h=0}^{k-1} R_{j_h^{\downarrow}} \rightarrow ((R'_{j_h^{\downarrow}} \wedge \mathrm{RF}'_{j_h^{\downarrow}} < \mathrm{RF}_{j_h^{\downarrow}}) \vee (\mathrm{RF}_{j_h^{\downarrow}} = \mathbf{0} \wedge R'_{j_{h+_{k^1}}^{\downarrow}}))$;
- $\mathcal{R}^{\downarrow} \doteq \{R_j \mid j \in idxs \wedge R_j \in \mathcal{R}\}$;
- $\mathcal{A}^{\downarrow} \doteq \{A_j \mid j \in idxs \wedge A_j \in \mathcal{A}\}$;
- $\mathcal{W}^{\downarrow} \doteq \{\mathrm{RF}_j \mid j \in idxs \wedge \mathrm{RF}_j \in \mathcal{W}\}$.

Notice that the projection operator does not modify the formulae representing regions, assumptions and ranking function of an E-comp, but

considers a subset of them. Instead, the operator restricts the set of initial states to only those in one of the restricted regions corresponding to the indexes $idxs$, and the transition relation is strengthened such that it imposes that the regions in $idxs$ are always visited in order.

Theorem 13 - *E*-comps are closed with respect to projection
The projection $H^{\downarrow}$ over indexes $idxs$ of an E*-comp H over regions $\mathcal{R}$, assumptions $\mathcal{A}$ and ranking functions $\mathcal{W}$ is an* E*-comp.*

Proof. We prove that hypotheses *EC.1–EC.4* hold for $H^{\downarrow}$.

- We begin considering Hyp. *EC.1*.
  Hyp. *EC.1* holds by construction since every state $\boldsymbol{v}$ such that $I^{\downarrow}(\boldsymbol{v})$ must also satisfy $\bigvee_{j \in idxs} R_j \wedge A_j$ hence, by definition of $\mathcal{R}^{\downarrow}$ and $\mathcal{A}^{\downarrow}$, $\boldsymbol{v}$ is also in some restricted region of $H^{\downarrow}$.

- Consider now Hyp. *EC.2*.
  For any $j^{\downarrow} \in idxs$, the region $R^{\downarrow}_{j^{\downarrow}}$, assumption $A^{\downarrow}_{j^{\downarrow}}$ and ranking function $\mathrm{RF}^{\downarrow}_{j^{\downarrow}}$ are in both $H^{\downarrow}$ and H. In all transitions such that $R_j \wedge \mathrm{RF}'_j < \mathrm{RF}_j \wedge R'_j$ holds for some $j \in idxs$, $T^{\downarrow}$ is equivalent to T. Therefore, since Hyp. *EC.2* holds for H, it must also hold for $H^{\downarrow}$: if T admits a successor for every state in $R_j \wedge A_j$ such that $\mathrm{RF}'_j < \mathrm{RF}_j \wedge R'_j$ hold, then so does $T^{\downarrow}$.

- Consider now Hyp. *EC.3*.
  By construction of $T^{\downarrow}$ admits no stutter transition. Therefore, the left-hand-side of the entailment is false and Hyp. *EC.3* holds.

- Finally, consider Hyp. *EC.4*.
  For any $j^{\downarrow}, j^{\downarrow'} \in idxs$, if they do not denote consecutive regions in the sequence, $H^{\downarrow}$ does not admit any transition between them and Hyp. *EC.4* holds. Otherwise, $j^{\downarrow}$ and $j^{\downarrow'}$ are the consecutive indexes of the regions $R^{\downarrow}_{j^{\downarrow}}$, $R^{\downarrow}_{j^{\downarrow'}}$, the assumptions $A^{\downarrow}_{j^{\downarrow}}$, $A^{\downarrow}_{j^{\downarrow'}}$ and ranking functions

$\text{RF}^{\downarrow}_{j\downarrow}$, $\text{RF}^{\downarrow}_{j\downarrow'}$. If H does not admit any progress transition between these regions, neither does $H^{\downarrow}$ and Hyp. *EC.4* holds. Otherwise if H admits at least one transition between these regions, the following holds:

$$\exists V, V' : R^{\downarrow}_{j\downarrow} \wedge A^{\downarrow}_{j\downarrow} \wedge \text{RF}^{\downarrow}_{j\downarrow} = \mathbf{0} \wedge T \wedge R^{\downarrow}_{j\downarrow'} \wedge A^{\downarrow}_{j\downarrow'}.$$

Every such V and V' satisfies $T^{\downarrow}$, hence it is also a transition for $H^{\downarrow}$. Therefore, since Hyp. *EC.4* holds for H, every state in $R^{\downarrow}_{j\downarrow} \wedge A^{\downarrow}_{j\downarrow} \wedge \text{RF}^{\downarrow}_{j\downarrow} = \mathbf{0}$ admits a successor in $R^{\downarrow}_{j\downarrow'} \wedge A^{\downarrow}_{j\downarrow'}$. Every such transition is also admitted by $H^{\downarrow}$ and Hyp. *EC.4* holds for $H^{\downarrow}$.

□

E-comp composition

We compose E-comps such that they meet their respective assumptions. Given a set $\{H^i\}_{i=0}^n$ of E-comps, we say that a set of transitions from regions $\{R^i_{j_i}\}_{i=0}^n$ to regions $\{R^i_{j'_i}\}_{i=0}^n$ are *compatible*, if every transition T^i ensures that $\bigwedge_{s=0,s\neq i}^n A^{s,i}_{j'_s}$ holds. In addition, we compose restricted regions of E-comps iff the corresponding ranking functions are independent, hence iff it is possible to decrease one independently from the others. In the following we define two binary predicates $compatible_{\{H^i\}_{i=0}^n}$ and $indepRank_{\{H^i\}_{i=0}^n}$ that hold iff the two conditions are met.

Definition 22 - Compatible transitions

Let $\{H^i\}_{i=0}^n$ *be a set of* E*-comps such that* $\{V^i\}_{i=0}^n$ *are pairwise disjoint and* $\bigcup_{i=0}^n V^i \subseteq V$*. A transition from state* $\boldsymbol{v}$ *to* $\boldsymbol{v'}$ *is* compatible *iff the transitions of the* E*-comps, from every pair of states in the same regions, meet the respective assumptions of the* E*-comps.*

$$compatible_{\{H^i\}_{i=0}^n}(\widehat{V},\widehat{V}') \doteq \forall V, V' : \underbrace{\bigwedge_{0\leq j_0<m^0, 0\leq j_0'<m^0, \ldots, 0\leq j_n<m^n, 0\leq j_n'<m^n}}_{\text{all possible pair of indexes for the E-comps } \{H^i\}_{i=0}^n}$$

$$\Big(\bigwedge_{i=0}^{n} \underbrace{R_{j_i}^i(\widehat{V}) \wedge A_{j_i}^i(\widehat{V}^{\neq i}) \wedge R_{j_i'}^i(\widehat{V}') \wedge A_{j_i'}^i(\widehat{V}^{\neq i\prime})}_{j_i, j_i' \text{ containing both } \widehat{V} \text{ and } \widehat{V}'} \wedge$$

$$\underbrace{R_{j_i}^i(V) \wedge A_{j_i}^i(V^{\neq i}) \wedge R_{j_i'}^i(V') \wedge A_{j_i'}^i(V^{\neq \{h\}_{h=0}^n\prime}) \wedge T^i(V,V')}_{\substack{\text{for all } V \text{ in } j_i,\ V' \text{ in } j_i' \text{ such that } V,V' \models T^i \text{ and } V' \text{ meets all} \\ \text{assumptions of } H^i \text{ at } j_i' \text{ on symbols of E-comps not in } \{H^i\}_{i=0}^n}} \quad \wedge$$

$$\underbrace{(\mathrm{RF}_{j_i'}^i(\widehat{V}') < \mathrm{RF}_{j_i}^i(\widehat{V}) \leftrightarrow \mathrm{RF}_{j_i'}^i(V') < \mathrm{RF}_{j_i}^i(V))}_{\text{transition } V,V' \text{ of the same type of transition} \widehat{V},\widehat{V}'} \wedge$$

$$(\mathbf{0} < \mathrm{RF}_{j_i}^i(\widehat{V}) \leftrightarrow \mathbf{0} < \mathrm{RF}_{j_i}^i(V)) \wedge (\mathbf{0} < \mathrm{RF}_{j_i'}^i(\widehat{V}') \leftrightarrow \mathbf{0} < \mathrm{RF}_{j_i'}^i(V'))) \rightarrow$$

$$\bigwedge_{i=0}^{n} \underbrace{\bigwedge_{h=0, h\neq i}^{n} A_{j_i'}^{i,h}(V^{h\prime})}_{\text{all assumptions of } H^i \text{ on the } \{V^h\}_{h=0}^n \text{ are met}}.$$

A set of transitions has *independent ranks* if it is possible to decrease each ranking function independently from the others. Consider the restricted regions $\{R_{j_i}^i \wedge A_{j_i}^i\}_{i=0}^n$, there exist transitions with *independent ranks* if, for each $\mathrm{RF}_{j_{ir}}^{i_r}$ with $0 \leq i_r \leq n$, it is possible to perform a self-loop on the conjunction of the restricted regions $\bigwedge_{i=0}^n R_{j_i}^i \wedge A_{j_i}^i$ such that $\mathrm{RF}_{j_{ir}}^{i_r}$ decreases and all the other ranking functions remain constant: $\bigwedge_{i=0, i\neq i_r}^n \mathrm{RF}_{j_i}^{i\prime} = \mathrm{RF}_{j_i}^i$.

Definition 23 - Independent ranks

Let $\{H^i\}_{i=0}^n$ *be a set of the* E*-comps such that* $\{V^i\}_{i=0}^n$ *are pairwise disjoint and* $\bigcup_{i=0}^n V^i \subseteq V$*. A self-loop over the intersection of the restricted regions has* independent ranks *iff for every ranking function there exists a compatible conjunction of the transitions decreasing only that function.*

$$
\begin{aligned}
&\mathit{indepRank}_{\{H^i\}_{i=0}^n}(\widehat{V},\widehat{V}') \doteq \underbrace{\bigwedge_{0\le j_0<m^0,\ldots,0\le j_n<m^n}}_{\text{all possible indexes for the E-comps } \{H^i\}_{i=0}^n} \\
&((\underbrace{(\sum_{i=0}^{n} \mathrm{RF}_{j_i}^i)(\widehat{V}') < (\sum_{i=0}^{n} \mathrm{RF}_{j_i}^i)(\widehat{V})}_{\text{some ranking function decreases, all others remain constant}} \quad \wedge \\
&\underbrace{\bigwedge_{i=0}^{n} R_{j_i}^i(\widehat{V}) \wedge A_{j_i}^i(\widehat{V}^{\neq i}) \wedge R_{j_i}^i(\widehat{V}') \wedge A_{j_i}^i(\widehat{V}^{\neq i\prime})}_{\widehat{V},\widehat{V}' \text{ are in restricted regions } j_i,j_i'}) \rightarrow \\
&\bigwedge_{i=0}^{n} \underbrace{(\forall V : (\bigwedge_{h=0}^{n} R_{j_h}^h(V) \wedge A_{j_h}^h(V^{\neq h})) \rightarrow \mathrm{RF}_{j_i}^i(V) = \mathbf{0})}_{\text{current ranking function } \mathrm{RF}_{j_i}^i \text{ is always } \mathbf{0}} \ \vee \\
&\exists V, V' : \underbrace{(\bigwedge_{h=0}^{n} R_{j_h}^h(V) \wedge A_{j_h}^h(V^{\neq h}) \wedge T^h(V,V') \wedge R_{j_h}^h(V') \wedge A_{j_h}^h(V^{\neq h\prime}))}_{V,V' \text{ in same restricted regions of } \widehat{V},\widehat{V}'} \wedge \\
&\underbrace{\mathrm{RF}_{j_i}^i(V') < \mathrm{RF}_{j_i}^i(V) \wedge (\bigwedge_{h=0,h\neq i}^{n} \mathrm{RF}_{j_h}^h(V') = \mathrm{RF}_{j_h}^h(V))}_{\text{current ranking function decreases, all others remain constant}} \wedge \\
&\mathit{compatible}_{\{H^k\}_{k=0}^n}(V,V'))
\end{aligned}
$$

The composition operator for a set of E-comps $\{H^i\}_{i=0}^n$ requires the corresponding sets $\{V^i\}_{i=0}^n$ to be pairwise disjoint. We write $\{V^i\}_{i\notin\{0,\ldots,n\}}$ for the possibly empty list of other sets to complete the partitioning and

$V^{\neq\{i\}_{i\notin\{0,\dots,n\}}}$ for their union. $\{V^i\}_{i=0}^n \cup \{V^i\}_{i\notin\{0,\dots,n\}}$ is a partitioning of V.

Definition 24 - Composition of *E*-comps

We define the composition of a set of E*-comps* $\{H^i\}_{i=0}^n$*, such that the sets of local symbols* $\{V^i\}_{i=0}^n$ *are pairwise disjoint, as* $H^c \doteq \bigotimes_{i=0}^n H^i = \langle V, I^c, T^c\rangle$ *responsible for symbols* V^c *and associated with regions* $\mathcal{R}^c$*, assumptions* $\mathcal{A}^c$ *and ranking functions* $\mathcal{W}^c$*; where:*

- $V^c \doteq \bigcup_{i=0}^n V^i$;
- $\mathcal{R}^c \doteq \{\bigwedge_{i=0}^n R^i_{j_i} \wedge \bigwedge_{h=0,h\neq i}^n A^{i,h}_{j_i} \mid j_i \in \{k\}_{k=0}^{m^i-1} \wedge R^i_{j_i} \in \mathcal{R}^i \wedge A^i_{j_i} \in \mathcal{A}^i \wedge A^{i,h}_{j_i} \in A^i_{j_i}\}$;
- $\mathcal{A}^c \doteq \{\bigwedge_{i=0}^n \bigwedge_{h\notin\{k\}_{k=0}^n} A^{i,h}_{j_i} \mid j_i \in \{k\}_{k=0}^{m^i-1} \wedge A^i_{j_i} \in \mathcal{A}^i \wedge A^{i,h}_{j_i} \in A^i_{j_i}\}$;
- $\mathcal{W}^c \doteq \{\sum_{i=0}^n \mathrm{RF}^i_{j_i} \mid j_i \in \{k\}_{k=0}^{m^i-1} \wedge \mathrm{RF}^i_{j_i} \in \mathcal{W}^i\}$;
- $I^c \doteq \bigwedge_{i=0}^n I^i$;
- $T^c \doteq \mathit{compatible}_{\{H^i\}_{i=0}^n} \wedge \mathit{indepRank}_{\{H^i\}_{i=0}^n} \wedge \bigwedge_{i=0}^n T^i$.

The composition H^c of $n+1$ E-comps $\{H^i\}_{i=0}^n$, is responsible for the symbols V^c defined as the union, for every $0 \leq i \leq n$, of the symbols V^i for which H^i is responsible. The regions, assumptions and initial states of H^c are obtained as the conjunction of the corresponding components of the $\{H^i\}_{i=0}^n$. Each region is defined as the conjunction of a region for each E-comp and the corresponding assumptions over the symbols in V^c. Instead, the assumptions are defined as the conjunction of the remaining assumptions, i.e. the ones over the symbols not in V^c. The ranking functions are obtained as the sum of the ranking functions of the $\{H^i\}_{i=0}^n$. Finally, the transition relation restricts the conjunction of the transition relations of the E-comps to the steps that are both compatible and allow for independent ranks. Therefore, it allows only for transitions that meet all the assumptions of the E-comps $\{H^i\}_{i=0}^n$ on the symbols in V^c.

Theorem 14 - *E*-comps are closed w.r.t. composition
Given a set of E*-comps* $\{H^i\}_{i=0}^n$*, their composition* $H^c \doteq \bigotimes_{i=0}^n H^i = \langle V, I^c, T^c \rangle$ *is an* E*-comps with respect to regions* $\mathcal{R}^c$*, assumptions* $\mathcal{A}^c$ *and ranking functions* $\mathcal{W}^c$*.*

Proof. We prove that hypotheses *EC.1–EC.4* hold for H^c of length m^c. In the following, we write $A_{j_i}^{i,\neq c}$ for $\bigwedge_{h \notin \{k\}_{k=0}^n} A_{j_i}^{i,h}(V^h)$.

- Consider first Hyp. *EC.1*.
  The initial states of H^c are a subset of the union of the regions because, by definition of I^c, every state in this set must satisfy $\bigvee_{j_c=0}^{m^c} R_{j_c}^c \wedge A_{j_c}^c$.
- Hyp. *EC.2* requires us to prove the following:

  $\forall j : 0 \leq j < m^c \rightarrow$
  $\quad \exists V, V' : (R_j^c \wedge A_j^c \wedge T^c \wedge \mathrm{RF}_j^{c\prime} < \mathrm{RF}_j^c \wedge R_j^{c\prime} \wedge A_j^{c\prime}) \quad \models$
  $\quad \forall V \exists V^{c\prime} \forall V^{\neq c\prime} : R_j^c \wedge A_j^c \wedge \mathbf{0} < \mathrm{RF}_j^c \wedge A_j^{c\prime} \rightarrow R_j^{c\prime} \wedge T^c \wedge \mathrm{RF}_j^{c\prime} < \mathrm{RF}_j^c.$

  $H^c \doteq \bigotimes_{i=0}^n H^i$ hence, by definition of $\otimes$, R_j^c and A_j^c are the conjunction of some region and assumptions of $\{H^i\}_{i=0}^n$. Therefore, we can rewrite it as follows:

  $$\forall \{j_i\}_{i=0}^n : \ (\bigwedge_{i=0}^{n} 0 \leq j_i < m^i) \rightarrow$$
  $$\exists V, V' : (\bigwedge_{i=0}^{n} R_{j_i}^i \wedge (\bigwedge_{h=0, h\neq i}^{n} A_{j_i}^{i,h}) \wedge A_{j_i}^{i,\neq c} \wedge T^i) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^n} \wedge$$
  $$\mathit{indepRank}_{\{H^i\}_{i=0}^n} \wedge \mathrm{RF}_j^{c\prime} < \mathrm{RF}_j^c \wedge (\bigwedge_{i=0}^{n} R_{j_i}^{i\ \prime} \wedge (\bigwedge_{h=0, h\neq i}^{n} A_{j_i}^{i,h\prime}) \wedge A_{j_i}^{i,\neq c\prime}) \models$$
  $$\forall V \exists \{V^{i\prime}\}_{i=0}^n \forall V^{\neq c\prime} : ((\bigwedge_{i=0}^{n} R_{j_i}^i \wedge (\bigwedge_{h=0, h\neq i}^{n} A_{j_i}^{i,h}) \wedge A_{j_i}^{i,\neq c}) \wedge A_j^{c\prime} \wedge \mathbf{0} < \mathrm{RF}_j^c) \rightarrow$$
  $$((\bigwedge_{i=0}^{n} R_{j_i}^{i\ \prime} \wedge (\bigwedge_{h=0, h\neq i}^{n} A_{j_i}^{i,h\prime}) \wedge T^i) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^n} \wedge \mathit{indepRank}_{\{H^i\}_{i=0}^n} \wedge$$
  $$\mathrm{RF}_j^{c\prime} < \mathrm{RF}_j^c).$$

For any $0 \leq i \leq n$ $A^i_j(V^{\neq c}) \wedge \bigwedge^n_{h=0,h\neq i} A^{i,h}_{j_i}(V^h)$ is equivalent to $A^i_j(V^{\neq i})$. Therefore, our objective formula can be rewritten as:

$$\forall\{j_i\}^n_{i=0} : (\bigwedge_{i=0}^{n} 0 \leq j_i < m^i) \rightarrow$$

$$\exists V, V' : (\bigwedge_{i=0}^{n} R^i_{j_i} \wedge A^i_{j_i} \wedge T^i \wedge R^{i\,\prime}_{j_i} \wedge A^{i\,\prime}_{j_i}) \wedge \mathit{compatible}_{\{H^i\}^n_{i=0}} \wedge$$

$$\mathit{indepRank}_{\{H^i\}^n_{i=0}} \wedge \mathrm{RF}^{c\prime}_j < \mathrm{RF}^c_j \quad \models$$

$$\forall V \exists\{V^{i\prime}\}^n_{i=0} \forall V^{\neq c\prime} : ((\bigwedge_{i=0}^{n} R^i_{j_i} \wedge A^i_{j_i}) \wedge A^{c\prime}_j \wedge \mathbf{0} < \mathrm{RF}^c_j) \rightarrow ((\bigwedge_{i=0}^{n} T^i \wedge R^{i\,\prime}_{j_i} \wedge$$

$$\bigwedge_{h=0,h\neq i}^{n} A^{i,h\prime}_{j_i}) \wedge \mathit{compatible}_{\{H^i\}^n_{i=0}} \wedge \mathit{indepRank}_{\{H^i\}^n_{i=0}} \wedge \mathrm{RF}^{c\prime}_j < \mathrm{RF}^c_j).$$

If $\mathit{indepRank}_{\{H^i\}^n_{i=0}}$ [resp. $\mathit{compatible}_{\{H^i\}^n_{i=0}}$] does not hold in the left-hand-side of the entailment the formula is trivially true. By definition of $\mathit{indepRank}_{\{H^i\}^n_{i=0}}$ [resp. $\mathit{compatible}_{\{H^i\}^n_{i=0}}$], if it holds in the left-hand-side of the entailment it must also hold on the right-hand-side, since on both sides V and V' belong to the same regions. Therefore, $\mathit{compatible}_{\{H^i\}^n_{i=0}}$ must hold and when both sides of the implication on the right-hand-side of the entailment hold, $\bigwedge^n_{i=0} \bigwedge^n_{h=0,h\neq i} A^{i,h}_{j'_i}(V^{h\prime})$ must be true. We can further simplify our objective formula as follows:

$$\forall\{j_i\}^n_{i=0} : (\bigwedge_{i=0}^{n} 0 \leq j_i < m^i) \rightarrow$$

$$\exists V, V' : (\bigwedge_{i=0}^{n} R^i_{j_i} \wedge A^i_{j_i} \wedge T^i \wedge R^{i\,\prime}_{j_i} \wedge A^{i\,\prime}_{j_i}) \wedge \mathit{compatible}_{\{H^i\}^n_{i=0}} \wedge$$

$$\mathit{indepRank}_{\{H^i\}^n_{i=0}} \wedge \mathrm{RF}^{c\prime}_j < \mathrm{RF}^c_j \quad \models \forall V \exists\{V^{i\prime}\}^n_{i=0} \forall V^{\neq c\prime} :$$

$$((\bigwedge_{i=0}^{n} R^i_{j_i} \wedge A^i_{j_i}) \wedge A^{c\prime}_j \wedge \mathbf{0} < \mathrm{RF}^c_j) \rightarrow ((\bigwedge_{i=0}^{n} T^i \wedge R^{i\,\prime}_{j_i}) \wedge \mathrm{RF}^{c\prime}_j < \mathrm{RF}^c_j).$$

If the left-hand-side of the entailment is false, then the formula is triv-

ially true. Therefore, assume that there exists a transition performing a self-loop on the restricted region $R_j^c \wedge A_j^c$ with *independent ranks* in which the sum of the ranking function decreases. Under this assumption, we need to prove the following for any $j \doteq \langle j_0, \ldots, j_n \rangle$ satisfying the above:

$$\forall V \exists \{V^{i\prime}\}_{i=0}^{n} \forall V^{\neq c\prime} :$$
$$((\bigwedge_{i=0}^{n} R_{j_i}^i \wedge A_{j_i}^i) \wedge A_j^{c\prime} \wedge \mathbf{0} < \mathrm{RF}_j^c(V)) \rightarrow ((\bigwedge_{i=0}^{n} T^i \wedge R_{j_i}^{i\,\prime}) \wedge \mathrm{RF}_j^{c\prime} < \mathrm{RF}_j^c).$$

Since $\mathit{indepRank}_{\{H^i\}_{i=0}^n}$ holds for indexes $\langle j_0, \ldots, j_n \rangle$ we have:

$$\bigwedge_{i=0}^{n} (\forall V : (\bigwedge_{h=0}^{n} R_{j_h}^h \wedge A_{j_h}^h) \rightarrow \mathrm{RF}_{j_i}^i = \mathbf{0}) \vee$$
$$\exists V, V' : (\bigwedge_{h=0}^{n} R_{j_h}^h \wedge A_{j_h}^h \wedge T^h \wedge R_{j_h}^{h\,\prime} \wedge A_{j_h}^{h\,\prime}) \wedge$$
$$\mathrm{RF}_{j_i}^{i\,\prime} < \mathrm{RF}_{j_i}^i \wedge (\bigwedge_{k=0, k\neq h}^{n} \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^i) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^n}.$$

In addition, since there exists a transition in the restricted regions such that RF_j^c decreases, there must be some $0 \leq i_r \leq n$ such that $\exists V : (\bigwedge_{h=0}^{n} R_{j_h}^h(V) \wedge A_{j_h}^h(V^{\neq h})) \wedge \mathbf{0} < \mathrm{RF}_{j_{ir}}^{i_r}(V)$. Then, there exist compatible transitions in which its ranking function decreases $\mathrm{RF}_{j_r}^{i_r}(V') < \mathrm{RF}_{j_r}^{i_r}(V)$, while all other ranking function remain constant, written $\bigwedge_{i=0, i\neq i_r}^{n} \mathrm{RF}_{j_i}^i(V') = \mathrm{RF}_{j_i}^i(V)$. Hyp. *EC.2* holds for H^{i_r}:

$$\forall j_r \in \{k\}_{k=0}^{m^{i_r}-1} : \exists V, V' : (R_{j_r}^{i_r} \wedge A_{j_r}^{i_r} \wedge T^{i_r} \wedge \mathrm{RF}_{j_r}^{i_r\,\prime} < \mathrm{RF}_{j_r}^{i_r} \wedge R_{j_r}^{i_r\,\prime} \wedge A_{j_r}^{i_r\,\prime}) \models$$
$$\forall V \exists V^{i_r\prime} \forall V^{\neq i_r\prime} : R_{j_r}^{i_r} \wedge A_{j_r}^{i_r} \wedge \mathbf{0} < \mathrm{RF}_{j_r}^{i_r} \wedge A_{j_r}^{i_r\,\prime} \rightarrow R_{j_r}^{i_r\,\prime} \wedge T^{i_r} \wedge \mathrm{RF}_{j_r}^{i_r\,\prime} < \mathrm{RF}_{j_r}^{i_r}.$$

and Hyp. *EC.3* holds for all $\{H^i\}_{i=0, i\neq i_r}^{n}$:

$$\forall j_i \in \{k\}_{k=0}^{m^i-1} : \exists V, V' : (R_{j_i}^i \wedge A_{j_i}^i \wedge T^i \wedge \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^i \wedge R_{j_i}^{i\,\prime} \wedge A_{j_i}^{i\,\prime}) \models$$
$$\forall V \exists V^{i\prime} \forall V^{\neq i\prime} : R_{j_i}^i \wedge A_{j_i}^i \wedge A_{j_i}^{i\,\prime} \rightarrow R_{j_i}^{i\,\prime} \wedge T^i \wedge \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^i.$$

If there is no transition in the intersection of the restricted regions such that RF_j^c decreases or they are not compatible, the objective formula trivially holds because the left-hand-side of the entailment is false. Then, the conjunction of the hypotheses for the $\{H^i\}_{i=0}^n$ implies:

$$\forall\{j_i\}_{i=0}^n \ : \ (\bigwedge_{i=0}^{n} 0 \leq j_i < m^i) \rightarrow$$
$$\exists V, V' : (\bigwedge_{i=0}^{n}(R_{j_i}^i \wedge A_{j_i}^i \wedge T^i \wedge R_{j_i}^{i\,\prime} \wedge A_{j_i}^{i\,\prime})\wedge$$
$$\mathrm{RF}_{j_r}^{i_r\prime} < \mathrm{RF}_{j_r}^{i_r} \wedge \bigwedge_{i=0, i\neq r}^{n} \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^{i}) \quad \models$$
$$\forall V \exists V^{i_r\prime} \forall V^{\neq i_r\prime} :$$
$$R_{j_r}^{i_r} \wedge A_{j_r}^{i_r} \wedge \mathbf{0} < \mathrm{RF}_{j_r}^{i_r} \wedge A_{j_r}^{i_r\prime} \rightarrow R_{j_r}^{i_r\prime} \wedge T^{i_r} \wedge \mathrm{RF}_{j_r}^{i_r\prime} < \mathrm{RF}_{j_r}^{i_r} \wedge$$
$$\bigwedge_{i=0, i\neq r}^{n} \forall V \exists V^{i\prime} \forall V^{\neq i\prime} : R_{j_i}^i \wedge A_{j_i}^i \wedge A_{j_i}^{i\,\prime} \rightarrow R_{j_i}^{i\,\prime} \wedge T^i \wedge \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^{i}.$$

The left hand side of the entailment must hold, otherwise our objective formula is trivially true.

$$\forall V \exists V^{i_r\prime} \forall V^{\neq i_r\prime} :$$
$$R_{j_r}^{i_r} \wedge A_{j_r}^{i_r} \wedge \mathbf{0} < \mathrm{RF}_{j_r}^{i_r} \wedge A_{j_r}^{i_r\prime} \rightarrow R_{j_r}^{i_r\prime} \wedge T^{i_r} \wedge \mathrm{RF}_{j_r}^{i_r\prime} < \mathrm{RF}_{j_r}^{i_r} \wedge$$
$$\bigwedge_{i=0, i\neq r}^{n} \forall V \exists V^{i\prime} \forall V^{\neq i\prime} : R_{j_i}^i \wedge A_{j_i}^i \wedge A_{j_i}^{i\,\prime} \rightarrow R_{j_i}^{i\,\prime} \wedge T^i \wedge \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^{i}.$$

If a $\forall V \exists V^{i\prime} \forall V^{\neq i\prime}$ quantified implication holds, then for every assignment to the symbols V such that $R_{j_i}^i(V)$, $A_{j_i}^i(V^{\neq i})$ and, if $i = i_r$, also $\mathbf{0} < \mathrm{RF}_{j_r}^{i_r}(V)$ hold, there exists an assignment to $V^{i\prime}$ satisfying the assumptions of all other E-comps $\bigwedge_{s=0, s\neq i}^{n} A_{j_s}^{s,i}(V^{i\prime})$, for all assignments

to the $V^{\neq i'}$. Therefore, we can write the following:

$$\begin{aligned}
&\forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} : \\
&(R_{j_r}^{i_r} \wedge A_{j_r}^{i_r} \wedge \mathbf{0} < \mathrm{RF}_{j_r}^{i_r} \wedge A_{j_r}^{i_r,\neq c'} \rightarrow {R_{j_r}^{i_r}}' \wedge T^{i_r} \wedge \mathrm{RF}_{j_r}^{i_r\,\prime} < \mathrm{RF}_{j_r}^{i_r}) \wedge \\
&\bigwedge_{i=0, i\neq r}^{n} R_{j_i}^{i} \wedge A_{j_i}^{i} \wedge A_{j_i}^{i,\neq c'} \rightarrow {R_{j_i}^{i}}' \wedge T^{i} \wedge \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^{i}.
\end{aligned}$$

$\mathbf{0} < \mathrm{RF}_{j_r}^{i_r}(V)$ implies $\mathbf{0} < \mathrm{RF}_{j}^{c}(V)$ and, since $(a \rightarrow b) \wedge (c \rightarrow d)$ implies $(a \wedge c) \rightarrow (b \wedge d)$, the formula above implies:

$$\begin{aligned}
&\forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} : (\mathbf{0} < \mathrm{RF}_{j}^{c} \wedge (\bigwedge_{i=0}^{n} R_{j_i}^{i} \wedge A_{j_i}^{i} \wedge A_{j_i}^{i,\neq c'})) \rightarrow \\
&\mathrm{RF}_{j_r}^{i_r\,\prime} < \mathrm{RF}_{j_r}^{i_r} \wedge (\bigwedge_{i=0, i\neq i_r}^{n} \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^{i}) \wedge \bigwedge_{i=0}^{n} {R_{j_i}^{i}}' \wedge T^{i}.
\end{aligned}$$

The formula $\mathrm{RF}_{j_r}^{i_r}(V') < \mathrm{RF}_{j_r}^{i_r}(V)) \wedge (\bigwedge_{i=0, i\neq i_r}^{n} \mathrm{RF}_{j_i}^{i}(V') = \mathrm{RF}_{j_i}^{i}(V))$ implies $\mathrm{RF}_{j}^{c}(V') < \mathrm{RF}_{j}^{c}(V)$ and $\bigwedge_{i=0}^{n} A_{j_i}^{i}(V^{\neq c})$ is equivalent to $A_{j}^{c}(V^{\neq c})$ Therefore, we obtain the implied statement:

$$\begin{aligned}
&\forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} : \\
&\quad ((\bigwedge_{i=0}^{n} R_{j_i}^{i} \wedge A_{j_i}^{i}) \wedge A_{j}^{c'} \wedge \mathbf{0} < \mathrm{RF}_{j}^{c}) \rightarrow ((\bigwedge_{i=0}^{n} T^{i} \wedge {R_{j_i}^{i}}') \wedge \mathrm{RF}_{j}^{c\,\prime} < \mathrm{RF}_{j}^{c}).
\end{aligned}$$

which is exactly the formula we wanted to prove.

- Hyp. *EC.3* requires us to prove the following:

$$\begin{aligned}
&\forall j : 0 \leq j < m^{c} \rightarrow \\
&\quad \exists V, V' : (R_{j}^{c} \wedge A_{j}^{c} \wedge T^{c} \wedge \mathrm{RF}_{j}^{c\,\prime} = \mathrm{RF}_{j}^{c} \wedge R_{j}^{c\,\prime} \wedge A_{j}^{c\,\prime}) \quad \models \\
&\quad \forall V \exists V^{c\,\prime} \forall V^{\neq c'} : R_{j}^{c} \wedge A_{j}^{c} \wedge A_{j}^{c\,\prime} \rightarrow R_{j}^{c\,\prime} \wedge T^{c} \wedge \mathrm{RF}_{j}^{c\,\prime} = \mathrm{RF}_{j}^{c}.
\end{aligned}$$

By definition of $\otimes$ and since $H^c \doteq \bigotimes_{i=0}^{n} H^i$ we can rewrite it as:

$$\forall \{j_i\}_{i=0}^{n} : \ (\bigwedge_{i=0}^{n} 0 \leq j_i < m^i) \rightarrow$$

$$\exists V, V' : (\bigwedge_{i=0}^{n} R^i_{j_i} \wedge (\bigwedge_{h=0, h\neq i}^{n} A^{i,h}_{j_i}) \wedge A^{i,\neq c}_{j_i} \wedge T^i) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^{n}} \wedge$$

$$\mathit{indepRank}_{\{H^i\}_{i=0}^{n}} \wedge \mathrm{RF}^{c'}_j = \mathrm{RF}^{c}_j \wedge (\bigwedge_{i=0}^{n} {R^i_{j_i}}' \wedge (\bigwedge_{h=0, h\neq i}^{n} A^{i,h'}_{j_i}) \wedge A^{i,\neq c'}_{j_i}) \models$$

$$\forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} : ((\bigwedge_{i=0}^{n} R^i_{j_i} \wedge (\bigwedge_{h=0, h\neq i}^{n} A^{i,h}_{j_i}) \wedge A^{i,\neq c}_{j_i}) \wedge A^{c'}_{j}) \rightarrow$$

$$((\bigwedge_{i=0}^{n} {R^i_{j_i}}' \wedge (\bigwedge_{h=0, h\neq i}^{n} A^{i,h'}_{j_i}) \wedge T^i) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^{n}} \wedge$$

$$\mathit{indepRank}_{\{H^i\}_{i=0}^{n}} \wedge \mathrm{RF}^{c'}_j = \mathrm{RF}^{c}_j).$$

On both sides of the entailment $\mathrm{RF}^{c}_j(V') = \mathrm{RF}^{c}_j(V)$ holds, hence $\mathit{indepRank}_{\{H^i\}_{i=0}^{n}}$ is trivially true: the left-hand-side of the implication in its definition is false. In addition, for any $0 \leq i \leq n$ $A^i_j(V^{\neq c}) \wedge \bigwedge_{h=0, h\neq i}^{n} A^{i,h}_{j_i}(V^h)$ is equivalent to $A^i_j(V^{\neq i})$. Therefore, our objective formula can be rewritten as:

$$\forall \{j_i\}_{i=0}^{n} : \ (\bigwedge_{i=0}^{n} 0 \leq j_i < m^i) \rightarrow$$

$$\exists V, V' : (\bigwedge_{i=0}^{n} R^i_{j_i} \wedge A^i_{j_i} \wedge T^i \wedge {R^i_{j_i}}' \wedge {A^i_{j_i}}') \wedge \mathrm{RF}^{c'}_j = \mathrm{RF}^{c}_j \wedge$$

$$\mathit{compatible}_{\{H^i\}_{i=0}^{n}} \quad \models \forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} :$$

$$((\bigwedge_{i=0}^{n} R^i_{j_i} \wedge A^i_{j_i}) \wedge A^{c'}_{j}) \rightarrow ((\bigwedge_{i=0}^{n} T^i \wedge {R^i_{j_i}}' \wedge \bigwedge_{h=0, h\neq i}^{n} A^{i,h'}_{j_i}) \wedge$$

$$\mathrm{RF}^{c'}_j = \mathrm{RF}^{c}_j \wedge \mathit{compatible}_{\{H^i\}_{i=0}^{n}}).$$

If $\mathit{compatible}_{\{H^i\}_{i=0}^{n}}(V, V')$ does not hold, then the left-hand-side of the entailment is false, hence the entailment is true.

Otherwise, $\mathit{compatible}_{\{H^i\}_{i=0}^n}$ holds and since it holds on the left-hand-side of the entailment, it must also hold on the right-hand-side; when both sides of the implication on the right-hand-side of the entailment hold, $\bigwedge_{i=0}^n \bigwedge_{h=0,h\neq i}^n A_{j_i'}^{i,h}(V^{h\prime})$ must be true since $\mathit{compatible}_{\{H^i\}_{i=0}^n}$ holds. We can further simplify our objective formula as follows:

$$\begin{aligned}
\forall\{j_i\}_{i=0}^n :\ & (\bigwedge_{i=0}^n 0 \leq j_i < m^i) \rightarrow \\
& \exists V, V' : (\bigwedge_{i=0}^n R_{j_i}^i \wedge A_{j_i}^i \wedge T^i \wedge R_{j_i}^{i\,\prime} \wedge A_{j_i}^{i\,\prime}) \wedge \mathrm{RF}_j^{c\prime} = \mathrm{RF}_j^c \wedge \\
& \quad \mathit{compatible}_{\{H^i\}_{i=0}^n} \quad \models \forall V \exists \{V^{i\prime}\}_{i=0}^n \forall V^{\neq c\prime} : \\
& \quad ((\bigwedge_{i=0}^n R_{j_i}^i \wedge A_{j_i}^i) \wedge A_j^{c\prime})) \rightarrow ((\bigwedge_{i=0}^n T^i \wedge R_{j_i}^{i\,\prime}) \wedge \mathrm{RF}_j^{c\prime} = \mathrm{RF}_j^c).
\end{aligned}$$

If the left-hand-side of the entailment is false, then the formula is trivially true. Therefore, assume that there exists a transition performing a self-loop on the restricted region $R_j^c \wedge A_j^c$ in which the ranking function remains constant. Under this assumption, we need to prove the following for any $j \doteq \langle j_0, \ldots, j_n \rangle$ satisfying the above:

$$\forall V \exists \{V^{i\prime}\}_{i=0}^n \forall V^{\neq c\prime} : ((\bigwedge_{i=0}^n R_{j_i}^i \wedge A_{j_i}^i) \wedge A_j^{c\prime}) \rightarrow (\mathrm{RF}_j^{c\prime} = \mathrm{RF}_j^c \wedge \bigwedge_{i=0}^n T^i \wedge R_{j_i}^{i\,\prime}).$$

Hyp. *EC.3* holds for all E-comps $\{H^i\}_{i=0}^n$:

$$\begin{aligned}
\forall j_i : 0 \leq j_i < m^i \rightarrow & \\
& \exists V, V' : (R_{j_i}^i \wedge A_{j_i}^i \wedge T^i \wedge \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^i \wedge R_{j_i}^{i\,\prime} \wedge A_{j_i}^{i\,\prime}) \quad \models \\
& \forall V \exists V^{i\prime} \forall V^{\neq i\prime} : R_{j_i}^i \wedge A_{j_i}^i \wedge A_{j_i}^{i\,\prime} \rightarrow R_{j_i}^{i\,\prime} \wedge T^i \wedge \mathrm{RF}_{j_i}^{i\,\prime} = \mathrm{RF}_{j_i}^i.
\end{aligned}$$

By assumption there exists a transition in the intersection of their restricted regions such that $\mathrm{RF}_j^c(V') = \mathrm{RF}_j^c(V)$, and hence $\mathrm{RF}_{j_i}^i(V') =$

$\mathrm{RF}^i_{j_i}(V)$ for all i. Therefore, their conjunction implies:

$$\bigwedge_{i=0}^{n} \forall V \exists V^{i\prime} \forall V^{\neq i\prime} : R^i_{j_i} \wedge A^i_{j_i} \wedge {A^i_{j_i}}' \rightarrow {R^i_{j_i}}' \wedge T^i \wedge {\mathrm{RF}^i_{j_i}}' = \mathrm{RF}^i_{j_i}.$$

If a $\forall V \exists V^{i\prime} \forall V^{\neq i\prime}$ quantified implication holds then for every assignment to the symbols V such that $R^i_{j_i}(V) \wedge A^i_{j_i}(V^{\neq i}) \wedge A^i_{j_i}(V^{\neq i\prime})$ holds, there exists an assignment to the $V^{i\prime}$ satisfying the assumptions of all other E-comps $\bigwedge_{s=0,s\neq i}^{n} A^{s,i}_{j_s}(V^{i\prime})$, for all assignments to the $V^{\neq i\prime}$. Therefore, we can write the following:

$$\forall V \exists \{V^{i\prime}\}_{i=0}^{n} \forall V^{\neq c\prime} : \bigwedge_{i=0}^{n} ((R^i_{j_i} \wedge A^i_{j_i} \wedge A^{i,\neq c\prime}_{j_i}) \rightarrow ({R^i_{j_i}}' \wedge T^i \wedge {\mathrm{RF}^i_{j_i}}' = \mathrm{RF}^i_{j_i})).$$

Since $(a \rightarrow b) \wedge (c \rightarrow d)$ implies $(a \wedge c) \rightarrow (b \wedge d)$ and $\bigwedge_{i=0}^{n} \mathrm{RF}^i_{j_i}(V') = \mathrm{RF}^i_{j_i}(V)$ implies $\mathrm{RF}^c_j(V') = \mathrm{RF}^c_j(V)$, the formula above implies:

$$\forall V \exists \{V^{i\prime}\}_{i=0}^{n} \forall V^{\neq c\prime} : (\bigwedge_{i=0}^{n} R^i_{j_i} \wedge A^i_{j_i}) \wedge A^{c\prime}_j \rightarrow (\mathrm{RF}^{c\prime}_j = \mathrm{RF}^c_j \wedge \bigwedge_{i=0}^{n} T^i \wedge {R^i_{j_i}}').$$

which is exactly the formula we wanted to prove.

- Hyp. *EC.4* requires us to prove the following:

$$\begin{aligned}
&\forall j, j' : \ 0 \leq j < m^c \wedge 0 \leq j' < m^c \rightarrow \\
&\exists V, V' : (R^c_j \wedge A^c_j \wedge T^c \wedge \mathrm{RF}^c_j = \mathbf{0} \wedge {R^c_{j'}}' \wedge {A^c_{j'}}') \quad \models \\
&\forall V \exists V^{c\prime} \forall V^{\neq c\prime} : R^c_j \wedge A^c_j \wedge \mathrm{RF}^c_j = \mathbf{0} \wedge {A^c_{j'}}' \rightarrow {R^c_{j'}}' \wedge T^c.
\end{aligned}$$

By definition of $\otimes$ and since $H^c \doteq \bigotimes_{i=0}^{n} H^i$ we can rewrite it as:

$$\forall \{j_i\}_{i=0}^{n}, \{j_i'\}_{i=0}^{n} : (\bigwedge_{i=0}^{n} 0 \leq j_i < m^i \wedge 0 \leq j_i' < m^i) \rightarrow$$

$$\exists V, V' : (\bigwedge_{i=0}^{n} R_{j_i}^i \wedge (\bigwedge_{h=0, h\neq i}^{n} A_{j_i}^{i,h}) \wedge A_{j_i}^{i,\neq c} \wedge T^i) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^{n}} \wedge$$

$$\mathit{indepRank}_{\{H^i\}_{i=0}^{n}} \wedge \mathrm{RF}_j^c = \mathbf{0} \wedge (\bigwedge_{i=0}^{n} {R_{j_i'}^i}' \wedge (\bigwedge_{h=0, h\neq i}^{n} A_{j_i'}^{i,h'}) \wedge A_{j_i'}^{i,\neq c'}) \models$$

$$\forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} :$$

$$((\bigwedge_{i=0}^{n} R_{j_i}^i \wedge (\bigwedge_{h=0, h\neq i}^{n} A_{j_i}^{i,h}) \wedge A_{j_i}^{i,\neq c}) \wedge \mathrm{RF}_j^c = \mathbf{0} \wedge {A_{j'}^c}') \rightarrow$$

$$((\bigwedge_{i=0}^{n} {R_{j_i'}^i}' \wedge T^i \wedge \bigwedge_{h=0, h\neq i}^{n} A_{j_i'}^{i,h'}) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^{n}} \wedge \mathit{indepRank}_{\{H^i\}_{i=0}^{n}}).$$

If $j \neq j'$, $\mathit{indepRank}_{\{H^i\}_{i=0}^{n}}$ trivially holds, since the left-hand-side of the implication in its definition is false. Otherwise, if $j = j'$, $\mathrm{RF}_j^c(V) = \mathbf{0}$ contradicts $\mathrm{RF}_j^c(V') < \mathrm{RF}_j^c(V)$ and again $\mathit{indepRank}_{\{H^i\}_{i=0}^{n}}$ trivially holds because the left-hand-side of the implication in its definition is false. In addition, for any $0 \leq i \leq n$ $A_j^i(V^{\neq c}) \wedge \bigwedge_{h=0, h\neq i}^{n} A_{j_i}^{i,h}(V^h)$ is equivalent to $A_j^i(V^{\neq i})$. Therefore, our objective formula can be rewritten as:

$$\forall \{j_i\}_{i=0}^{n}, \{j_i'\}_{i=0}^{n} : (\bigwedge_{i=0}^{n} 0 \leq j_i < m^i \wedge 0 \leq j_i' < m^i) \rightarrow$$

$$\exists V, V' : (\bigwedge_{i=0}^{n} R_{j_i}^i \wedge A_{j_i}^i \wedge T^i \wedge {R_{j_i'}^i}' \wedge {A_{j_i'}^i}') \wedge \mathit{compatible}_{\{H^i\}_{i=0}^{n}} \wedge \mathrm{RF}_j^c = \mathbf{0} \models$$

$$\forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} : ((\bigwedge_{i=0}^{n} R_{j_i}^i \wedge A_{j_i}^i) \wedge \mathrm{RF}_j^c = \mathbf{0} \wedge {A_{j'}^c}') \rightarrow$$

$$((\bigwedge_{i=0}^{n} T^i \wedge {R_{j_i'}^i}' \wedge \bigwedge_{h=0, h\neq i}^{n} A_{j_i'}^{i,h'}) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^{n}}).$$

If $\mathit{compatible}_{\{H^i\}_{i=0}^n}(V, V')$ does not hold, then the left-hand-side of the entailment is false, hence the entailment is true.
Otherwise $\mathit{compatible}_{\{H^i\}_{i=0}^n}$ holds and since it holds on the left-hand-side of the entailment, it must also hold on the right-hand-side; when both sides of the implication on the right-hand-side of the entailment hold, $\bigwedge_{i=0}^n \bigwedge_{h=0,h\neq i}^n A_{j_i'}^{i,h}(V^{h\prime})$ must be true since $\mathit{compatible}_{\{H^i\}_{i=0}^n}$ holds. We can further simplify our objective formula as follows:

$$\forall \{j_i\}_{i=0}^n, \{j_i'\}_{i=0}^n : (\bigwedge_{i=0}^n 0 \leq j_i < m^i \wedge 0 \leq j_i' < m^i) \rightarrow$$

$$\exists V, V' : (\bigwedge_{i=0}^n R_{j_i}^i \wedge A_{j_i}^i \wedge T^i \wedge R_{j_i'}^{i\prime} \wedge A_{j_i'}^{i\prime}) \wedge \mathit{compatible}_{\{H^i\}_{i=0}^n} \wedge \mathrm{RF}_j^c = \mathbf{0} \models$$

$$\forall V \exists \{V^{i\prime}\}_{i=0}^n \forall V^{\neq c\prime} : (\mathrm{RF}_j^c = \mathbf{0} \wedge \bigwedge_{i=0}^n R_{j_i}^i \wedge A_{j_i}^i \wedge A_{j_i'}^{i,\neq c\prime}) \rightarrow (\bigwedge_{i=0}^n T^i \wedge R_{j_i'}^{i\prime}).$$

If the left-hand-side of the entailment is false, then the formula is trivially true. Therefore, assume that there exists a transition from a state in $R_j^c \wedge A_j^c \wedge \mathrm{RF}_j^c = \mathbf{0}$ to $R_{j'}^c \wedge A_{j'}^c$ Under this assumption, we need to prove the following for any $j \doteq \langle j_0, \ldots, j_n \rangle$ satisfying the above:

$$\forall V \exists \{V^{i\prime}\}_{i=0}^n \forall V^{\neq c\prime} : (\mathrm{RF}_j^c = \mathbf{0} \wedge \bigwedge_{i=0}^n R_{j_i}^i \wedge A_{j_i}^i \wedge A_{j_i'}^{i,\neq c\prime}) \rightarrow (\bigwedge_{i=0}^n T^i \wedge R_{j_i'}^{i\prime}).$$

Each E-comp H^i allows for a transition from its restricted region with index j_i to the one with index j_i'. In this transition since $\mathrm{RF}_j^c(V) = \mathbf{0}$, then $\mathrm{RF}_{j_i}^i(V) = \mathbf{0}$ holds in the source state. The following holds since Hyp. *EC.4* holds for all $\{H^i\}_{i=0}^n$.

$$\begin{aligned} &\exists V, V' : (R_{j_i}^i \wedge A_{j_i}^i \wedge T^i \wedge \mathrm{RF}_{j_i}^i = \mathbf{0} \wedge R_{j_i'}^{i\prime} \wedge A_{j_i'}^{i\prime}) \quad \models \\ &\forall V \exists V^{i\prime} \forall V^{\neq i\prime} : (R_{j_i}^i \wedge A_{j_i}^i \wedge \mathrm{RF}_{j_i}^i = \mathbf{0} \wedge A_{j_i'}^{i\prime}) \rightarrow R_{j_i'}^{i\prime} \wedge T^i. \end{aligned}$$

If a $\forall V \exists V^{i\prime} \forall V^{\neq i\prime}$ quantified implication holds then for every assignment to the symbols V such that $R_{j_i}^i(V) \wedge A_{j_i}^i(V^{\neq i}) \wedge A_{j_i'}^i(V^{\neq i\prime})$ holds,

there exists an assignment to the $V^{i'}$ satisfying the assumptions of all other E-comps $\bigwedge_{s=0,s\neq i}^{n} A_{j'_s}^{s,i}(V^{i'})$, for all assignments to the $V^{\neq i'}$. Therefore, we can write the following:

$$\forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} : \bigwedge_{i=0}^{n}((R_{j_i}^{i} \wedge A_{j_i}^{i} \wedge \mathrm{RF}_{j_i}^{i} = \mathbf{0} \wedge A_{j'_i}^{i,\neq c'}) \rightarrow (R_{j'_i}^{i\,\prime} \wedge T^{i})).$$

Since $(a \rightarrow b) \wedge (c \rightarrow d)$ implies $(a \wedge c) \rightarrow (b \wedge d)$ and $\bigwedge_{i=0}^{n} \mathrm{RF}_{j_i}^{i}(V) = \mathbf{0}$ implies $\mathrm{RF}_{j}^{c}(V) = \mathbf{0}$, we can write the following implied statement:

$$\forall V \exists \{V^{i'}\}_{i=0}^{n} \forall V^{\neq c'} : (\mathrm{RF}_{j}^{c} = \mathbf{0} \wedge \bigwedge_{i=0}^{n} R_{j_i}^{i} \wedge A_{j_i}^{i} \wedge A_{j'_i}^{i,\neq c'}) \rightarrow (\bigwedge_{i=0}^{n} T^{i} \wedge R_{j'_i}^{i\,\prime}).$$

which is exactly the formula we wanted to prove.

□

We consider only simple interactions between the ranking functions of different E-comps. It is possible to extend the operator to allow for more complex combinations such as nesting of ranking functions or allowing the ranking function of an E-comp to decrease once every time all the other E-comps perform a loop over their regions. However, including these kinds of compositions would make the definitions and proofs much more complex and with many more cases to be considered.

6.3.5 Example Decomposition: Composing *E*-comps

We exemplify how these operators can be used to combine E-comps via two different examples. First, in 6.3.5, we consider a hybrid system describing a bouncing ball and show how it can be described as a composition of E-comps. Then, in 6.3.5, we consider the fair transition system Ex, defined in Sec. 6.1, and obtain an E-comp proving the existence of at least one fair path by composing the E-comps we defined in §6.3.2.

Bouncing ball example

We now show how the E-comp in Fig. 6.8 can be represented as composition of smaller E-comps. The E-comp describes the evolution of a bouncing ball subjected to the constant acceleration g. Its behaviour is defined in terms of the height h, velocity v and the elapse of time is given by the sum of the δ; while c counts the number of bounces. The transition from R_1 to R_0 corresponds to a bounce of the ball, while the transition from R_0 to R_1 corresponds to the motion of the ball between consecutive bounces. We define three E-comps where all ranking functions are always equal to

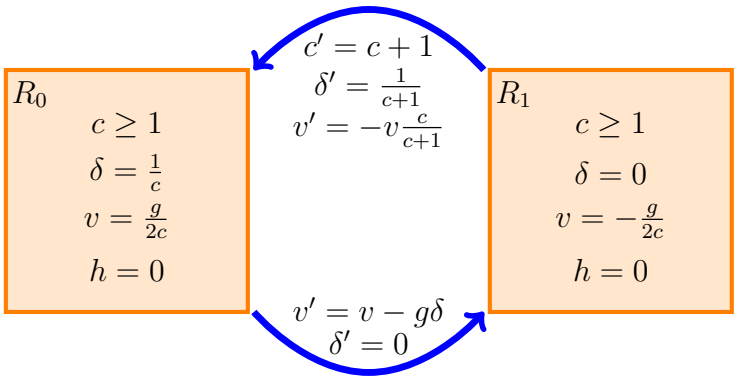

Figure 6.8: E-comp with no assumptions.

their minimal element, hence we will avoid to explicitly define the set of ranking functions for each of them. We consider the partitioning of V given by $V^C \doteq \{c\}$, $V^H \doteq \{h\}$ and $V^{DV} \doteq \{\delta, v\}$. We define three corresponding E-comps C, H and DV as follows.

$$C \doteq \langle V, c \geq 1, c' = c + 1 \vee c' = c \rangle$$

responsible for V^C, with no assumptions and a single region $c \geq 1$.

$$\begin{aligned} H \doteq \langle V, &(R_0^H \wedge A_0^H) \vee (R_1^H \wedge A_1^H), \\ &(R_0^H \wedge A_0^H \wedge T_{0,0}^H \wedge R_0^{H\prime} \wedge A_0^{H\prime}) \vee \\ &(R_0^H \wedge A_0^H \wedge T_{0,1}^H \wedge R_1^{H\prime} \wedge A_1^{H\prime}) \vee \\ &(R_1^H \wedge A_1^H \wedge (T_{1,0,0}^H \vee T_{1,0,1}^H) \wedge R_0^{H\prime} \wedge A_0^{H\prime}) \rangle, \end{aligned}$$

responsible for V^H and such that: (i) $R_0^H \equiv R_1^H \doteq h = 0$, (ii) $A_0^H \doteq \delta = 0$, (iii) $A_1^H \doteq \delta = \frac{2v}{g}$, (iv) $T_{0,0}^H \equiv T_{0,1}^H \equiv T_{1,0,0}^H \doteq h' = h$ and $T_{1,0,1}^H \doteq h' = h + v\delta - \frac{g}{2}\delta^2$. Finally, we define

$$\begin{aligned} DV \doteq \langle V, &(R_0^{DV} \wedge A_0^{DV}) \vee (R_1^{DV} \wedge A_1^{DV}), \\ &(R_0^{DV} \wedge A_0^{DV} \wedge T_{0,1}^{DV} \wedge R_1^{DV\prime} \wedge A_1^{DV\prime}) \vee \\ &(R_1^{DV} \wedge A_1^{DV} \wedge T_{1,0}^{DV} \wedge R_0^{DV\prime} \wedge A_0^{DV\prime}) \rangle, \end{aligned}$$

responsible for V^{DV}, where $R_0^{DV} \doteq \delta = 0 \wedge v = -\frac{g}{2c}$ and $R_1^{DV} \doteq \delta = \frac{1}{c} \wedge v = \frac{g}{2c}$, the two assumptions are $A_0^{DV} \equiv A_1^{DV} = c \geq 1 \wedge h = 0$ and the two components of the transition relation are defined as $T_{0,1}^{DV} \doteq \delta' = \frac{1}{c+1} \wedge v' = -v\frac{c}{c+1}$ and $T_{1,0}^{DV} \doteq \delta' = 0 \wedge v' = v - g\delta$.

The three E-comps satisfy all hypotheses required by Def. 20. Applying the composition operator and removing empty regions and transitions we obtain

$$B \doteq C \otimes DV \otimes H = \langle V, R_0^B \vee R_1^B, (R_0^B \wedge T_{0,1}^B \wedge R_1^{B\prime}) \vee (R_1^B \wedge T_{1,0}^B \wedge R_0^{B\prime}) \rangle,$$

with two regions $\{R_0^B, R_1^B\}$ and no assumptions, where:

$$\begin{aligned} R_0^B &\doteq c \geq 1 \wedge \delta = \frac{1}{c} \wedge v = \frac{g}{2c} \wedge h = 0; \\ R_1^B &\doteq c \geq 1 \wedge \delta = 0 \wedge v = -\frac{g}{2c} \wedge h = 0; \\ T_{0,1}^B &\doteq c' = c \wedge \delta' = 0 \wedge v' = v - g\delta \wedge h' = h; \\ T_{1,0}^B &\doteq c' = c + 1 \wedge \delta' = \frac{1}{c+1} \wedge v' = -v\frac{c}{c+1} \wedge h' = h. \end{aligned}$$

Region R_0^B implies the fairness condition $h = 0 \wedge v > 0$ and we obtain the E-comp $\langle V, \{R_0^B, R_1^B\}, T^B \rangle$, where $T^B \doteq \bigvee_{i \in \{0,1\}} (R_i^B \wedge T_{i,1-i}^B \wedge R_{1-i}^{B\,\prime})$ which is exactly the definition of H shown in Fig. 6.8.

Running example

We now show how the E-comps we defined in §6.3.2 can be combined to conclude the existence of a fair path in the fair transition system Ex defined in Sec. 6.1.

We first compute the E-comp $H^{f_0,f_1} \doteq H^{f_0} \otimes H^{f_1}$, depicted in Fig. 6.9. H^{f_0} and H^{f_1} have no assumptions and all ranking functions are always equal to their minimal element. Therefore, all transitions are compatible and the result of the composition is the synchronous product of the two E-comps. H^{f_0,f_1} has four regions, one for each of the possible truth assignments of the two Boolean symbols f_0 and f_1 and it allows all 16 possible transitions and self-loops over them.

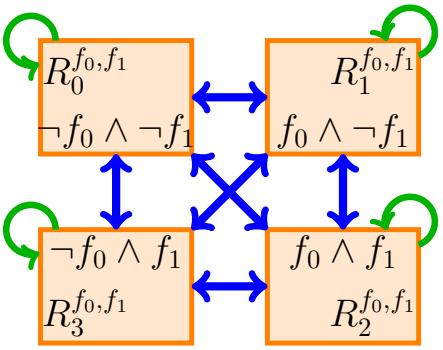

Figure 6.9: E-comp responsible for $\{f_0, f_1\}$.

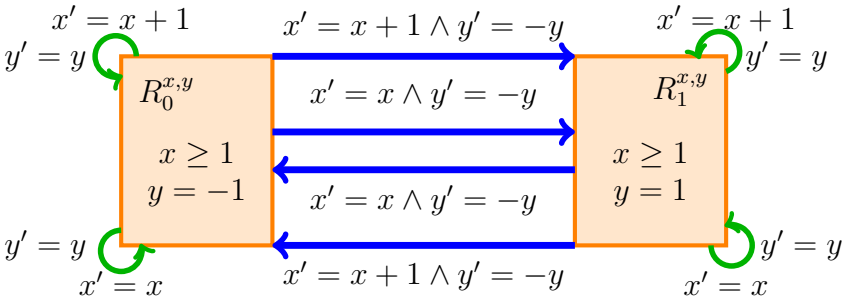

Figure 6.10: E-comp responsible for $\{x, y\}$.

We now compute $H^{x,y} \doteq H^x \otimes H^y$, depicted in Fig. 6.10. The assumption of H^x requires $y \leq 1$ and both assumptions of H^y require $x \geq 1$. Therefore,

H^x will always meet the assumptions of H^y and vice-versa also H^y meets the assumption of H^x. The two E-comps do not have any assumptions on the other symbols and the resulting E-comp $H^{x,y}$ has no assumptions. $H^{x,y}$ has two regions, obtained by the conjunction of the two regions of H^y with the only region of H^x and its transition relation is the conjunction of the transition relations of H^x and H^y. Both regions of $H^{x,y}$ admit stutter transitions of two kinds: one in which both variables x and y remain constant, and one in which y is constant and x increases by one. Finally, $H^{x,y}$ also admits progress transitions from one region to the other of two kinds: in both cases the value of y changes its sign, while in one case x remains constant and in the other it increments by one.

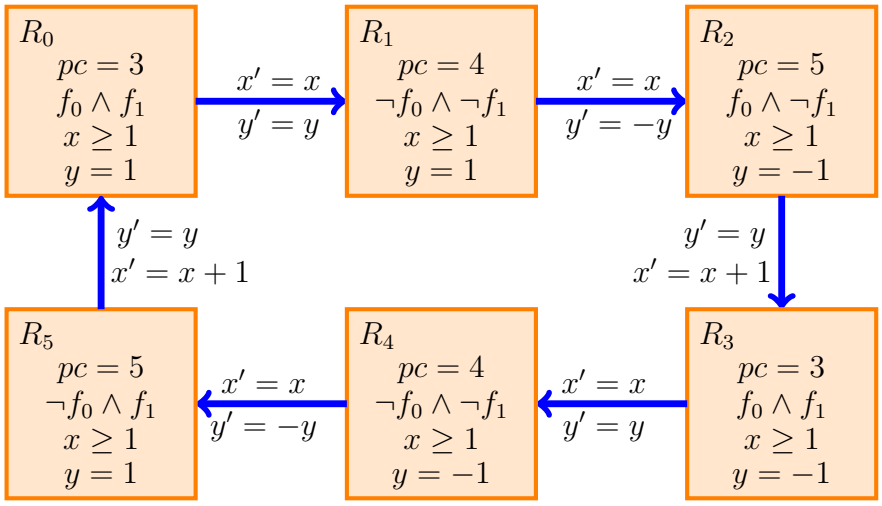

Figure 6.11: E-comp responsible for all symbols $\{pc, f_0, f_1, x, y\}$.

Finally, we compute $H \doteq H^{pc} \otimes H^{x,y} \otimes H^{f_0,f_1}$. None of the E-comps has assumptions and all their ranking functions are always equal to the minimal element. For this reason, all transitions are compatible and have independent ranks. Therefore, the transition relation of H is the conjunction of the transition relations of the three E-comps. H has 24 regions, given by the product of the 3 regions of H^{pc}, 2 of $H^{x,y}$ and 4 of H_{f_0,f_1}. The regions represent all the configurations that can be reached by employing

compatible transitions of our E-comps H^{pc}, H^{f_0}, H^{f_1}, H^x and H^y. Recall that our objective is to identify fair paths for the fair transition system Ex defined in Sec. 6.1. Not all transitions of H are also transition of Ex. For example, H admits a transition that increases the value of x from states where $pc = 3$, while this is not possible in Ex. However, using the projection operator we can restrict H by considering a subset of its regions. In particular, we are interested in the sequence of regions that would allow us to obtain a representation of at least one fair path for Ex. We select six regions and depict the projection $H^{\downarrow}$ of H over such regions in Fig. 6.11. In particular, we consider the following regions:

$$
\begin{aligned}
R_0 &\doteq \quad f_0 \wedge \quad f_1 \wedge y = \quad 1 \wedge x \geq 1 \wedge pc = 3;\\
R_1 &\doteq \neg f_0 \wedge \neg f_1 \wedge y = \quad 1 \wedge x \geq 1 \wedge pc = 4;\\
R_2 &\doteq \quad f_0 \wedge \neg f_1 \wedge y = -1 \wedge x \geq 1 \wedge pc = 5;\\
R_3 &\doteq \quad f_0 \wedge \quad f_1 \wedge y = -1 \wedge x \geq 1 \wedge pc = 3;\\
R_4 &\doteq \neg f_0 \wedge \neg f_1 \wedge y = -1 \wedge x \geq 1 \wedge pc = 4;\\
R_5 &\doteq \neg f_0 \wedge \quad f_1 \wedge y = \quad 1 \wedge x \geq 1 \wedge pc = 5.
\end{aligned}
$$

Notice that the 6 regions underapproximate those that we have already considered in the funnel-loop described in §6.2.1. In particular, for every $i \in \{0, \ldots, 5\}$ R_i underapproximates S_i. Moreover, $H^{\downarrow}$ satisfies all hypotheses of Th. 12, hence by Th. 6, it proves the existence of a fair path in the language of the fair transition system Ex and we reached our goal.

Chapter 7

Search Procedure

This chapter describes two different approaches that given a fair transition system try to synthesise a corresponding funnel-loop.

Sec. 7.1 presents a sound and complete reduction of the synthesis problem into E-CHC (defined in §2.4.1). Therefore, the completeness of such encoding and the language used to represent the funnel depend on the ones of the underlying procedure used to solve the E-CHC problem. Unfortunately, we were not able to obtain any such tool and could not evaluate the approach. However, we believe that the encoding provides a formalisation of the synthesis problem in a solver-independent language that could ease the understanding of the problem at hand and also support reasoning.

Sec. 7.2 provides a high-level description of an ad-hoc procedure. The procedure performs the search via a sequence of SMT-queries and adopts a template-based approach to identify the funnel-loop.

7.1 Encoding of Funnel-loop Search in E-CHC

We now present an encoding of the search for a funnel-loop for a fair transition system in E-CHCs. E-CHCs allow us to clearly define the search problem using an established formalism and notation.

We present a sound and complete encoding for the search problem of a

funnel-loop of length one in E-CHCs. While it is possible to represent the search of a funnel-loop of arbitrary length n, Th. 7 ensures that looking for funnel-loops of length one is sufficient. In addition, it is possible to define a similar E-CHC encoding that also considers a set of user-defined E-comps as hints, similarly to the procedure we describe in Sec. 7.2. However, such encoding is rather complex and does not provide any additional contribution to our discussion since we were not able to obtain any tool capable of identifying solutions for E-CHCs.

Let $M \doteq \langle V, I^M, T^M, F^M \rangle$ be a fair transition system and let $R(c, V)$, $T(V, V')$ and $Rank(V, V')$ be query symbols, where c is a fresh Boolean symbol ($c \notin V$). A solution to the E-CHC problem below is an intepretation for R, T and $Rank$ satisfying all its formulae. R represents the source region, and $c \wedge R(c, V)$ underapproximates the fair states. Th. 15 shows that any solution to the E-CHC below corresponds to a funnel-loop and Th. 16 shows that if M admits a funnel-loop then there exists an interpretation for the query symbols satisfying the following E-CHC. Therefore, the encoding is sound (Th. 15) and relatively complete (Th. 16).

$$\top \;\rightarrow\; \exists c, V : R(c, V) \wedge I^M(V) \tag{7.1}$$
$$T(V, V') \;\rightarrow\; \exists c : R(c, V') \tag{7.2}$$
$$R(c, V) \wedge T(V, V') \;\rightarrow\; T^M(V, V') \tag{7.3}$$
$$R(c, V) \;\rightarrow\; \exists V' : T(V, V') \tag{7.4}$$
$$c \wedge R(c, V) \;\rightarrow\; F^M(V) \tag{7.5}$$
$$\neg c \wedge R(c, V) \wedge T(V, V') \;\rightarrow\; Rank(V, V') \tag{7.6}$$
$$wf(Rank) \tag{7.7}$$

Let $Cex \doteq \langle V, \exists c : R(c, V), T(V, V'), \top \rangle$ be the transition system associated with an interpretation of the query symbols of the E-CHC above. Eq. (7.1) requires the existence of some initial state of M in R, hence that

the set of initial states of *Cex* is not empty. Every path of *Cex* is also a path in M or, in other words, *Cex* is simulated by M. In fact, Eq. (7.2) ensures that T can only reach states in R, and Eq. (7.3) guarantees that in such region T is an underapproximation of T^M. We require all paths of *Cex* to be infinite (i.e. *Cex* never reaches a deadlock). This is guaranteed by Eq. (7.4) that requires T to be left-total with respect to R. Finally, all such paths must be fair. Eq. (7.5) guarantees that $R(\bot, V)$ is a subset of the fair states of M. Eq. (7.6) requires the relation $T(V, V')$ describing pairs of current and next states such that the first one is in $R(\bot, V)$ to underapproximate some well-founded relation *Rank*. The well-foundedness of *Rank* ensures that there is no infinite chain of states in $R(\bot, V)$, hence *Cex* must eventually reach a state in $R(\top, V)$ and, by Eq. (7.5), it must eventually reach a fair state.

Theorem 15 - E-CHC Encoding is Sound
Given a fair transition system $M \doteq \langle V, I^M, T^M, F^M \rangle$*; if there exists an interpretation for the queries* R*,* T *and Rank that satisfies all Eqs.* (7.1)–(7.7)*, then there exists a funnel-loop for* M*.*

Proof. We first show that $R(c, V)$ and $T(V, V')$ describe a funnel and then show that such funnel corresponds to a funnel-loop of length one. We define a funnel $fnl \doteq \langle V, S(V), T_{fnl}(V, V'), D(V), \mathrm{RF}(V) \rangle$, where (i) $S(V) \doteq \exists c : R(c, V)$, (ii) $T_{fnl}(V, V') \doteq T(V, V') \wedge (D(V') \leftrightarrow \mathrm{RF}(V) = \mathbf{0})$, (iii) $D(V) \doteq \exists c : c \wedge R(c, V)$ and (iv) $\mathrm{RF}(V)$ is a ranking function witnessing the well-foundedness of relation $Rank(V, V')$ such that $\mathrm{RF}(V) = 0$ for all V for which there exists no V' making $\neg c \wedge R(c, V) \wedge T(V, V') \wedge \wedge c' R(c', V')$ true.

$$\forall c, V, c' : \neg(\exists V' : \neg c \wedge R(c, V) \wedge T(V, V') \wedge c' \wedge R(c', V')) \rightarrow \mathrm{RF}_i(V) = 0.$$

In all other cases (i.e. when $R(c, V) \wedge T(V, V') \wedge R(c, V')$ holds for all V, V') the following must hold:

$$\forall V, V' : (\neg c \wedge R(c, V) \wedge T(V, V') \wedge R(c, V')) \rightarrow \mathrm{RF}(V) \geq \mathrm{RF}(V') + 1.$$

These two constraints allow for many different interpretations of RF. Every such interpretation satisfies our requirements and it is sufficient for such set to be nonempty. The well-foundedness of *Rank* implies, by Eq. (7.6), that $\neg c \wedge R(c, V) \wedge T(V, V') \wedge R(c, V')$ is well-founded. Therefore, there must exist some V such that $\text{RF}(V) = 0$. The formula holds for all the states in $\neg c \wedge \neg R(c, V)$ and all the states in $\neg c \wedge R(c, V)$ for which T does not admit any successor in the same region. Since $\neg c \wedge R(c, V) \wedge T(V, V') \wedge R(c, V')$ is well-founded it cannot allow for any infinite chain of states, hence it cannot allow any loop of states. Therefore, the constraints above do not contain any circular dependency in the definition of the assignments to the $\text{RF}(V)$ and there exists at least one interpretation for RF.

We now show that *fnl* satisfies all hypotheses required by Def. 18.

- Hyp. *F.1* follows directly from Eq. (7.4) and the fact that $\text{RF} = \mathbf{0}$ implies that T does not admit any successor in $\neg c \wedge R(c, V)$, hence it must admit some successor in $c \wedge R(c, V)$, which by definition is in D.

- Consider now Hyp. *F.2*.
  By construction S contains all states of $\exists c : R(c, V)$. Eq. (7.2) ensures that this is an invariant, hence Hyp. *F.2* holds.

- Consider Hyp. *F.3*.
  By construction, RF assigns decreasing integers to the chains described by the relation $\neg c \wedge R(c, V) \wedge T(V, V') \wedge R(c, V')$. Therefore, at every such step RF must decrease and Hyp. *F.3* holds.

- Finally, consider Hyp. *F.4*.
  Eq. (7.2) and the well-foundedness of $\neg c \wedge R(c, V) \wedge T(V, V')$, ensures that from a state in $\neg c \wedge R(c, V)$, in a finite number of T steps, we must reach a state in $c \wedge R(c, V)$. We defined RF such that $\text{RF} = \mathbf{0}$ in the states whose T successors are in $c \wedge R(c, V)$, hence in D. Therefore, Hyp. *F.4* holds.

We now show that *fnl* is a funnel-loop, i.e. it meets Hyp. *FL.1* required by Def. 19. *fnl* is the only funnel, hence we simply need to show that the destination region D underapproximates the source region S. We defined S as the union of $c \wedge R(c, V)$ and $\neg c \wedge R(c, v)$ and D as $c \wedge R(c, V)$. Therefore $D \rightarrow S$ and Hyp. *FL.1* holds.

Finally, we show that this funnel-loop represents at least one fair path of M by showing that it meets all hypotheses of Th. 6.

- Hyp. *FF.1* holds since Eq. (7.1) ensures that $R(c, V)$ has a nonempty intersection with the initial states I^M.
- Hyp. *FF.2* holds since Eq. 7.5 ensures that every state in $c \wedge R(c, V)$ satisfies $F^M(V)$. We defined $D \doteq R(\top, V)$, hence $D \rightarrow F^M$ and Hyp. *FF.2* must hold.
- Finally, Hyp. *FF.3* follows directly from Eq. (7.3).

□

Theorem 16 - E-CHC Encoding is Relatively Complete
Let floop be a funnel-loop of length one for a fair transition system $M \doteq \langle V, I^M, T^M, F^M \rangle$. *Then, there exists an intepretation for the query symbols* R, T *and Rank satisfying all Eqs.* (7.1)–(7.7).

Proof. Given a *floop* of length one, we define an interpretation for the query symbols R, T and *Rank* for the E-CHC. Let $fnl \doteq \langle V, S, T_{fnl}, D, \text{RF} \rangle$ be the only funnel in *floop*. Th. 6 ensures the existence of a finite sequence of states σ such that: (i) it starts from an initial state of M, (ii) follows the transition relation of M and (iii) ends in a state in the source region S. Without loss of generality we assume that σ does not contain any state in S other than the last one. In the following we write $\sigma(V)$ for the predicate that holds iff V is in σ and $\sigma(V, V')$ for the predicate that holds iff V and V' are two consecutive states in σ. Define the interpretation of the queries as follows: (i) $R(c, V) \doteq (\sigma(V) \vee S(V)) \wedge (c \leftrightarrow D(V))$, (ii) $T(V, V') \doteq \sigma(V, V') \vee$

$(S(V) \wedge T_{fnl}(V,V'))$ and (iii) $Rank(V,V') \doteq \sigma(V,V') \vee \text{RF}(V') < \text{RF}(V)$. We show that the interpretation satisfies Eqs. (7.1)–(7.7).

- Consider first Eq. (7.1).
  By construction $\neg c \wedge R(c,V)$ contains all states in σ. By hypothesis, the first state of σ is an initial state of M. Therefore, Eq. (7.1) holds.

- We now show that Eq. (7.2) holds.
  $R(c,V)$ contains all states of σ and of S. T either follows the transitions of σ or, once it reaches S, it follows the transition relation of *fnl*. By hypotheses *F.2*, *F.4* and *FL.1* such transitions must remain in S. Therefore, from every state not in S and not in σ, T is false and the left-hand-side of Eq. (7.2) is false; otherwise, every T transition must remain within $R(c,V)$ and Eq. (7.2) is true.

- Consider now Eq. (7.3).
  Every step in σ is also a step in M and by Hyp. *FF.3* every step of *floop* underapproximates the transition relation of M. Therefore, $T(V,V')$ underapproximates T^M and Eq. (7.3) holds.

- Consider Eq. (7.4).
  Hyp. *F.1* must hold for *fnl* and every state in σ must admit a successor until a state in S is reached. Thus, by construction, $T(V,V')$ always allows from some successor state in region $R(V,c)$ and Eq. (7.4) holds.

- We now prove that Eq. (7.5) holds.
  By Hyp. *FF.2*, D underapproximates the fair states and, by construction, $c \wedge R(c,V)$ is equivalent to such region. Therefore, Eq. 7.5 holds.

- Eq. (7.6) holds by construction of the interpretation for *Rank*.

- Finally, Eq. (7.7) holds since σ is a finite sequence of states and RF is a ranking function with respect to T_{fnl} and S.

□

7.2 Direct Procedure

In this section we propose a fully-automated procedure that, given a fair transition system and a possibly empty set of E-comps, searches for a funnel-loop containing at least one and only fair paths. The main procedure is described by Alg. 1. Alg. 1 searches for a funnel-loop by enumerating candidate fair loops of the transition system. Alg. 2 details how these candidates can be generated. It considers finite paths such that their first and last states are in the same abstract state with respect to a set of abstraction predicates. Given a candidate loop, Alg. 1 proceeds by computing a sequence of regions and transitions containing it, via Alg. 3. The procedure then searches for a funnel-loop corresponding to a strengthening of the sequence of regions and transitions such that all required hypotheses are met (Sec. 7.2.4). If this succeeds, then the procedure returns the obtained funnel-loop, otherwise it continues by analysing the next candidate fair loop.

The procedure, is fully-automated and looks for a funnel-loop representable using quantifier-free FOL formulae over the theory of linear and non-linear mixed integer real arithmetic. This problem is undecidable, hence there will always exist some inputs for which it fails to provide an answer and, from a more practical perspective, inputs for which it takes a very long time to provide an answer. For this reason, the procedure is capable of exploiting some additional information in the form of a set of E-comps. If some E-comps are provided, the procedure identifies candidate fair loops that are also a path for some composition and projection of a subset of the E-comps. It then tries to identify the funnel-loop that corresponds to an E-comp describing the behaviour for the missing symbols. It completes the E-comp with a transition relation over the remaining symbols such that all assumptions are met. This feature allows the procedure

return a witness by exploiting the human ingenuity to define some useful E-comps. This is particularly relevant in cases where the fully-automated version without hints and its heuristics fail.

Given a fair transition system M and a set of E-comps $\mathcal{H}$, the procedure tries to find, in a fully automated manner, a funnel-loop *fnl_loop* for M and a finite path of M ending in a region of *fnl_loop*. $\mathcal{H}$ is a possibly empty set of E-comps provided as input to guide the search, for this reason we will refer to them as *hints*. The procedure selects a possibly empty subset of hints and uses them as building blocks to define the funnel-loop while synthesising the missing components. When the set of hints is empty the procedure identifies a funnel-loop for a fair transition system without relying on any additional information. In the following, we call trivial hint the E-comp $H \doteq \langle V, \top, \top \rangle$ responsible for no symbols ($V^H \doteq \emptyset$) such that all its regions and assumptions are the constant $\top$ and all its ranking functions are always equal to $\mathbf{0}$. Alg. 1 describes the main steps

Algorithm 1 SEARCH-FUNNEL-LOOP(M, $\mathcal{H}$)

```
    ▷ Iterate over candidate loops of increasing length.
 1: for all ⟨prefix, loop_r, loop_t, H⟩ ∈ GENERATE-CANDIDATE-LOOPS(M, H) do
 2:     v_0 ← prefix[len(prefix) − 1]              ▷ Witness for reachability, Hyp. FF.1.
        ▷ Iterate over funnel-loop templates for current candidate loop.
 3:     for all template ∈ GENERATE-TEMPLATES(v_0, loop_r, loop_t, H) do
 4:         ef_constrs ← template.ef_constraints()                ▷ Get ∃∀ problem.
 5:         ⟨found, model⟩ ← SEACH-PARAMETER-ASSIGNMENT(ef_constrs)
 6:         if found == ⊤ then                ▷ Replace parameters with assignment.
 7:             fnl_loop ← template.instantiate(model)
 8:             return ⟨prefix, fnl_loop⟩          ▷ Reachability witness and funnel-loop.
 9:         end if
10:     end for
11: end for
12: return unknown
```

of the procedure. We reduce the synthesis problem to a sequence of SMT

queries. In order to reduce the search space, given an E-comp H we only look for funnel-loops obtained by deterministic completions of H. For this reason, we strengthen the transition relation of H by adding deterministic assignments to the symbols for which H is not responsible. More in detail, Alg. 1 enumerates candidate conjunctive fair loops of the fair transition system and compositions of E-comps that admit such loop (line 1). If GENERATE-CANDIDATE-LOOPS selects no hints or $\mathcal{H}$ is empty the returned H is the trivial hint. For each candidate loop, the procedure generates a sequence of parameterised funnel-loops, called funnel-loop templates, as a strengthening of the corresponding E-comp (line 3). This is achieved via the function GENERATE-TEMPLATES detailed in Alg 3. The predicates of a funnel-loop template are over the symbols of the system M and a set of parameters P. P is a set of fresh integer or real variables and the procedure searches for an assignment (or interpretation) to the parameters such that all the hypotheses of Defs. 18 and 19 and of Th. 6 hold. At line 4 the procedure obtains the $\exists\forall$-quantified problem associated with the funnel-loop template and then, at line 5 tries to solve it. Finally, at line 7, it replaces the parameters with the assignment identified at the previous step, obtaining the desired funnel-loop.

The procedure relies on ranking functions to perform two different tasks. Alg. 2 tries to synthesise ranking functions to avoid considering candidate loops for which we know a ranking function exists. The existence of the ranking function proves that the loop must eventually terminate, hence it cannot correspond to an infinite path. Then, ranking function templates are also used as components for the funnels of the funnel-loop template generated by Alg. 3.

Before providing further details about the procedure, we illustrate it in §7.2.1 by showing its application to the fair transition system Ex defined in Sec. 6.1. Then, we describe how we represent and enumerate candidate

loops and compositions of E-comps for the transition system M in §7.2.2. After that, in §7.2.3, we detail how a funnel-loop template is generated from a candidate loop and E-comp. Finally, §7.2.4 reports the synthesis problem associated with a funnel-loop template.

7.2.1 Example Funnel-loop Search

We first recall the definition of the fair transition system Ex, introduced in Sec. 6.1. Let $V \doteq \{x, y, pc, f_0, f_1\}$ be a set of symbols such that pc and x are integer variables, y has real type and f_0 and f_1 are two Boolean symbols. Then, the fair transition system is $Ex \doteq \langle V, I, T, F \rangle$, where:

$$
\begin{aligned}
I &\doteq pc = 3;\\
F &\doteq f_0 \wedge f_1;\\
T &\doteq (pc = 3 \rightarrow (x^2 \geq xy \wedge pc' = 4 \wedge x' = x \wedge y' = y)) \wedge\\
&\quad\ (pc = 4 \rightarrow (pc' = 5 \wedge x' = x)) \wedge\\
&\quad\ (pc = 5 \rightarrow (pc' = 3 \wedge x' = x + 1 \wedge y' = y)) \wedge\\
&\quad\ ((f_0 \wedge f_1) \rightarrow (\neg f_0' \wedge \neg f_1')) \wedge\\
&\quad\ (f_0' \rightarrow (f_0 \vee y > 0)) \ \wedge\ (f_1' \rightarrow (f_1 \vee y < 0)).
\end{aligned}
$$

In addition, we assume no hints were provided, i.e. $\mathcal{H} \doteq \emptyset$. Let Ex and $\mathcal{H}$ be the inputs of our procedure. Alg. 1, at line 1, iterates over the candidate loops generated from Ex and $\mathcal{H}$. Each candidate loop is described by a 4-tuple $\langle \boldsymbol{v}_0, loop\_r, loop\_t, H \rangle$. $loop\_r$ and $loop\_t$ are sequences of predicates over V and $V \cup V'$ respectively. The two sequences, together with H, describe the abstract loop. Instead, $\boldsymbol{v}_0$ is a state in the first region of $loop\_r$ reachable in Ex. Therefore, it is the last state of a finite path *prefix* of Ex that starts its initial states and ends in $\boldsymbol{v_0}$. We compute $\langle \boldsymbol{v}_0, loop\_r, loop\_t, H \rangle$ by employing a liveness-to-safety [29] transformation of Ex where the loop-back is identified in an abstract state. We then em-

ploy an unrolling of the transition relation in the style of Bounded Model Checking (BMC) [31] to enumerate concrete paths of Ex with such abstract loop-back. The stem of this concrete path corresponds to our *prefix*. *loop_r* and *loop_t* are obtained from the loop of the concrete path by computing an implicant for the unrolling of the transition relation of Ex. We then partition the predicates in the implicant depending on their index in the unrolling and whether they contain only current (*loop_r*) or both current and next-state variables (*loop_t*). Assume we are considering a BMC unrolling of 6 transitions of Ex and obtain the following path:

$$\begin{aligned}
0: &\quad f_0 \wedge \quad f_1 \wedge pc = 3 \wedge x = 1 \wedge y = \quad 1;\\
1: &\neg f_0 \wedge \neg f_1 \wedge pc = 4 \wedge x = 1 \wedge y = \quad 1;\\
2: &\quad f_0 \wedge \neg f_1 \wedge pc = 5 \wedge x = 1 \wedge y = -1;\\
3: &\quad f_0 \wedge \quad f_1 \wedge pc = 3 \wedge x = 2 \wedge y = -1;\\
4: &\neg f_0 \wedge \neg f_1 \wedge pc = 4 \wedge x = 2 \wedge y = -1;\\
5: &\neg f_0 \wedge \quad f_1 \wedge pc = 5 \wedge x = 2 \wedge y = \quad 2;\\
6: &\quad f_0 \wedge \quad f_1 \wedge pc = 3 \wedge x = 3 \wedge y = \quad 2;
\end{aligned}$$

where the states with indexes 0 and 6 correspond to the same state in the abstract space defined by the predicates appearing in Ex. We use this path to compute an implicant for the formula $F(V_0) \wedge \bigwedge_{i=0}^{5} T(V_i, V_{i+1})$. The implicant is a conjunction of a subset of the atoms appearing in the formula such that it implies the formula itself. In addition, the path is a satisfying assignment also for the implicant. Each predicate in the unrolling depends either on a single V_i or on $V_i \cup V_{i+1}$ for some i, hence the same holds for the predicates in the implicant. We partition the atoms of the implicant such that the predicates that depend only on V_i are in *loop_r*$[i\%6]$ and those that depend on $V_i \cup V_{i+1}$ are placed in *loop_t*$[i]$. The first and last state correspond to the same abstract region, hence their predicates are placed together into *loop_r*$[0]$.

The computation above allows us to obtain the following. Since $\mathcal{H}$ is empty H is the trivial hint. The *prefix* contains a single state: $prefix \doteq [f_0 \wedge f_1 \wedge pc = 3 \wedge x = 1 \wedge y = 1]$, $loop\_r$ and $loop\_t$ have length 6 and each $loop\_r[i] \wedge loop\_t[i]$ underapproximates the transition relation T.

	$loop\_r \doteq [$	$loop\_t \doteq [$
0 :	$f_0 \wedge f_1 \wedge pc = 3 \wedge x^2 \geq xy,$	$\neg f_0' \wedge \neg f_1' \wedge pc' = 4 \wedge x' = x \wedge y' = y,$
1 :	$\neg f_0 \wedge \neg f_1 \wedge pc = 4 \wedge y > 0,$	$f_0' \wedge \neg f_1' \wedge pc' = 5 \wedge x' = x,$
2 :	$f_0 \wedge \neg f_1 \wedge pc = 5 \wedge y < 0,$	$f_0' \wedge f_1' \wedge pc' = 3 \wedge x' = x + 1 \wedge y' = y,$
3 :	$f_0 \wedge f_1 \wedge pc = 3 \wedge x^2 \geq xy,$	$\neg f_0' \wedge \neg f_1' \wedge pc' = 4 \wedge x' = x \wedge y' = y,$
4 :	$\neg f_0 \wedge \neg f_1 \wedge pc = 4 \wedge y < 0,$	$\neg f_0' \wedge f_1' \wedge pc' = 5 \wedge x' = x,$
5 :	$\neg f_0 \wedge f_1 \wedge pc = 5 \wedge y > 0];$	$f_0' \wedge f_1' \wedge pc' = 3 \wedge x' = x + 1 \wedge y' = y].$

We now search for a funnel-loop as a strengthening of this candidate loop. Notice that the candidate loop by itself is not sufficient. In fact, $loop\_t[1]$ does not constrain the next assignment of y', hence it does not guarantee that $y < 0$ holds in the next state as required by $loop\_r[2]$. Before building the funnel-loop template, we can perform some simplifications on the candidate loop to reduce the number of parameters introduced by the template and ease the presentation. First of all, notice that every step i in $loop\_t$ assigns to the variables f_0, f_1 and pc a constant value that corresponds to the one required by $loop\_r[i +_6 1]$. Therefore, for brevity, we will omit such constraints from the formulae in $loop\_t$ and focus our presentation on x and y. Moreover, many steps in $loop\_t$ require x or y to remain constant. Consider a step $t \doteq loop\_t[i]$ that requires y to be constant. We need t to map states in $r_s \doteq loop\_r[i]$ into $r_d \doteq loop\_r[i +_6 1]$. Therefore, if r_d requires y to be positive ($y > 0$), then the same must hold in r_s and vice-versa. We can exploit identity relations in $loop\_t$ to symbolically propagate constraints in

$loop\_r$. By employing these transformations we obtain the following:

$$
\begin{array}{lll}
 & loop\_r \doteq [& loop\_t \doteq [\\
0: & \quad f_0 \wedge \quad f_1 \wedge pc = 3 \wedge y > 0 \wedge x^2 \geq xy, & \quad x' = x \wedge y' = y, \\
1: & \neg f_0 \wedge \neg f_1 \wedge pc = 4 \wedge y > 0, & \quad x' = x, \\
2: & \quad f_0 \wedge \neg f_1 \wedge pc = 5 \wedge y < 0, & \quad x' = x + 1 \wedge y' = y, \\
3: & \quad f_0 \wedge \quad f_1 \wedge pc = 3 \wedge y < 0 \wedge x^2 \geq xy, & \quad x' = x \wedge y' = y, \\
4: & \neg f_0 \wedge \neg f_1 \wedge pc = 4 \wedge y < 0, & \quad x' = x, \\
5: & \neg f_0 \wedge \quad f_1 \wedge pc = 5 \wedge y > 0]; & \quad x' = x + 1 \wedge y' = y].
\end{array}
$$

We now define a funnel-loop template of length 6 that can be generated by Alg. 1 at line 3. For $i \in \{0, \ldots, 5\}$, we define the i^{th} funnel of the template as a strengthening of $loop\_r[i]$ and $loop\_t[i]$. In the template we use symbols from the set $P \doteq \{p_i | i \in \mathbb{N}\}$, disjoint from V, as parameters. The parameters are variables for which we need to find an assignment such that the template corresponds to an actual funnel-loop. Notice that the steps in $loop\_t$ already prescribe functional assignments for all variables but for y at steps 1 and 4. For this reason, we introduce 2 parametric affine expressions to underapproximate the assignment to y'. In addition, we introduce parametric affine inequalities over x and y to strengthen the elements of $loop\_r$. Also in this case we reduce the number of parameters we need to introduce by exploiting the functional assignments of $loop\_t$. For $i \in \{0, 1 \ldots, 5\}$, let $fnl_i \doteq \langle V, src_i, t_i, \mathbf{0}, dst_i \rangle$ be the i^{th} funnel of the template. We define each destination region dst_i as the set of states reachable from the previous source region when the ranking function is equal to the minimal element. Since we defined every ranking function to be always equal to the minimal element, we define each destination region as:

$$dst_i \doteq \exists V : src_i(V, P) \wedge t_i(V, V', P).$$

We define the source regions and transition relations as follows.

$$
\begin{aligned}
src_0 &\doteq loop\_r[0] \wedge p_6x + p_7y + p_8 \geq 0;\\
t_0 &\doteq loop\_t[0];\\
src_1 &\doteq loop\_r[1] \wedge p_6x + p_7y + p_8 \geq 0;\\
t_1 &\doteq loop\_t[1] \wedge y' = p_0x + p_1y + p_2;\\
src_2 &\doteq loop\_r[2] \wedge p_9x + p_{10}y + p_{11} + p_9 \geq 0;\\
t_2 &\doteq loop\_t[2];\\
src_3 &\doteq loop\_r[3] \wedge p_9x + p_{10}y + p_{11} \geq 0;\\
t_3 &\doteq loop\_t[3];\\
src_4 &\doteq loop\_r[4] \wedge p_9x + p_{10}y + p_{11} \geq 0;\\
t_4 &\doteq loop\_t[4] \wedge y' = p_3x + p_4y + p_5;\\
src_5 &\doteq loop\_r[5] \wedge p_6x + p_7y + p_8 + p_6 \geq 0;\\
t_5 &\doteq loop\_t[5].
\end{aligned}
$$

We introduced two parametric inequalities: $p_6x + p_7y + p_8 \geq 0$ at index 1 and $p_9x + p_{10}y + p_{11} \geq 0$ at index 4. Then, we propagated the inequalities backward exploiting the assignments to x and y of $loop\_t$. In particular, in $loop\_t[0]$ and $loop\_t[3]$ both x and y must remain constant. In $loop\_t[2]$ and $loop\_t[5]$, instead, y remains constant and x increases by 1. Therefore, $p_9x+p_{10}y+p_{11} \geq 0$ in src_3 implies that $p_9x+p_{10}y+p_{11}+p_9 \geq 0$ must hold in src_2 and similarly $p_6x+p_7y+p_8 \geq 0$ in src_0 implies $p_6x+p_7y+p_8+p_6 \geq 0$ at src_5. We remark that exploiting the equalities in the transition relations is an optimisation we employ to reduce the number of parameters and has no effect on the correctness of the approach.

Now, we need to identify an assignment to the parameters $p_0, \ldots, p_{11}$ such that the funnel-loop template satisfies all hypotheses of Def. 18, Def. 19 and Th. 6. The procedure generates this synthesis problem at line 4 and it searches for a solution (assignment to the parameters) at

line 5. The synthesis problem requires the funnel-loop to be reachable in *Ex* (*FF.1*), hence also not empty. We ensure this by requiring the first region of the funnel-loop to contain the last state of the prefix, hence the state $f_0, f_1, pc = 3, x = 1, y = 1$ must be in src_0. Then, the funnel-loop must never encounter a deadlock (*F.1*). This holds by construction of the transition relations of the funnels; in fact, every t_i is left-total for every assignment to the parameters. We need the funnels to be correctly chained (*FL.1*) and to underapproximate the transition relation T of *Ex* (*FF.3*). We defined the destination regions as the set of states reachable from the source region in one step. Therefore, we require the following to hold:

$$\exists P \forall V : src_i(V, P) \wedge t_i(V, V', P) \rightarrow src_{i+_6 1}(V', P) \wedge T(V, V').$$

Finally, every state in src_0 is a fair state, hence every path through the funnel-loop template is a fair path of *Ex* (*FF.2*).

The following assignment to the parameters satisfies all these requirements: $p_0 = 0, p_1 = -1, p_2 = 0, p_3 = 0, p_4 = -1, p_5 = 0, p_6 = 1, p_7 = -1, p_8 = 0, p_9 = 1, p_{10} = 1, p_{11} = 0$. We can substitute these values in the funnel-loop template and obtain the following funnel-loop.

$$\begin{aligned}
src_0 &\doteq loop_r[0] \wedge x \geq y; & t_0 &\doteq loop_t[0];\\
src_1 &\doteq loop_r[1] \wedge x \geq y; & t_1 &\doteq loop_t[1] \wedge y' = -y;\\
src_2 &\doteq loop_r[2] \wedge x \geq -y; & t_2 &\doteq loop_t[2];\\
src_3 &\doteq loop_r[3] \wedge x \geq -y; & t_3 &\doteq loop_t[3];\\
src_4 &\doteq loop_r[4] \wedge x \geq -y; & t_4 &\doteq loop_t[4] \wedge y' = -y;\\
src_5 &\doteq loop_r[5] \wedge x \geq y; & t_5 &\doteq loop_t[5].
\end{aligned}$$

Notice that in this process the parametric expressions allowed us to identify an underapproximation of the transition relation of *Ex* that toggles the sign of y instead of allowing any possible assignment. In addition, the parametric inequalities restricted the regions we obtained from the candi-

date loop to only the states in which $x \geq |y|$, hence ensuring that the loop condition $x^2 \geq xy$ of our example holds. In fact, $x^2 \geq xy$ is redundant in src_0 and src_3; it is implied by $x \geq y \wedge y > 0$ in the first region and by $x \geq -y \wedge y < 0$ in the second one. Therefore, this funnel-loop is equivalent to the one we defined in §6.2.1.

7.2.2 Candidate Fair Loops Enumeration

We enumerate candidate fair loops by identifying lasso-shaped paths in the abstract space built by the assignments to a finite set of predicates. Two states that agree on the truth assignment of all such predicates belong to the same abstract state. We then represent the fair loop as a sequence of transitions and regions (sets of states) that underapproximate the transition relation of M.

Given a fair transition system $M \doteq \langle V, I^M, T^M, F^M \rangle$ we describe a candidate fair loop of length n for M, associated with an E-comp $H \doteq \langle V, I^H, T^H \rangle$ over regions $\mathcal{R} \doteq [R_i]_{i=0}^{n-1}$, assumptions $\mathcal{A} \doteq [A_i]_{i=0}^{n-1}$, ranking functions $\mathrm{R_F} \doteq [\mathrm{R_F}_i]_{i=0}^{n-1}$ and responsible for symbols $V^H \subseteq V$, as a sequence of regions $loop\_r \doteq [loop\_r_i(V)]_{i=0}^{n-1}$, transitions $loop\_t \doteq [loop\_t_i(V, V^{\neq H\prime})]_{i=0}^{n-2}$ and an initial state $\boldsymbol{v}_0$, where $V^{\neq H} \doteq V \setminus V^H$. Such that: (i) the state $\boldsymbol{v}_0$ is reachable in M; (ii) the conjunction of the first region and the initial states of H contains the initial state $\boldsymbol{v}_0$ (i.e. is not empty); (iii) the conjunction of a $loop\_r_i$ and the corresponding restricted region $R_i \wedge A_i$ underapproximates the fair states; (iv) for each step, the conjunction of $loop\_t_i$ and the transition relation T^H of H is an implicant for a transition in M. These

condition can be formally written as follows:

$$
\begin{aligned}
M &\rightsquigarrow \boldsymbol{v}_0;\\
\boldsymbol{v}_0 &\models \mathit{loop\_r}_0 \wedge I^H;\\
\exists i \forall V &: \mathit{loop\_r}_i \wedge R_i \wedge A_i \rightarrow F^M;\\
\forall i, V, V' &: \mathit{loop\_r}_i \wedge R_i \wedge A_i \wedge \mathit{loop\_t}_i \wedge T^H \wedge\\
&\quad ((\mathbf{0} < \mathrm{RF}_i \wedge R'_i) \vee (\mathrm{RF}_i = \mathbf{0} \wedge R'_{i+_n i})) \rightarrow T^M.
\end{aligned}
$$

Without loss of generality, and to simplify the presentation, we assume the fair region to be the first one: $\forall V : \mathit{loop\_r}_0 \wedge R_0 \wedge A_0 \rightarrow F^M$. The structure of a candidate loop resembles a funnel-loop. However, candidate loops are not guaranteed to satisfy all required hypotheses. In particular, the transitions $\mathit{loop\_t}_i \wedge T^H$ could admit deadlocks (Hyp. *F.1*) and they are not guaranteed to map every state in the previous region into some state in the following one (Hyp. *FL.1*). In addition, H may not provide all the required ranking functions. For this reason, in order to identify a funnel-loop, we look for a strengthening of the candidate loop that also satisfies these conditions.

The enumeration of candidate loops and compositions is performed by Alg. 2. The procedure is based on Bounded Model Checking (BMC) [31] for the enumeration of candidate paths, and on the computation of an underapproximation of M for each path.

The function ENCODE-L2S-FAIR-ABSTRACT-LOOP (line 1) encodes the search for a fair lasso-shaped path in the intersection of M and the composition of a subset of $\mathcal{H}$ into a reachability problem given by the 4-tuple $\langle V, I, T, \mathit{bad}\rangle$. The problem requires us to identify paths over the variables V, starting in $I(V)$ and following the steps given by $T(V, V')$ that end in some state in $\mathit{bad}(V)$. We obtain this encoding via a liveness-to-safety [29] construction that transforms the problem of identifying an abstract lasso into a reachability problem. The loop-back state is identified in the abstract space

Algorithm 2 GENERATE-CANDIDATE-LOOPS(M, $\mathcal{H}$)

▷ L2S encoding into reachability problem and E-comp selection.
1: $\langle V, I, T, bad\rangle \leftarrow$ ENCODE-L2S-FAIR-ABSTRACT-LOOP$(M, \mathcal{H})$
2: **for** $k \in [0, 1, 2, \ldots]$ **do** ▷ BMC unrolling: k steps.
3: $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$ ▷ BMC reachability.
4: $\langle sat, model\rangle \leftarrow$ SMT-SOLVE$(query)$ ▷ Find first path of length k.
5: $refs \leftarrow []$ ▷ Keep track of visited paths of length k.
6: **while** sat **do** ▷ Generate all candidates from paths of same length.
7: $H \leftarrow$ GET-CANDIDATE-COMPOSITION$(model)$ ▷ Path selects hints.
8: $\langle conflict\rangle \leftarrow$ GET-COMP-ERROR(H)
9: **if** $conflict \neq \bot$ **then** ▷ Learn incompatible transitions.
10: $\langle V, I, T, bad\rangle \leftarrow$ REMOVE-CONFLICT$(V, I, T, bad, conflict)$
11: **else** ▷ H is valid E-comp.
12: $\langle loop\_r, loop\_t\rangle \leftarrow$ UNDERAPPROXIMATE$(model, query, H)$
13: $\langle is\_ranked, rf\rangle \leftarrow$ RANK-LOOP$(loop_r, loop_t, H)$
14: **if** is_ranked **then** ▷ Learn ranking function.
15: $\langle V, I, T, bad\rangle \leftarrow$ REMOVE-RANKED-LOOPS(V, I, T, bad, rf)
16: **else** ▷ Unable to find ranking function, could be nonterminating.
17: $prefix \leftarrow$ GET-PREFIX$(model)$ ▷ Get stem of abstract lasso.
18: **yield** $\langle prefix, loop\_r, loop\_t, H\rangle$ ▷ Coroutine returns triples.
19: $refs.append(\neg(\bigwedge_{r \in loop\_r} r \wedge \bigwedge_{t \in loop\_t} t))$ ▷ Mark visited.
20: **end if**
21: **end if**
22: $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k) \wedge \bigwedge_{ref \in refs} ref$
23: $\langle sat, model\rangle \leftarrow$ SMT-SOLVE$(query)$ ▷ Find next path of length k.
24: **end while**
25: **end for**

defined by the predicates appearing in the transition relation and fairness condition of M, together with the ones in the E-comps. The last state and the loop-back state must agree on the truth assignment of all such predicates, hence they may not be the very same assignment. A set of fresh Boolean variables selects the E-comps to be considered, and the path must be such that at most one ranking function decreases at a time. We then rely

on a SMT-solver to identify fair lasso paths of increasing length k (line 2), as done for the abstract liveness-to-safety algorithm of [73]. The models of this BMC unrolling describe a path in the language of both M and the composition of a subset of the E-comps in $\mathcal{H}$. If $\mathcal{H}$ is empty or none of the hints is selected, GET-CANDIDATE-COMPOSITION (line 7) returns the trivial hint H of length equal to the number of states in the abstract lasso. Instead, if some hints are selected, H is given by their composition projected over the ordered sequence of regions visited by the path. The selected E-comps might not be compatible, for this reason, after extracting the candidate composition from the BMC model (line 7), GET-COMP-ERROR (line 8) checks if each transition in the composition is compatible (the trivial hint is trivally compatible). If this is not the case, a conflict predicate representing the transitions that are not compatible is used by REMOVE-CONFLICT to refine the reachability problem $\langle V, I, T, bad\rangle$ such that we avoid generating the same conflict again. If H is a valid E-comp the function UNDERAPPROXIMATE (line 12) computes two sequences of $n-1$ formulae $loop\_r \doteq [loop\_r_i(V)]_{i=0}^{n-2}$ and $loop\_t \doteq [loop\_t_i(V, V')]_{i=0}^{n-2}$ such that each $loop\_r_i \wedge loop\_t_i$, together with H, underapproximates the transition relation of M. The function computes an implicant for the formula *query* such that the assignments of the lasso described by *model* satisfy both formulae. Then, for each step i, the function partitions the predicates in the implicant into two sets. All predicates containing only symbols in V at step i are in $loop\_r_i$, while the predicates containing symbols in $V \cup V'$ at step i are in $loop\_t_i$. Therefore, we split the predicates used to describe the regions from the ones that constrain the transitions. We use $loop\_r_i$ and $loop\_t_i$ as formulae meaning the conjunction of all the predicates in the set and they, together with H, describe the candidate fair loop.

Then, function RANK-LOOP (line 13) tries to synthesise a ranking function for such candidate loop. In the literature there are many approaches

for the synthesis of ranking functions [133, 14, 41], here we simply assume we are given a method that returns the representation of a ranking function *rf* proving the termination of a candidate loop. If the procedure succeeds in identifying a ranking function, then the reachability problem $\langle V, I, T, bad \rangle$ is refined such that we avoid enumerating other loops ranked by the same function, as described in [73]. This is achieved by calling REMOVE-RANKED-LOOPS at line 15. Otherwise, at line 17, GET-PREFIX extracts from *model* the prefix of the loop; i.e. the path of M ending in the the loop-back state. The prefix represents a witness for the reachability of the first region of the candidate loop in M and the procedure returns it together with the current candidate loop, at line 18. If no other candidate loop of length k exists, we clear the list of refinements *refs* and enumerate the candidate loops of length $k + 1$.

Example. We now provide a brief example of the computation of the underapproximation of M described by *loop_r* and *loop_t*. Assume the transition relation of M is $T \doteq (a \leq 1 \rightarrow b' > b) \wedge (a \geq 2 \rightarrow b' < b)$, and the loop described by *model* is given by the assignments $a_0 = 1, b_0 = 0$, $a_1 = 2, b_1 = 1$ and $a_2 = 0, b_2 = 0$. A path of M with three states is represented by the formula $T(V_0, V_1) \wedge T(V_1, V_2)$. An implicant for such formula satisfied by *model* is $\{a_0 \leq 1, b_1 > b_0, a_1 \geq 2, b_2 < b_1\}$. Such an implicant can be obtained, for example, by applying the techniques presented in [77] and [147]. Finally, we partition this set into *loop_r* and *loop_t* by defining their components as follows: $loop\_r_0 \doteq a \leq 1$, $loop\_t_0 \doteq b' > b$, $loop\_r_1 \doteq a_1 \geq 2$ and $loop\_t_1 \doteq b' < b$. Therefore, we have obtained a candidate loop where the first region requires $a \leq 1$, then it follows the transition relation $b' > b$ to reach the second region $a_1 \geq 2$, finally the transition $b' < b$ concludes the abstract loop by going back to the first region. Notice that such candidate loop does not immediately correspond to a funnel-loop. For

example, it is not the case that $loop\_r_0(V) \wedge loop\_t_0(V, V') \rightarrow loop\_r_1(V')$. For this reason, we employ them to obtain funnel-loop templates that we later try to instantiate into actual funnel-loops.

7.2.3 Funnel-loop Templates

We call funnel-loop template a candidate funnel-loop whose predicates contain symbols of both V and a set of parameters P (P and V are disjoint). We call parameter a fresh (not in V) integer or real variable for which we want to identify an interpretation that satisfies some constraints. In more detail, for each candidate fair loop we generate a sequence of funnel-loop templates and try to identify an assignment to the symbols P such that by replacing them with the identified values we obtain a funnel-loop satisfying all the required hypotheses. In the following, the function called NEW-PARAM-EXPR generates expressions over the symbols V and the parameters P, e.g. affine linear functions $p_0 + \sum_{i=1}^{|V|} p_i v_i$, where $|V|$ is the number of symbols in V and for all i, $p_i \in P$ and $v_i \in V$. The function introduces as many parameters as necessary to generate the required expressions. Alg. 3 shows the procedure we use to generate funnel-loop templates from a candidate loop. We generate templates of the same length of the candidate loop. Function HEURISTIC-PICK-NUM-INEQS (line 1) selects a list of natural numbers to be used to generate the funnel-loop templates. Each number corresponds to the amount of parametric inequalities added to each region of the candidate loop to define the corresponding source region of a funnel template (line 7). The higher the number the more freedom will the template have in shrinking the regions, but in the search problem we will have more parameters, hence a larger space to explore. Notice that, since the first region of the candidate loop is fair by construction, then the last destination region in the funnel-loop template will be fair and Hyp. *FF.2* holds. We create the i^{th} funnel of the funnel-loop tem-

Algorithm 3 GENERATE-TEMPLATES($\boldsymbol{v}_0, loop\_r, loop\_t, H$)

1: $ineqs \leftarrow$ HEURISTIC-PICK-NUM-INEQS($loop\_r, loop\_t, H$)
2: $\langle V^H, I^H, T^H, \mathcal{R}, \mathcal{A}, \text{RF} \rangle \leftarrow H$ ▷ Get components defining H.
3: **for** $ineq \in ineqs$ **do** ▷ Use $ineq$ parametric inequalities in regions.
4: $n \leftarrow \textbf{len}(loop\_r)$ ▷ Length of template + 1: loop-back region.
5: $funnels \leftarrow []$ ▷ List of funnels for funnel-loop template.
6: **for** $i \in [0..n-2]$ **do** ▷ Create i^{th} funnel: $\langle V, src, t, rf, dst \rangle$.
7: $src \leftarrow loop\_r[i] \wedge \mathcal{R}[i] \wedge \mathcal{A}[i] \wedge \bigwedge_{j=0}^{ineq-1} \text{NEW-PARAM-EXPR}(V) \geq 0$
8: **if** $\exists V : \mathbf{0} < \text{RF}[i](V)$ **then**
9: $rf \leftarrow \text{RF}[i]$ ▷ H defines ranking function.
10: **else**
11: $rf \leftarrow \text{NEW-PARAM-EXPR}(V)$ ▷ Parametric ranking function.
12: **end if**
13: $t \leftarrow \mathcal{R}[i] \wedge \mathcal{A}[i]$ ▷ Transition of H in i^{th} region.
14: $t \leftarrow t \wedge T^H \wedge ((\mathbf{0} < rf \wedge \mathcal{R}[i]' \wedge rf' \leq rf) \vee (rf = \mathbf{0} \wedge \mathcal{R}[i+1]'))$
15: **for** $v_{i+1} \in V_{i+1} \setminus V_{i+1}^H$ **do** ▷ Add functional assign for v_{i+1} in t
16: **if** $v_{i+1} = f(V_i) \in loop\_t[i]$ for some function f **then**
17: $t \leftarrow t \wedge v_{i+1} = f(V_i)$ ▷ Functional assignment in candidate.
18: **else**
19: $t \leftarrow t \wedge v_{i+1} = \text{NEW-PARAM-EXPR}(V_i)$ ▷ Create new expr.
20: **end if**
21: **end for**
22: $P \leftarrow \text{COLLECT-PARAMETERS}(src, rf, t)$ ▷ Params in current funnel.
23: $dst(V', P) \leftarrow \exists V \ : \ src(V, P) \wedge rf(V, P) = \mathbf{0} \wedge t(V, V', P)$
24: $funnels.append(\text{FUNNEL}(src, t, rf, dst))$
25: **end for**
26: **yield** FUNNEL-LOOP($funnels, \boldsymbol{v}_0$) ▷ Coroutine returns templates.
27: **end for**

plate (lines 6–25) as a restriction of the conjunction of the i^{th} region and transition of the candidate loop. In addition, the only nondeterministic component in t is given by the transition relation of H. All other components of the transition relation t of the funnel are deterministic functional assignments as follows. Let V^H be the symbols for which H is responsible.

For each symbol $v_{i+1} \in V_{i+1} \setminus V_{i+1}^H$, if $loop\_t_i$ already contains a functional assignment for v_{i+1}, then we use that (line 17). Otherwise, we generate a functional assignment for v_{i+1} as a parametric expression over the symbols in V_i (line 19). The resulting transition relation is total and Hyp. *F.1* holds. We define the destination region of a funnel implicitly as the set of states reachable in one step from $S \wedge \text{RF} = \mathbf{0}$ (line 23), hence Hyp. *F.4* holds by construction. Finally, at line 26, the procedure returns the funnel-loop template associated with the list of parametric funnels and initial state $\boldsymbol{v}_0$. We will ensure that $\boldsymbol{v}_0$ is in the first source region of the funnel-loop. Therefore, since $\boldsymbol{v}_0$ is reachable in M, Hyp. *FF.1* holds.

Example. We now provide an example to illustrate how a funnel is generated in the lines from 7 to 24. In this example we assume the following: (i) $V \doteq \{a, b, c\}$ is a set of real-valued symbols; (ii) NEW-PARAM-EXPR generates affine linear expressions over V and a set of parameters $P \doteq \{p_i\}_{i \in \mathbb{N}}$; (iii) we are constructing a funnel-loop template adding a single predicate to shrink the region ($ineq = 1$); (iv) $loop\_r[i] \doteq b < c$; (v) $loop\_t[i] \doteq c' = c \wedge b' > b + a \wedge b' > c$ and (vi) the hint H responsible for $\{a\}$ has the following components: $\mathcal{R}[i] \doteq a > 0$, $\mathcal{R}[i+1] \doteq a > 0$, $\mathcal{A}[i] \doteq \top$, $\text{RF}[i] \doteq \mathbf{0}$ and $T^H \doteq a' > a$.

For simplicity, we defined P as an infinite set. However, in this example we will use 12 parameters $\{p_i\}_{i=0}^{11}$; we will introduce 3 affine parametric expressions each of which requires 4 parameters. The first expression represents an additional inequality for the region, the second one is used to represent the ranking function, and the last one corresponds to the functional assignment of b' in the transition relation.

Line 7 defines the source region *src* of the funnel as the conjunction of the $loop\_r[i]$, the restricted region of H and, since $ineq = 1$ it introduces a

single parametric predicate: $p_0 + p_1 a + p_2 b + p_3 c \geq 0$.

$$src(V, P) \doteq b < c \wedge a > 0 \wedge p_0 + p_1 a + p_2 b + p_3 c \geq 0.$$

The condition at line 8 is false since the ranking function provided by H is always equal to $\mathbf{0}$. The procedure then executes line 11 that introduces a new parametric expression to represent the ranking function:

$$rf(V, P) \doteq p_4 + p_5 a + p_6 b + p_7 c.$$

We implicitly consider the function equal to $\mathbf{0}$ if $rf(V, P) \leq 0$. Then, line 14 initialises t from the transition relation of H as:

$$t \doteq a > 0 \wedge a' > a \wedge ((0 < rf(V, P) \wedge a' > 0 \wedge rf(V', P) \leq rf(V, P)) \vee (rf(V, P) \leq 0 \wedge a' > 0)).$$

The loop starting at line 15 iterates over the symbols in $\{b, c\}$. Consider first the symbol c, in $loop\_t[i]$ we find the equality $c' = c$, hence the condition at line 16 holds and the equality is added to t as a conjunct. Consider now the symbol b, $loop\_t[i]$ prescribes no equality for b', hence a new parametric expression is introduced and added to t at line 19; let such equality be $b' = p_8 + p_9 a + p_{10} b + p_{11} c$. Therefore, the final t is as follows:

$$t \doteq a > 0 \wedge a' > a \wedge ((0 < rf(V, P) \wedge a' > 0 \wedge rf(V', P) \leq rf(V, P)) \vee (rf(V, P) \leq 0 \wedge a' > 0)) \wedge c' = c \wedge b' = p_8 + p_9 a + p_{10} b + p_{11} c.$$

Finally, dst is defined as the set of states that admit a predecessor through t in src with $rf = \mathbf{0}$:

$$dst(a', b', c', P) \doteq \exists a, b, c : src(a, b, c, P) \wedge rf(a, b, c, P) \leq 0 \wedge t(a, b, c, a', b', c', P).$$

7.2.4 Funnel-loop Synthesis Problem

We now describe the $\exists\forall$ quantified formula that corresponds to the synthesis problem of a funnel-loop template. Alg. 1 computes this formula

for every funnel-loop template *template* via the method *ef_constraints* at line 4. We search for an assignment to the parameters P of the funnel-loop template such that by replacing them with the identified values we obtain a funnel-loop that satisfies all hypotheses of Defs. 18, 19 and of Th. 6. In the hypotheses, for every funnel $fnl_i \doteq \langle V, S_i, T_i, D_i, \mathrm{RF}_i \rangle$, we replace each destination region D_i with the quantified formula:

$$D_i(V') \doteq \exists V : S_i(V) \wedge \mathrm{RF}_i(V) = \mathbf{0} \wedge T_i(V, V'). \tag{7.8}$$

Every instance of the funnel-loop template must contain a fair region since $loop\_r_0$ is a subset of the fair states and S_0, by construction, underapproximates $loop\_r_0$. We ensure that Hyp. *FF.1* holds by requiring that $\boldsymbol{v}_0$ is in the source region of the first funnel fnl_0 with the constraint:

$$\exists P \ : \ S_0(\boldsymbol{v}_0, P). \tag{7.9}$$

Hyp. *F.1* holds by construction, since the next region implies the assumptions required by the E-comp. Therefore, the transition relation of the E-comp must always allow for a successor for all assignments to the $V^{\neq H'}$. In addition, the other components of the transition relation of the funnel describe a functional assignment to the $V^{\neq H'}$ without any circular dependency. Hyp. *F.4* holds since we implicitly defined the destination region of each funnel fnl_i as the set of states reachable in one step from $S_i \wedge \mathrm{RF}_i = \mathbf{0}$. Then, we ensure that every instantiation of every funnel template fnl_i in the funnel-loop template satisfies hypotheses *F.2* and *F.3* by requiring the following to hold:

$$\begin{aligned} \exists P \ \forall V, V' : (S_i(V, P) \wedge \mathbf{0} < \mathrm{RF}_i(V, P) \wedge T_i(V, V', P)) \rightarrow \\ S_i(V', P) \wedge \mathrm{RF}_i(V', P) < \mathrm{RF}_i(V, P). \end{aligned} \tag{7.10}$$

Consider now Hyp. *FL.1*; this formula requires the funnels to be correctly chained. Notice that Hyp. *FL.1* consists of an implication whose left-hand-side is D_i. We replace D_i with its definition (Eq. (7.8)) and rewrite

it into an equivalent formula in which we bring the existential quantifier out in front of the formula as a universal quantifier. Therefore, to ensure that Hyp. *FL.1* holds, we require every two consecutive funnel templates fnl_i and $fnl_{i+_n 1}$ in the funnel-loop template to satisfy the following:

$$\exists P\ \forall V, V' : (S_i(V,P) \wedge \mathrm{RF}_i(V,P) = \mathbf{0} \wedge T_i(V,V',P)) \rightarrow S_{i+_n 1}(V',P). \tag{7.11}$$

This ensures that every D_i is a subset of $S_{i+_n 1}$. We have observed above that S_0 contains only fair states, hence D_{n-1} underapproximates the fair states and Hyp. *FF.2* holds. Finally, we ensure that each funnel-loop instance underapproximates M (Hyp. *FF.3*) by requiring the following to hold for every funnel fnl_i:

$$\exists P\ \forall V, V'\ :\ S_i(V,P) \wedge T_i(V,V',P) \rightarrow T^M(V,V'). \tag{7.12}$$

The final synthesis problem is then given by the conjunction of all the constraints (7.9)–(7.12). Notice that their conjunction is a formula of the form $\bigwedge_{i=0}^{3} \exists P \forall V, V' : \phi_i$, since Eq. (7.9) does not contain any symbol in $V \cup V'$. We can rewrite the conjunction into an equivalent formula by bringing the quantifiers in front and obtain a single equivalent synthesis problem $\exists P \forall V, V' : \bigwedge_{i=0}^{3} \phi_i$. A solution for this problem is a model that assigns a value to each parameter in P such that the formula $\bigwedge_{i=0}^{3} \phi_i$ holds for all possible assignments to the symbols in $V \cup V'$. From one such model we can generate a concrete funnel-loop by substituting the parameters P with their assignment.

Chapter 8

Synthesis of E-comps

The procedure described in the previous chapter takes as input a set of E-comps. This chapter describes two procedures that could automatise or help in their definition. In the literature there are many techniques that consider problems with a particular structure, some examples are linear software programs, timed automata and dynamical systems. Some of these techniques can be used to define E-comps for a system. From this perspective, the compositional approach described in Sec. 6.3 allows for the use of specialised techniques to analyse portions of the system with different characteristics. The main objective of this chapter is to show that E-comps are sufficiently expressive to represent infinite paths identified using different techniques employed in different contexts.

Generally speaking, every such technique has some assumptions on the systems it considers and adopts some structure to represent one or more of its infinite executions. Given one such technique, we are faced with two key challenges for its adoption in the synthesis of E-comps. First, we need to explore the fair transition system to identify components or portions of it that satisfy the assumptions required by the technique, e.g. by identifying components representable as lasso programs or timed automata. We could search for such components, for example, by analysing the model syntac-

tically and looking for constraints over a subset of the symbols matching some pattern. Another possibility is to enumerate candidate (abstract) loops of the system, as in Alg. 2, and check whether the candidate loop implies a structure satisfying the required constraints. The second challenge is to define a transformation from the structure the approach uses to represent infinite paths to E-comps. These two steps strongly depend on the considered approach and need to be defined for each case.

The chapter is organised as follows. Sec. 8.1 considers Lyapunov stability, a widely used concept in the study of dynamical systems, and defines an E-comp representing the stability condition of the system. Instead, Sec. 8.2 describes a novel technique to identify traces with ultimately diverging symbols and defines a corresponding E-comp.

8.1 Lyapunov Stability

Lyapunov stability [152] is a central notion in the study of dynamical systems. Intuitively, the property states that if the systems begins in a configuration sufficiently close to an equilibrium point, then every state in the execution will lay within a neighbourhood of the equilibrium.

Definition 25 - Lyapunov stability
Given a set of symbols V, a continuous function $f : V \mapsto V$ and a distance function $d : V \mapsto \mathbb{R}_0^+$ such that $\langle V, d\rangle$ is a metric space. A state $\boldsymbol{v}_\epsilon$ is Lyapunov stable iff:

$$\forall \epsilon > 0 \exists \delta > 0 \forall n \in \mathbb{N}, V : d(\boldsymbol{v}_\epsilon, V) < \delta \rightarrow d(f^n(\boldsymbol{v}_\epsilon), f^n(V)) < \epsilon.$$

A state $\boldsymbol{v}_\epsilon$ is Lyapunov stable with respect to a transition relation $V' = f(V)$, if the distance between the corresponding states of every path starting from $\boldsymbol{v}_\epsilon$ and every other path starting from some state distant less that δ from $\boldsymbol{v}_\epsilon$, is always bounded by ϵ. The property guarantees the

existence of a δ neighbourhood for the initial state $\boldsymbol{v}_\epsilon$ such that all paths starting from some state in that neighbourhood have distance at most ϵ from the one starting from $\boldsymbol{v}_\epsilon$. In particular, we can constrain the paths to be arbitrarily close to the one starting in $\boldsymbol{v}_\epsilon$ by choosing small values for ϵ. The property ensures that for any such ϵ there will be a δ neighbourhood of state $\boldsymbol{v}_\epsilon$ such that the distance between any path starting in such neighbourhood and the path starting in $\boldsymbol{v}_\epsilon$ is bounded by ϵ.

We now define an E-comp H with a single region R that represents such paths. In the hypothesis above, we Skolemize δ and obtain a function $\delta : \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$ that given the selected value for ϵ computes the corresponding value for δ such that the hypothesis holds. This allows us to define H such that we nondeterministically select ϵ in the initial state and, depending on this choice, the region R provides tighter or coarser bounds. In more detail, let $H \doteq \langle V^M \cup V_\epsilon \cup \{\epsilon\}, I, T\rangle$ responsible for V, with ranking functions $\mathcal{W}$ and assumptions $\mathcal{A}$ such that: (i) ϵ is a fresh real valued symbol and $V_\epsilon \doteq \{v_\epsilon | v \in V\}$; (ii) $I \doteq \epsilon > 0 \wedge d(V, V_\epsilon) < \delta(\epsilon)$; (iii) $T \doteq \epsilon' = \epsilon \wedge V' = f(V) \wedge V'_\epsilon = f(V_\epsilon)$; (iv) every ranking function $\textsc{Rf} \in \mathcal{W}$ is always equal to the minimal $\textsc{Rf} = \mathbf{0}$; (v) finally, we define the region R as $d(V, V_\epsilon) < \delta(\epsilon)$. With respect to the assumptions $\mathcal{A}$, if our whole system is represented by the continuous function f, hence it has no discrete component, then all assumptions are empty ($\top$). Otherwise, we need to inspect the system and characterise the conditions under which its behaviour with respect to a subset of the symbols $V_L \subseteq V$ can be underapproximated by a continuous function as above. If such conditions can be underapproximated by some formula over $V \setminus V_L$, then such conditions define the assumptions of the E-comp responsible for V_L built from the Lyapunov stability of the continuous function $f : V_L \mapsto V_L$.

Notice that we have defined a single E-comp representing an unbounded number of possible bounds on the distance between the configurations. In

addition, if $\boldsymbol{v}_\epsilon$ is a fixpoint for f (i.e. $f(\boldsymbol{v}_\epsilon) = \boldsymbol{v}_\epsilon$) then H can be greatly simplified and the region simply bounds the distance from the fixpoint.

8.2 Ultimately Diverging Symbols

Note. The approach described in this section is based on the one we described in [54]. The work [54] is a collaboration with Stefano Tonetta and Marco Roveri and presents an ad-hoc technique to identify infinite traces of timed transition systems where some clock variables can diverge. This section extends its applicability by considering infinite-state systems and by relying on the computation of absolute positiveness bounds. In addition, the approach is presented in the compositional framework of Sec. 6.3.

This section describes a novel approach to identify infinite executions in which some symbols diverge.

A transition system allows or forbids some behaviour based on the truth value of the atomic predicates used to describe its transition relation. Therefore, if we identify a region where the truth assignment of all such predicates is constant and the region is closed with respect to the transition relation, then the system admits some infinite execution remaining in such region. In this sense we are looking for a recurrent set or stable region, where stability is interpreted with respect to the truth assignment of the predicates describing the system. In more detail, we are interested in regions where all assignments of some symbol above [resp. below] a certain threshold share the same truth assignment for the predicates in the model. An example of such regions are the unbounded ones defined by the region abstraction of timed automata. When a clock symbol is greater than the maximum constant to which it is compared to, the distance between the clock evaluation and such threshold is irrelevant and all such states

correspond to the same abstract state.

We employ the same idea in the context of infinite-state transition systems. First, in §8.2.1, we define a set of sufficient conditions and, given a structure satisfying the conditions, we build a corresponding E-comp. Then, §8.2.2 describes a procedure to extract such structures from an infinite-state transition system. Finally, §8.2.3 provides an example.

8.2.1 Sufficient Conditions and *E*-comp

Consider a set of symbols V partitioned into two sets V_D and $V_B \doteq V \setminus V_D$ such that every symbol in V_D has an infinite domain, either reals or integers. Let $\mathcal{P}$ be a set of atomic predicates over $V \cup V'$ such that every predicate can be written in the forms $f(V) \bowtie 0$ or $x' \bowtie f(V)$, where $\bowtie \in \{<, \leq, \geq, >\}$ and $x' \in V'_D$. We define a set of sufficient conditions for the symbols in V_D to remain in a region provided the ones in V_B satisfy some assumptions. We consider the case in which all V_D are positive in the region. The case with some negative variable $v \in V_D$ can be reduced to the case above by replacing every occurrence of v with $-v$.

We partition $\mathcal{P}$ into disjoint subsets based on their syntactic structure. The first set Curr contains all predicates that depend only on the current state variables V; then, we define for every $x \in V_D$ four different sets, two for the expressions representing lower bounds of x' (strict and non-strict) and two for its upper bounds.

$$
\begin{aligned}
\textsc{Curr} &\doteq \{f(V) \bowtie 0 \mid \ \bowtie \in \{\leq, <, \geq, >\} \wedge f(V) \bowtie 0 \in \mathcal{P}\};\\
\textsc{Us}_x &\doteq \{u(V) \mid x' < u(V) \in \mathcal{P}\};\\
\textsc{Ue}_x &\doteq \{u(V) \mid x' \leq u(V) \in \mathcal{P}\};\\
\textsc{Ls}_x &\doteq \{l(V) \mid x' > l(V) \in \mathcal{P}\};\\
\textsc{Le}_x &\doteq \{l(V) \mid x' \geq l(V) \in \mathcal{P}\}.
\end{aligned}
$$

We define the region for V_D where the predicates in $\mathcal{P}$ are stable relative to V_B as $R(V) \doteq \bigwedge_{x \in V_D} x \geq \mathit{bound}_x(V_B)$, where $\mathit{bound}_x(V_B)$ is an expression representing the lower bound for each $x \in V_D$; we assume the domain of $\mathit{bound}_x(V_B)$ to be included in the domain of x. In addition, let $\mathit{bound}_{V_D}(V_B)$ be the vector of $\{\mathit{bound}_x\}_{x \in V_D}$ and $A(V_B)$ be a formula representing the required conditions on the symbols in V_B. Therefore, we are considering a restricted region $R(V) \wedge A(V_B)$.

We now describe the conditions that must hold for the truth assignment of the predicates $\mathcal{P}$ to be stable relative to V_B in the restricted region. First we require the assignment to the predicates depending only on the current state variables to be constant relative to V_B in the region:

$$\forall V : (R \wedge A) \rightarrow \bigwedge_{f \bowtie 0 \in \textsc{Curr}} (f(\mathit{bound}_{V_D}(V_B), V_B) \bowtie 0 \leftrightarrow f(V_D, V_B) \bowtie 0). \tag{8.1}$$

Eq. (8.1) implies that, in the restricted region, the truth assignment of the predicates $\mathcal{P}$ depends only on the assignment to V_B. Given an assignment to V_B, the value of $\mathit{bound}_{V_D}(V_B)$ is uniquely identified, hence also the truth value of all $f(\mathit{bound}_{V_D}(V_B), V_B) \bowtie 0$. In order for Eq. (8.1) to hold, this truth assignment must correspond to the one of all the $f(V_D, V_B) \bowtie 0 \in$ Curr for all V_D in R. Therefore, given an assignment to the V_B, all assignments to the symbols in V_D, such that $R \wedge A$ holds, must agree on the truth value of all predicates in Curr. For example, no f can be of the form $\alpha x - \beta y$, for some $\alpha, \beta > 0$ and $x, y \in V_D$. In fact, both bound_x and bound_y depend only on V_B, hence their value is independent from the assignments to x and y. The truth value of $\alpha x - \beta y \bowtie 0$ depends on whether x is less or greater than $\frac{\beta}{\alpha} y$, which is not taken into account in the formula above. However, f could be defined as the sum of symbols in V_D, since if every symbol is greater than 0, then f is always positive.

We now consider the predicates containing one next state symbol and let

$x' \in V'_D$ be such symbol. We need to guarantee that the predicates always allow for a successor. Therefore, the minimum upper bound of x (i.e. minimum expression in $\text{Us}_x \cup \text{Ue}_x$) must be greater than the maximum of its lower bounds ($\text{Ls}_x \cup \text{Le}_x$). Furthermore, we need to ensure that between these two expressions there always is at least one element that is also in the domain of x. If x is a real variable, then every nonempty interval contains such a value. Otherwise, x has integer domain and we need to ensure that the interval contains at least one element of $\mathbb{Z}$. Let $Int(x)$ hold iff x has integer domain, then we can formally write the requirement above as follows.

$$\forall V : (R \wedge A) \rightarrow \Big(\bigwedge_{u \in \text{Us}_x} \Big(\bigwedge_{l \in \text{Ls}_x \cup \text{Le}_x} u > l \Big) \wedge \bigwedge_{u \in \text{Ue}_x} \Big(\bigwedge_{l \in \text{Ls}_x} u > l \wedge \bigwedge_{l \in \text{Le}_x} u \geq l \Big) \wedge Int(x) \rightarrow \Big(\bigwedge_{u \in \text{Ue}_x \cup \text{Us}_x} \bigwedge_{l \in \text{Le}_x \cup \text{Ls}_x} \lfloor u \rfloor \geq \lceil l \rceil \Big) \Big). \tag{8.2}$$

Finally, we require $bound_{V_D}(V_B)$ to be a minimum for every $u \in \text{Us}_x \cup \text{Ue}_x$ and $x \in V_D$.

$$\forall x \in V_D, u_x \in U_x, V : (R \wedge A) \rightarrow u_x(V) \geq u_x(bound_{V_D}(V_B), V_B). \tag{8.3}$$

Therefore, as for the predicates in Curr, for two symbols $x, y \in V_D$ the upper bound functions $u_x \in U_x$ cannot be of the form $\alpha x - \beta y$ with $\alpha, \beta > 0$.

E-comp definition

Given V_B, V_D, the bound expressions $\{bound_x(V_B)\}_{x \in V_D}$ and formula $A(V_B)$ satisfying equations (8.1), (8.2) and (8.3) with respect to the set of predicates $\mathcal{P}$. We define an E-comp $H \doteq \langle V, I, T \rangle$ of size $|\mathcal{R}^{|V_D|} \times \mathcal{R}^{|V_D|} \times 2^{\text{Curr}}|$ responsible for V_D over regions $\mathcal{R}$, assumptions $\mathcal{A}$ and ranking functions $\mathcal{W}$. We define all ranking functions as always equal to their minimal element. Then, each restricted region is the strengthening of $R \wedge A$ such

that in every region (i) $bound_{V_D}$ is bounded from above by some constant $max_b \in \mathbb{R}^{|V_D|}$ for every symbol in V_D, (ii) each $u \in \text{Us}_x \cup \text{Ue}_x$ is bounded from below by some constant $min_u[x] \in \mathbb{R}$ and (iv) R prescribes a truth assignment to all predicates in Curr. Every predicate in Curr is the comparison of an expression with 0, hence its negation can be equivalently written by inverting the comparison operator. For this reason, every truth assignment to the predicates in Curr can be written as a conjunction $\bigwedge_{i=0}^{|\text{Curr}|-1} f_i(V) \bowtie 0$. In the following we write 2^{Curr} for the set of such formulae and we also assume a total order such that every $0 \leq h < 2^{|\text{Curr}|}$ uniquely identifies the formula $2^{\text{Curr}}[h]$ in the set. We define the set of regions as $\mathcal{R} \doteq \{R_{max_b,min_u,h}\}_{max_b \in \mathbb{R}^{|V_D|}, min_u \in \mathbb{R}^{|V_D|}, h \in \{0,\ldots,2^{|\text{Curr}|}-1\}}$ and assumptions as $\mathcal{A} \doteq \{A_{max_b,min_u,h}\}_{max_b \in \mathbb{R}^{|V_D|}, min_u \in \mathbb{R}^{|V_D|}, h \in \{0,\ldots,2^{|\text{Curr}|}-1\}}$, where: each $R_{max_b,min_u,h}$ is defined as

$$R_{max_b,min_u,h}(V) \doteq R \wedge 2^{\text{Curr}}[h](V_D, V_B);$$

every $A_{max_b,min_u,h}$ is given by

$$\begin{aligned} A_{max_b,min_u,h}(V_B) \doteq A \wedge 2^{\text{Curr}}[h](bound_{V_D}(V_B), V_B) \wedge bound_{V_D}(V_B) \leq max_b \wedge \\ \bigwedge_{x \in V_D} \bigwedge_{u \in \text{Us}_x} u(bound_{V_D}(V_B), V_B) > min_u[x] \wedge \\ \bigwedge_{u \in \text{Ue}_x} u(bound_{V_D}(V_B), V_B) \geq min_u[x]; \end{aligned}$$

$min_u[x]$ is the component of min_u relative to symbol $x \in V_D$ and, as above, $R \doteq \bigwedge_{x \in V_D} x \geq bound_x(V_B)$. Therefore, every restricted region prescribes both an upper bound for the $\{bound_x\}_{x \in V_D}$ and a lower bound for every expression in $\bigcup_{x \in V_D} \text{Us}_x \cup \text{Ue}_x$. In addition, every restricted region implies a specific truth assignment for the predicates in Curr. The bounds and the truth assignments are uniquely identified by the triple we use to index both regions $\mathcal{R}$ and assumptions $\mathcal{A}$.

We define the initial states as the disjunction of the restricted regions:

$$I \doteq \bigvee_{max_b \in \mathbb{R}^{|V_D|}} \bigvee_{min_u \in \mathbb{R}^{|V_D|}} \bigvee_{h=0}^{2^{|\textsc{Curr}|}-1} R_{max_b,min_u,h} \wedge A_{max_b,min_u,h}.$$

Finally, the transition relation is given by the conjunction of the upper and lower bounds of every symbol in V_D and requires the minimum of every upper bound to be greater than the corresponding bound in the next state. Assuming the restricted regions to be disjoint we define it as follows.

$$\begin{aligned} T \doteq & \bigwedge_{x \in V_D} \bigwedge_{u \in \textsc{Us}_x} x' < u(V) \wedge \bigwedge_{u \in \textsc{Ue}_x} x' \leq u(V) \wedge \bigwedge_{l \in \textsc{Ls}_x} x' > l(V) \wedge \bigwedge_{l \in \textsc{Le}_x} x' \geq l(V) \wedge \\ & \bigwedge_{max_b \in \mathbb{R}^{|V_D|}} \bigwedge_{min_u \in \mathbb{R}^{|V_D|}} \bigwedge_{h=0}^{2^{|\textsc{Curr}|}-1} R_{max_b,min_u,h} \wedge A_{max_b,min_u,h} \rightarrow \\ & \bigwedge_{max'_b \in \mathbb{R}^{|V_D|}} \bigwedge_{min'_u \in \mathbb{R}^{|V_D|}} \bigwedge_{h'=0}^{2^{|\textsc{Curr}|}-1} R'_{max'_b,min'_u,h'} \wedge A'_{max'_b,min'_u,h'} \rightarrow \\ & min_u \geq max'_b \wedge \bigwedge_{x \in V_D} Int(x) \rightarrow \lfloor min_u[x] \rfloor \geq \lceil max'_b[x] \rceil. \end{aligned}$$

The first line of the formula requires the next assignment of every symbol in V_D to lay between its corresponding upper and lower bounds. Then, the last three lines allow only the transitions between the regions with indexes $\langle max_b, min_u, h \rangle$ and $\langle max'_b, min'_u, h' \rangle$ such that the minimum upper bound of the current region min_u is greater than the maximum upper bound max'_b for $bound_{V_D}$ in the next region.

Notice that both I and T are infinite formulae. Moreover, also $\mathcal{R}$ and $\mathcal{A}$ contain an unbounded number of elements. In fact, they are indexed via triples of the form $\langle max_b, min_u, h \rangle$, where $max_b, min_u \in \mathbb{R}^{|V_D|}$ have infinite domain. Therefore, H has an infinite number of regions and, since this is not allowed by Def. 20, H is not an E-comp. However, we prove that any projection of H on a finite number of its regions satisfies all required hypotheses.

Theorem 17 - Every finite projection of H is an E-comp
Every projection of H on a finite number of its regions satisfies all hypotheses of Def. 20.

Proof. We need to show that H satisfies hypotheses *EC.1*, *EC.2*, *EC.3* and *EC.4*. Then, every projection of H will also satisfy them by the same argument of Th. 13, and if the projection is finite then it must be a E-comp. Hyp. *EC.1* holds for H by construction; in fact, the initial states are defined as the disjunction of the restricted regions. Hypotheses *EC.2* and *EC.3* hold trivially since all ranking functions in $\mathcal{W}$ are always equal to their minimal element. Therefore, we only need to show that Hyp. *EC.4* holds for H.

Consider every pair of restricted regions. If no transition exists from the first region to the second one, then we do not need to prove anything. Otherwise, assume there exists a transition between the restricted region with index $\langle max_b, min_u, h\rangle$ and the one with index $\langle max'_b, min'_u, h'\rangle$. We need to show that every state in the first one admits a successor in the second one. By definition of T, every next assignment to the symbols in V_D must lay between its upper and lower bounds. Eq. (8.2) ensures that there exists a valid assignment in the domain of x satisfying such constraint for every $x \in V_D$.

We now need to show that for every $x \in V_D$ among such values there exists one that is in $R_{max'_b,min'_u,h'}$, assuming $A_{max'_b,min'_u,h'}$ holds. $R_{max'_b,min'_u,h'}$ is composed of 2 conjuncts: one requires every x to be greater than its bound, the other one prescribes the assignment to the predicates in Curr.

Consider the first conjunct, we need to show that for every $x \in V_D$ among the possible next assignments that lay between its upper and lower bounds, there exists at least one that is also greater than $bound_x$ computed over the next state variables. By definition of T, $min_u \geq max'_b$. In addition, the definition of $A_{max_b,min_u,h}$ implies that every state in the first restricted

region satisfies the following formula:

$$\bigwedge_{x \in V_D} \bigwedge_{u \in \mathrm{Us}_x} u(bound_{V_D}, V_B) > min_u[x] \wedge \bigwedge_{u \in \mathrm{UE}_x} u(bound_{V_D}, V_B) \geq min_u[x].$$

Finally, Eq. (8.3) ensures that $u(V) \geq u(bound_{V_D}, V_B)$. Therefore, every state in the restricted region $\langle max_b, min_u, h \rangle$ is such that every upper bound expression of every $x \in V_D$ is greater than min_u, hence also than max'_b:

$$\bigwedge_{x \in V_D} \bigwedge_{u \in \mathrm{Us}_x} u(bound_{V_D}, V_B) > max'_b[x] \wedge \bigwedge_{u \in \mathrm{UE}_x} u(bound_{V_D}, V_B) \geq max'_b[x].$$

In addition, for every symbol $x \in V_D$ with integer domain, $\lfloor min_u[x] \rfloor \geq \lceil max'_b[x] \rceil$ must hold. This implies the existence of at least one integer value in the interval $[max'_b[x], min_u[x]]$.

Consider now the second conjunct of $R_{max'_b, min'_u, h'}$. We need to prove that the states in the region share the same truth assignment to the predicates in CURR. $A_{max'_b, min'_u, h'}$ implies that $2^{\mathrm{CURR}}[h](bound_{V_D}(V_B), V_B)$ holds. Eq. (8.1) ensures that all assignments such that $\bigwedge_{x \in V_D} x \geq bound_x(V_B)$ agree on that truth assignment and, by construction, $\bigwedge_{x \in V_D} x \geq bound_x(V_B)$ holds in the region. □

8.2.2 Synthesis

We now describe a possible strategy to extract sets of predicates satisfying equations (8.1), (8.2) and (8.3). We achieve this objective by relying on the concept of *absolute positiveness* for a multivariate function. Absolute positiveness allows us to define two overapproximated bounds for each multivariate function. Given a function $f(V_D, V_B)$ we define the two bounds as the formula $abscond_{f,V_D}(V_B)$ and the expression $abspos_{f,V_D}(V_B)$. Finally, we employ them to define the $\{bound_x\}_{x \in V_D}$ and formula A for a set of predicates $\mathcal{P}$ such that all equations (8.1), (8.2) and (8.3) hold.

The absolute positiveness of a multivariate function f is an upper bound for all the roots of the function. Therefore, after such bound the function is always positive or always negative. For simplicity of presentation we will consider only the positive case.

Definition 26 - Absolute Positiveness
A function $f(V)$ is absolutely positive from $B \in \mathbb{R}^{|V|}$ iff $f(B) > 0$ and every non-zero partial derivative of every order of f is positive in $\bigwedge_{x \in V} x \geq B[x]$.

Notice that absolute positiveness of f from B implies that $\forall V : (\bigwedge_{x \in V} x \geq B[x]) \rightarrow f(V) > 0$, since all its derivatives are non-negative.

An approach for the computation of absolute positiveness of univariate polynomials is the Cauchy bound [46, 113]. Given a univariate polynomial $f(x) \doteq x^n + \sum_{i=0}^{n-1} a_i x^i$ with $a_i \in \mathbb{R}$ for all i, its Cauchy bound is $1 + max(\{|a_i|\}_{i=0}^{n-1})$, hence the function is absolutely positive for all $x > 1 + max(\{|a_i|\}_{i=0}^{n-1})$. In our setting we are interested in the case where such bounds can depend on some symbols V_B, hence the coefficients a_i are not simply constants but expressions over the symbols in V_B. The bound can introduce some assumptions on the coefficients, hence constraints over V_B. For example, given the univariate polynomial $f(x) \doteq \sum_{i=0}^{n} a_i x^i$, where $x \in V_D$ and each a_i is an expression over V_B we can compute the Cauchy bound by identifying the non-zero coefficient of the x with largest exponent:

$$abspos_{f,V_D} \doteq \begin{cases} a_n(V_B) \neq 0 & \text{then } 1 + max(\{|\frac{a_i}{a_n}|\}_{i=0}^{n-1}); \\ \quad \vdots & \\ a_m(V_B) \neq 0 \wedge \bigwedge_{i=m+1}^{n} a_i(V_B) = 0 & \text{then } 1 + max(\{|\frac{a_i}{a_m}|\}_{i=0}^{m-1}); \\ \quad \vdots & \\ a_0(V_B) \neq 0 \wedge \bigwedge_{i=1}^{n} a_i(V_B) = 0 & \text{then } 1. \end{cases}$$

Notice that if every coefficient is equal to 0 the bound is undefined. For this reason we introduce $abscond_{f,V_D}$, a formula over the symbols in V_B

representing the region in which the bound $abspos_{f,V_D}$ applies. In our example, we can define $abscond_{f,V_D}(V_B)$ as $\bigvee_{i=0}^{n} a_i(V_B) \neq 0$.

In the literature, there are many procedures to overapproximate the absolute positiveness of univariate and multivariate functions. They require different assumptions on the coefficients, provide bounds with different precision and generate expressions with different complexities. A comparative discussion of these techniques is out of the scope of this thesis and we simply assume that we are able to define the operators *abspos* and *abscond*. The interested reader can refer to [113, 2, 145, 110, 146], for a more detailed discussion on the computation of absolute positiveness bounds.

More formally, given the sets of symbols V, $V_D \subseteq V$ and $V_B \doteq V \setminus V_D$ and an expression $f(V_D, V_B)$, we define $abscond_{f,V_D}(V_B)$ as the formula over V_B describing the region in which $abspos_{f,V_D}(V_B)$ overapproximates the absolute positiveness bound for f with respect to V_D. Notice that both $abscond_{f,V_D}$ and $abspos_{f,V_D}$ depend only on the symbols V_B.

We will now define how the set of symbols V_D and predicates $\mathcal{P}$ can be extracted from a transition system and we employ *abspos* and *abscond* to define the set of bounds $\{bound_x\}$ for each $x \in V_D$ and the formula A for the assumptions over V_B, where $V_B \subseteq V \setminus V_D$ is the set of symbols not in V_D that appear in some predicate in $\mathcal{P}$.

Given a transition system $M \doteq \langle V, I, T \rangle$ we extract the set of predicates $\mathcal{P}$ by identifying a subset of the symbols $V_D \subseteq V$ such that all atomic predicates in T containing symbols in $V_D \cup V'_D$ are of the form $f(V) \bowtie 0$ or $x' \bowtie g(V)$, where $x' \in V'_D$, $\bowtie \in \{<, \leq, >, \geq\}$ and f and g are polynomials over the symbols in V. Similarly, we could also enumerate candidate loops of M, as in Alg. 2, and identify V_D as above but relative to the predicates in the candidate loop instead of the whole transition relation T.

We partition $\mathcal{P}$ in the following sets:

$$\begin{aligned}\textsc{Curr} &\doteq \{f(V) \bowtie 0 \mid \ \bowtie \in \{>, \geq\} \wedge f(V) \bowtie 0 \in \mathcal{P}\};\\ \mathrm{U}_x &\doteq \{u(V) \mid x' < u(V) \in \mathcal{P} \vee x' \leq u(V) \in \mathcal{P}\};\\ \mathrm{L}_x &\doteq \{l(V) \mid x' > l(V) \in \mathcal{P} \vee x' \geq l(V) \in \mathcal{P}\}.\end{aligned}$$

Then, for every $x \in V_D$ we define $bound_x(V_B)$ as the maximum of the absolute positiveness of every $f \in \textsc{Curr}$ and of every difference $u - l$ for $u \in \mathrm{Us}_x \cup \mathrm{Ue}_x$ and $l \in \mathrm{Ls}_x \cup \mathrm{Le}_x$. More formally,

$$bound_x(V_B) \doteq max(abspos_{\textsc{Curr}} \cup abspos_{\mathrm{U}} \cup abspos_{\textsc{Diffs}} \cup abspos_{\textsc{Diffs}_I});$$

where:

$$\begin{aligned}abspos_{\textsc{Curr}} &\doteq \{abspos_{f,V_D}(V_B) \mid f(V_D, V_B) \bowtie 0 \in \textsc{Curr}\};\\ abspos_{\mathrm{U}} &\doteq \{abspos_{u,V_D}(V_B) \mid u(V) \in \mathrm{U}_x\};\\ abspos_{\textsc{Diffs}} &\doteq \{abspos_{u-l,V_D}(V_B) \mid x \in V_D \wedge \neg Int(x) \wedge u(V) \in \mathrm{U}_x \wedge l(V) \in \mathrm{L}_x\};\\ abspos_{\textsc{Diffs}_I} &\doteq \{abspos_{\lfloor u \rfloor - \lceil l \rceil,V_D}(V_B) \mid x \in V_D \wedge Int(x) \wedge u(V) \in \mathrm{U}_x \wedge l(V) \in \mathrm{L}_x\}.\end{aligned}$$

Finally, A is defined as the conjunction of all assumptions required for all expressions:

$$A(V_B) \doteq abscond_{\textsc{Curr}} \wedge abscond_{\mathrm{U}} \wedge abscond_{\textsc{Diffs}} \wedge abscond_{\textsc{Diffs}_I};$$

where:

$$\begin{aligned}abscond_{\textsc{Curr}} &\doteq \bigwedge_{f \bowtie 0 \in \textsc{Curr}} abscond_{f,V_D};\\ abscond_{\mathrm{U}} &\doteq \bigwedge_{u \in \mathrm{U}_x} abscond_{u,V_D};\\ abscond_{\textsc{Diffs}} &\doteq \bigwedge_{u \in \mathrm{U}_x} \bigwedge_{l \in \mathrm{L}_x} abscond_{u-l,V_D};\\ abscond_{\textsc{Diffs}_I} &\doteq \bigwedge_{u \in \mathrm{U}_x} \bigwedge_{l \in \mathrm{L}_x} abscond_{\lfloor u \rfloor - \lceil l \rceil,V_D}.\end{aligned}$$

Theorem 18 - Synthesis approach is sound
Given the set of predicates $\mathcal{P}$ over symbols $V_D \cup V_B$. A and $\{bound_x\}_{x \in V_D}$ identified by the procedure above satisfy all equations (8.1), (8.2) *and* (8.3).

Proof. We prove the equations one at a time.

- Consider first Eq. (8.1).
  R requires every $x \in V_D$ to be greater than its corresponding $bound_x$ for every V_B. $bound_x$ requires x to be in the absolute positive region of each function f. By definition of absolute positiveness, for every such x the sign of f is constant. Therefore, Eq. (8.1) holds.
- We now consider Eq. (8.2).
  For every $x \in V_D$, for every pair of expressions l and u representing respectively a lower and upper bound for x, $bound_x$ requires x to be in the absolute positive region of $u - l$, and also of $\lfloor u \rfloor - \lceil l \rceil$ if x has integer domain. This implies that their difference is positive, hence u must be greater than l for every state in R, hence Eq. (8.2) holds.
- Finally consider Eq. (8.3).
  For every $x \in V_D$, for every upper bound u for x, $bound_x$ requires x to be in the absolute positive region of u. Therefore, in R every partial derivative of every order of u must be positive. Therefore, for increasing values of any $x \in V_D$ also the value of u must increase. Therefore, $u(bound_{V_D}(V_B), V_B)$ is the minimum and Eq. (8.3) holds.

□

8.2.3 Example

Consider the fair transition system $M \doteq \langle V, I, T, F \rangle$, defined over the variables $V \doteq \{b, x\}$ such that b is Boolean and x has domain $\mathbb{Z}$. Let $I \doteq \top$, $F \doteq b$ and $T \doteq x^3 - 5x^2 + 2 > 0 \wedge x' > 20x^2 \wedge x' < \frac{x^3}{40} \wedge (b \rightarrow \neg b') \wedge (b' \rightarrow x' > x)$. We define $V_D \doteq \{x\}$ and $\mathcal{P}$ as the set of predicates in M containing x:

$\mathcal{P}\doteq\{x^3 - 5x^2 + 2, x' > 20x^2, x' < \frac{x^3}{40}, x' > x\}$. Then, $\mathcal{P}$ is partitioned into the 3 sets as follows:

$$\begin{aligned} \textsc{Curr} &\doteq \{x^3 - 5x^2 + 2 > 0\}; \\ \mathrm{U}_x &\doteq \{\frac{x^3}{40}\}; \\ \mathrm{L}_x &\doteq \{x, 20x^2\}. \end{aligned}$$

We now employ the Cauchy bound to compute the absolute positiveness of these univariate polynomials. The computation is straightforward in most cases, we detail only the one involving the difference between the upper bound $\frac{x^3}{40}$ and the lower bound $20x^2$. In this case we need to compute the absolute positiveness of $\lfloor\frac{x^3}{40}\rfloor - \lceil 20x^2\rceil$. Since x has integer values, then $20x^2$ must be an integer and $\lceil 20x^2\rceil = 20x^2$. Instead, $\frac{x^3}{40}$ is not guaranteed to be an integer. However, since we are only interested in the absolute positiveness of the expression, we can approximate it by considering $\frac{x^3}{40} - 20x^2 - 1$. Therefore, we removed the "floor" operator by subtracting 1 to the expression. By construction the previous expression is always greater than or equal to the new one, hence the absolute positiveness of the latter implies absolute positiveness of the former. Finally, we can employ the Cauchy bound and the absolute positiveness bound is $1 + max(20 \cdot 40) = 801$. A similar reasoning allows the computation of all other bounds and we obtain:

$$\begin{aligned} abspos_{\textsc{Curr}} &\doteq \{1 + max(5, 2)\} = \{6\}; \\ abspos_{\mathrm{U}} &\doteq \{1\}; \\ abspos_{\textsc{Diffs}_I} &\doteq \{41, 801\}. \end{aligned}$$

We can now compute $bound_x$ as the maximum of all these values, hence

$$bound_x \doteq 801.$$

Notice that we did not require any assumption on the other symbols, hence $A \doteq \top$. Therefore, we now have all the elements required to define the E-comp H^x.

H^x is depicted in Fig. 8.1. It has a single region $R_0^x \doteq x \geq 801$ and trivial ranking function $\text{RF} \doteq \mathbf{0}$. In order to define its assumptions, we first need to define min_u and max_b that represent the two bounds on the expressions in $\text{US}_x \doteq \text{U}_x$ and $bound_{V_D}$ at $x = bound_x$. We can easily compute these values by substitution and obtain $max_b \doteq bound_x \doteq 801$ and $min_u \doteq \frac{801^3}{40} - 1$. Therefore, the assumptions of H^x are given by the formula $A_0^x \doteq \top$, since all constraints are comparisons between constants and are always true. Finally, the transition relation is

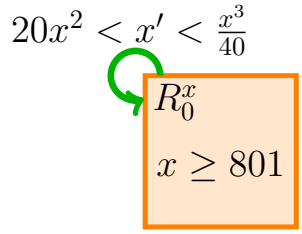

Figure 8.1: E-comp H^x.

$$T^x \doteq x' < \frac{x^3}{40} \wedge x' > x \wedge x' > 20x^2 \wedge$$
$$(x \geq 801 \rightarrow (x' \geq 801 \rightarrow \lfloor \frac{801^3}{40} \rfloor - 1 \geq \lceil 801 \rceil)).$$

The formula $\lfloor \frac{801^3}{40} \rfloor - 1 \geq \lceil 801 \rceil$ always holds, hence we can simplify the relation and obtain: $T^x \doteq x' < \frac{x^3}{40} \wedge x' > x \wedge x' > 20x^2$. Finally, in the region $x \geq 801$, which implies $20x^2 > x$. This allows us to further simplify it as: $T^x \doteq 20x^2 < x' < \frac{x^3}{40}$.

H^x by itself is not sufficient to prove the existence of a fair path in $\mathcal{L}(M)$. For this reason, we define an E-comp H^b responsible for b. H^b has two regions, $R_0^b \doteq b$ and $R_1^b \doteq \neg b$, transition relation $T^b \doteq \neg b'$ and trivial assumptions and ranking functions. The composition $H^{x,b}$ of H^x and H^b proving the existence of a fair path for M is depicted in Fig. 8.2.

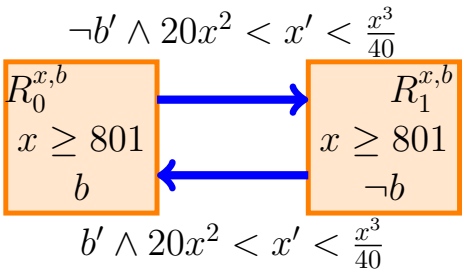

Figure 8.2: E-comp $H^{x,b}$.

Chapter 9

Related Work

This chapter describes current state-of-the-art approaches for the falsification of temporal properties and its sub-problems. Most of the literature in verification of temporal properties of infinite-state and timed transition systems focuses on the universal case (i.e. proving that all traces satisfy a property), while the existential one has received relatively little attention. In addition, most approaches focus on a specific context, for example software nontermination, consider decidable subproblems or identify only lasso-shaped witnesses.

We classify the techniques based on the kind of modelling and specifications languages they consider.

9.1 Term Rewriting Nontermination

Term rewriting systems (TRS) are formal models describing systems that operate on *terms*, an example of TRS is the well-known λ-calculus. In TRS a term (or expression) is a tree-structure over some symbols and each subtree is called subterm or subexpression. A TRS defines a set of *rewriting rules*. A rewriting rule $l \to r$ can be applied to a term s if l matches some subterm of s. The result of the application of the rule is a new term in which a subterm matching l is replaced with r in s. We call execution

of a TRS the application of a possibly infinite sequence of rewritings to some initial term. An execution of a TRS terminates when there are no applicable rules and a TRS is terminating if all its executions terminate.

Many techniques identify nonterminating executions by looking for applications of rules that describe a loop [92, 96, 143, 163]. The work [83], instead, focuses on non-looping nontermination. The approach proposes to manipulate the rules of the TRS by applying a set of operators and generating pattern rules. The space of pattern rules is then explored to identify a rule $l \to r$ such that r admits some subterm that matches l.

However, TRS do not have a notion of initial states nor a built-in notion of fairness ([137] describes a reduction from fair termination to termination). Finally, the profound differences in the model of computation between TRS and transition systems hinders the applicability of approaches developed in one context into the other.

9.2 Software Nontermination

The halting problem is a well-known and studied undecidable problem. Its complement is the problem of identifying whether a program admits at least one infinite run. This decision problem is a particular instance of LTL model checking in infinite-state systems. In fact, a software program can be modelled as a infinite-state system. For example, it is possible to define a transition system where all deadlock states correspond to states in which the program terminates. Therefore, the halting problem can be encoded as the search for an infinite path for the transition system i.e. deciding the language emptiness for a fair transition system where all states are fair.

Some tools, such as AProVe [94], transform the software program into a TRS and try to prove (non)termination of the program by proving (non)termination of the corresponding TRS. TRS, with respect to tran-

sition systems, allow for a more natural modelling of procedure calls and recursive functions.

Other approaches try to prove that a program is nonterminating by identifying a reachable open/closed recurrent set for the program. These procedures search for a recurrent set by relying on quantifier elimination, SMT-solvers or forward/backward analysis of the software program, sometimes combined with abstraction. Open recurrent sets have been introduced in [100]. In this work the control flow graph of the software program is exploited to enumerate candidate lassos and each candidate is analysed to check whether it admits a recurrent set. The search for the recurrent set is performed by generating a sequence of SMT problems each of which corresponds to a recurrent set template, i.e. a formula with some parameters that need to be defined such that the resulting formula corresponds to a recurrent set. However, in this work they assume the program to be fully deterministic and without deadlocks.

Other approaches, such as [66, 49, 131, 17, 18, 87], search for a closed recurrent set. In [131] and [66] the search for a recurrent set is reduced to a sequence of MAX-SMT and SMT queries respectively. The first one uses MAX-SMT to prune as many terminating branches as possible, while the second adopts an approach similar to the one of [100], but the template represents a closed recurrent set and it is also capable of dealing with nonlinearities by approximating them. However, [66] considers only simple loop programs over the integers without deadlocks and [131] requires the knowledge of the control flow graph in order to prune paths. The work [49] takes a different approach and reduces the search for a closed recurrent set to a sequence of safety queries. It progressively computes an underapproximation of the program by identifying and removing terminating executions, until either the program becomes empty or it proves nontermination. Given a terminating execution it performs a backward

analysis to identify a condition that caused the execution to terminate and the program is refined by adding the negation of such condition.

In [17] and [18] the search for a recurrent set is performed in an abstract space. The approach of [17] employs forward analysis. It starts from the initial conditions of the software loop and it symbolically propagates them forward following the control flow graph (performing the case splits on branching conditions) and progressively removing the terminating paths until a formula representing a closed recurrent set is obtained. Instead [18] relies on (over)approximated backward analysis to identify candidates via trace partitioning. Then the approach performs forward analysis on the candidate trying to refine it into a closed recurrent set.

The works [87] and [48] employ transformations on the software program before searching for a recurrent set. The technique described in [87] tries to simplify the software representation to obtain a simple loop program by learning invariants and *chaining*. Simple loops are then *accelerated*, the approach computes a closed form representing k iterations of the loop. Nontermination is then detected by checking whether the guard of the loop is an invariant or if the loop has a fixpoint. Notice that a reachable fixpoint for the loop corresponds to a lasso-shaped nonterminating execution. Instead, in [48], the program is *reversed* obtaining a transition system that starts from the end states and moves backward. The terminating executions of the original program correspond to paths that reach the initial states in the reversed transition system. Therefore, all initial states that correspond to a terminating run must satisfy the invariants of the transition system, called backward invariants. The nondeterminism in the program is resolved using symbolic polynomial assignments and nontermination is then detected by identifying backward invariants whose complement is reachable.

Finally, other approaches look for specific classes of programs or specific

nontermination arguments. [88] and [115] prove the decidability of termination for linear loops over the integers, while in [134] nontermination is seen as the sum of geometric series.

However, all of these works deal with nontermination and do not support the verification/falsification of temporal properties; fairness constraints are not considered and they rely on the existence of a control flow graph. Instead, we work at the level of transition system and support full LTL.

9.3 LTL Falsification on ITS

In the context of model checking LTL specifications on infinite-state systems, as discussed in §2.7.3, a simple visit or fixpoint computation will not be able to conclude that a property holds and a model might not admit any lasso-shaped counterexample. However, it is still safe to conclude that the property is violated if such a counterexample is found. BMC and L2S can be employed in this context as sound procedures to identify lasso-shaped counterexamples. However, if no such counterexample exists they cannot provide an answer.

The work [69] reduces the verification of the universal fragment of CTL on a infinite-state transition system to the problem of deciding whether a program always returns true. The approach can be applied also on LTL properties by relying on a reduction based on prophecy variables and it relies on some off-the-shelf tool for the analysis of the program. Therefore, its capability of proving or identifying a counterexample for some property depends on the ones of the considered underlying tool.

The work [68] explicitly deals with fairness for infinite-state programs supporting full CTL*. The technique presented in it is able to deal with existential properties and provides fair paths as witnesses. The approach focuses on programs manipulating integer variables, with an explicit control-

flow graph, rather than more general symbolic transition systems expressed over different theories. Another approach supporting full CTL* is proposed in [121]. The work presents a model checking algorithm for the verification of CTL* on finite-state systems and a deductive proof system for CTL* on infinite-state systems. In the first case the authors reduce the verification of CTL* properties to the verification of properties without temporal operators and a single fair path quantifier in front of the formula. To the best of our knowledge there is no generalisation of this algorithm, first reported in [122] and then also in [123], to the infinite-state setting. The rules presented in the second case have been exploited in [28] to implement a procedure for the verification of CTL properties, while our objective is the falsification of LTL properties.

9.4 LTL Verification on TA

LTL verification on timed automata is a decidable problem. Timed automata allow for finite abstractions that preserve both reachability and language emptiness. Tools, such as CTAV [135], DiVinE [19], MITLBMC [125] and LTSmin-Opaal [119] support LTL verification on timed automata by relying on the generation of a finite zone abstraction that preserves not only reachability but also language emptiness. *k-extrapolation* [157] and *LU-abstraction* [22] are two examples of zone-based abstractions that preserve both reachability and emptiness of timed automata. Most tools (e.g. CTAV, DiVinE and LTSmin-Opaal) rely on an explicit state representation of the locations of the automaton while the timing dimension and clock constraints are represented symbolically using specialised data-structures called *Difference Bound Matrices*. Other tools (e.g. MITLBMC), rely on a fully symbolic representation of the system. In [125] well-known algorithms for symbolic model checking of finite-state systems are specialised

for the verification of LTL and (a fragment of) MTL specifications on timed automata. The specialisation consists of a symbolic encoding of the region abstraction for timed automata.

Tools and approaches in this context differ in how they deal with Zeno paths. Tools, such as CTAV, simply require the input model to not contain Zeno paths. Other approaches check whether from all reachable states it is possible to make a time elapse of at least 1 time unit [156]. Another possibility, adopted for example by UPPAAL [21], DIVINE [19] and LTSMIN-OPAAL [119], is to disregard all Zeno paths during the verification procedure. They compose the input model with a monitor automaton that changes location every c time units. The progress of time is ensured by requiring that the monitor automaton visits each location infinitely often.

However, the verification problem on TA is decidable hence less relevant for the purposes of this thesis.

9.5 LTL Verification on HS

Most of the work on hybrid systems is concerned with the reachability problem. The aim is to prove that the system is safe by proving that some events cannot happen. The reachability problem for hybrid automata is undecidable [108]. For this reason many sub-classes have been introduced in the attempt to clearly identify the boundary of decidability with respect to mixed continuous and discrete systems. In [108] Henzinger et alia prove that the invariant checking problem on initialised rectangular hybrid automata, timed automata included, is PSPACE-complete. In addition, they show that relaxing the initialisation constraint or allowing comparisons between continuous variables leads to undecidability. Other results on the decidability of different problems in this context are presented in [24, 12]. We simply highlight that very small extensions to the language of ini-

tialised rectangular hybrid automata are sufficient to make the verification of invariant properties undecidable. For example, the class of linear hybrid automata is semi-decidable [108].

Unfortunately, the decidable fragments are not expressive enough to model many real-world systems [4]. In many cases the expressiveness of rectangular hybrid automata does not allow for a sufficiently precise modelling of the real-world system [106]. For this reason many model checkers for hybrid systems allow for the specification of linear hybrid automata. In linear hybrid automata it is possible to represent the continuous state symbolically as a polyhedra and the reachable states can be computed using operations on polyhedra. This technique was implemented in HyTech [107]. Since then, the research effort has moved towards a more expressive class of hybrid systems allowing the specification of the continuous dynamics using ODE. In this setting a single polyhedra is not sufficient to represent the continuous dynamics (e.g. $\dot{x} = x$ describes an exponential dynamic). It is possible to over-approximate the flow using a piecewise linear envelope. This approach, called *flowpipe* approximation, was first proposed in [50], refined in [11] and still proves to be competitive as shown by more recent tools such as PHAVerLite [20]. Other works experimented with different representations for the sets of reachable states leading, for example, to techniques based on zonotopes [97] and support functions [99]. Some tools, for example CORA [3], allow the combination of different representations, such as zonotopes, zonotopes bundles, polytopes, taylor models and set of vertices. In order to handle complex nonlinear dynamics, other model checkers, such as FLOW*, exploit the flowpipe construction on the taylor expansion of the continuous dynamics. Tools based on the over-approximating flowpipe construction are unable to falsify specifications. Ariadne [23] proposes to solve this problem by employing both over and under approximations, allowing it to both verify and refute a specification.

In more recent years, also SMT-based techniques have been applied in this context. Some examples are HyCOMP [58], HYDLOGIC [117], HySat [85] and dReach [126].

However, there is a lack of support for the verification of more expressive specification languages such as LTL and MTL on infinite traces.

HyCOMP [58] is a SMT-based tool for the verification of invariants and LTL properties on hybrid systems. The hybrid system is translated into a corresponding infinite-state transition system and the verification task is solved using algorithms developed in this context. It supports full LTL and removes Zeno paths from the model by computing the product with a monitor automaton as other tools in the context of timed automata. However, HyCOMP is capable of deciding that a LTL specification does not hold only if it finds a lasso-shaped counterexamples.

Other techniques, search for a counterexample by posing the falsification problem as an optimisation of a robustness function [141, 9, 150, 162]. These works consider an expressive temporal specification language (e.g. MTL), however they are interested in bounded-time verification, hence do not consider infinite traces of the system.

Part IV

Tools and Experimental Evaluation

Chapter 10

Tools: F3 and nuXmv

The techniques described in Part II of this thesis have been implemented in the nuXmv model checker. In addition, we have extended the falsification capability of nuXmv for timed transition systems. In this context, we extended its BMC encoding and spuriousness checks of its IC3-based algorithm by considering the composition of the E-comp defined as described in Sec. 8.2 with one in which all variables perform a concrete lasso. The falsification approaches described in Part III are available in an open-source prototype we call Find-Fair-Funnel, abbreviated as F3. This chapter first describes the main features and architecture of F3 in Sec. 10.1. Then, Sec. 10.2 introduces nuXmv and describes how we extended it to support the verification of $\mathrm{MTL}_{0,\infty}$ and LTL-EF on timed systems and also how its model checking algorithms have been enhanced to include the falsification techniques described in this thesis.

10.1 Find-Fair-Funnel

We have implemented the falsification procedures, detailed in Sec. 7.2, in a prototype called F3[1], written in Python. F3 takes as input a transition system, a possibly empty set of E-comps and, if provided with a fairness

[1]the tool and the benchmarks can be downloaded from `https://github.com/enmag/F3`

condition it tries to identify a fair path for the given model, otherwise if an LTL specification is given as input it tries to identify a counterexample for such property. In addition, it is possible to specify a list of symbols whose assignments must correspond to a diverging sequence; this enables us to use F3 to identify non-Zeno fair paths in timed systems.

10.1.1 Architecture

F3 consists of about 10000 lines of Python code and 1500 lines of C++ code. F3 uses MathSAT5 [59] and Z3 [76] as underlying SMT engines, interacting with them through PYSMT [89]. PYSMT provides all the basic functionalities to represent formulae, symbols with their types and supports various operations over the formulae. The C++ code, instead, has been taken from [73] and implements the tableau construction necessary to support LTL specifications.

F3 is divided into several modules; in the following we describe the main components and highlight some relevant implementation details.

- *Multisolver* wraps the solver interface of PYSMT and, in particular, the SMT-SOLVE method. SMT-solvers sometimes take a very long time on a single query, for this reason the module associates a timeout to each call. The module has a sequence of solvers, it submits each query to each solver in order until one succeeds in providing an answer within the given time. If no solver is able to provide such answer, F3 assumes unknown as result and continues. This procedure could be optimised by submitting the query to different solvers in parallel, for simplicity of implementation we do not exploit parallelism and only one solver is called at a time.

- *Runner* is the entry module. It parses the command line arguments and retrieves the inputs for F3. Each input is a Python source file

implementing specific methods that return the transition system, diverging symbols, set of E-comps and a fairness condition or LTL specification. The runner employs reflection to identify which methods are implemented by each input. It relies on the *LTL* and *diverging encoder* modules to reduce each input problem to finding a fair path of a transition system.

- *LTL* implements the tableau construction for LTL specifications and also the structure used to represent LTL formulae. The implementation of this module is an adaptation of the one in the artefact of [73].
- *Diverging encoder* implements the product construction described in [57] to remove all Zeno-paths of the model.
- *BMC* implements the Bounded Model Checking algorithm to identify candidate fair loops of the transition system.
- *Implicant* implements three procedures to compute model-based implicants of a formula. Given a formula and an assignment satisfying it, they compute another formula satisfied by the given assignment and that implies the input formula. The first procedure, returns a subset of the predicates appearing in the formula, satisfied by the given assignment, whose conjunction implies the formula. It relies only on Boolean reasoning and all theory predicates are abstracted as Boolean atoms. The second procedure computes the implicant by computing the unsat-cores of the negation of the formula in conjunction with the truth assignment of the predicates in the formula given by the satisfying assignment. This approach also considers the theories of the atoms and asks the SMT-solver to compute the unsat-cores until a fixpoint is reached. The last procedure relies on the interpolants computed by SMT-solvers instead of the unsat-cores. By default, F3 first computes an implicant using the Boolean reasoning of the first

approach, then the second approach is used to remove other redundant predicates based on theory reasoning.

- *EF-solver* and *Motkin*. F3 solves the parameter synthesis problem described in Sec. 7.2 via a combination of the EF-SMT procedure of [80] and the application of Motzkin's transposition theorem [139]. By default, F3 first tries to apply EF-SMT and resorts to the elimination of universal quantifiers only if this fails to provide a definite answer. Motzkin's transposition transforms the synthesis problem into a purely existentially-quantified one which can be solved via standard quantifier-free SMT reasoning. However, it requires the predicates to be affine, hence F3 replaces non-linear terms with fresh symbols, in order to obtain an affine system. This simple way of handling non-linearities has the benefit of being very easy to implement; however, it can produce coarse approximations, which can prevent F3 from finding counterexamples in cases where non-linearities play a significant role. A possible approach to handle non-linearities in a more precise manner is described in [10].

- *Hint* module provides the data structure used to represent E-comps.

- *Funnel* and *Floop* modules implement the representation and operations of funnels and funnel-loops. The Floop module also defines the methods that generate the funnel-loop templates given the candidate loop. By default, F3 considers a minimum of 0 and a maximum of 2 inequalities in the implementation of HEURISTIC-PICK-NUM-INEQS of Alg. 3. It considers only simple ranking functions corresponding to the PR-ranking template described in [133], i.e. simple affine linear functions. In addition, we only synthesise predicates in the form of affine linear equalities or inequalities; the implementation of the function NEW-PARAMETRIC-EXPR in F3 generates affine linear expressions.

F3 implements two template generators that differ in how they choose when to create a ranking function for a funnel (line 11 of Alg. 3). One generator matches the candidate loop sequence of regions and transition with a regular expression. The other generator simply looks for a pair of abstract states that agree on the assignment to the Boolean symbols of the transition system. If the generator decides not to introduce a ranking function for a funnel, its *rf* is simply set to the constant **0**. This avoids the introduction of unnecessary parameters for funnels that do not need an explicit ranking function.

10.2 nuXmv

nuXmv is a well-known symbolic model checker. It is the successor of the open-source NuSMV and supports model checking of invariants and LTL specification on infinite-state transition systems specified in the SMV language. We extended nuXmv with a new software module implementing the reduction techniques described in Chapter 4. This extension has been released as version 2 of the model checker. nuXmv2 supports verification of $\mathrm{MTL}_{0,\infty}$ and LTL-EF on models with dense or super-dense time model. In addition, it supports model simulation and also load and re-execution of traces. These functionalities ease the inspection of the model to ensure it correctly represents the system.

In the following we detail the architecture of nuXmv2, extending the one of nuXmv for the verification of timed systems.

10.2.1 Architecture

Fig. 10.1 depicts the high level architecture of nuXmv extended with the new module to handle timed transition systems (nuXmv2). The new version of nuXmv preserves full backward compatibility, apart from some

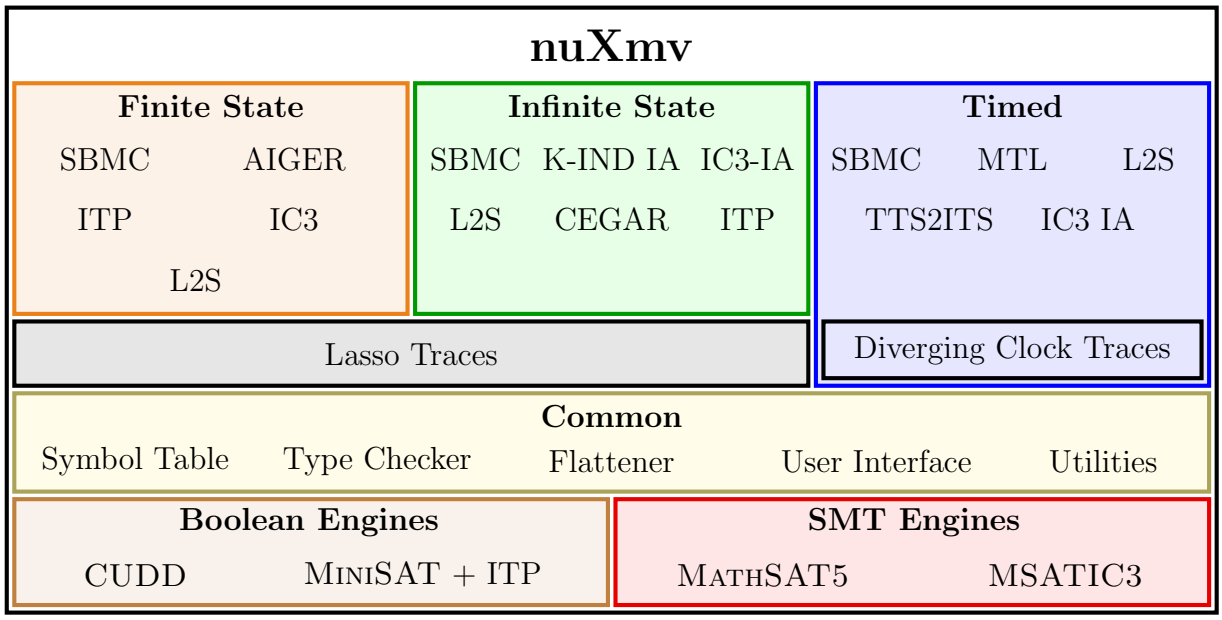

Figure 10.1: The high level architecture of nuXmv.

new reserved keywords, with respect to its previous versions [47]. Indeed, it shares with the previous version all the basic support functionalities, such as the symbol table, the flattening of the design, the Boolean encoding of scalar variables and the representation of the finite-state machines at the different abstraction levels (e.g. scalar, BDD). This implies that it supports all the basic model checking algorithms for finite domains using both BDDs (via CUDD [153]) and SAT (e.g. via MiniSAT [81]). For infinite domains it supports various SMT-based model checking procedures. These procedures are implemented via a tight integration with the MathSAT5 solver [59], some examples are SBMC [132], IC3 [40, 103, 160], k-liveness [62] and liveness-to-safety [151].

In the following we detail how nuXmv2 extends its previous version described in [47] to support the specification and model checking of invariant, LTL and $\text{MTL}_{0,\infty}$ properties for timed transitions systems, and for the validity checking of properties over dense and super-dense time semantics.

```
1  @TIME_DOMAIN continuous  -- annotation to specify the time semantics, in this case dense time.
2
3  MODULE main
4   FROZENVAR
5    p: real; -- parameter.
6   VAR
7    i: real;  -- input of the sensor.
8    s: Sensor(i);
9    m: Monitor(s.o, p);
10
11  INIT p > 0;  -- parameter must be positive.
12
13  LTLSPEC G ( s.fault -> F [0, p] m.alarm )  -- any fault is detected in p timed units.
14
15 MODULE Sensor(i)
16  VAR
17   o: real;
18   fault: boolean;
19
20  TRANS !fault -> next(o) = i  -- if not faulty, the sensor provides in output directly the input.
21  TRANS fault -> next(o) = o  -- if faulty, the sensor output is stuck at the last value.
22  TRANS fault -> next(fault)  -- the fault is permanent.
23
24 MODULE Monitor (i,p)
25  VAR
26   previous_value: real;
27   c: clock;
28   alarm: boolean;
29
30  INIT c = 0 & previous_value = i & !alarm;
31  INVAR TRUE -> c <= p;
32  TRANS (c = p & next(c) = 0 & next(previous_value) = i) |  -- monitor reads the sensor every p time units.
33        (c <= p & next(c) = c & next(previous_value) = previous_value);
34  -- alarm raised when the same value read twice consecutively.
35  TRANS next(alarm) <-> (alarm | i = previous_value);
```

Figure 10.2: A simple nuXmv2 model.

- The parser has been extended and now the user can choose the time semantics to use for the read model. Depending on the time model some parse constructs and checks are enabled and/or disabled. For instance, variables of type clock are allowed only if the time semantics is not "discrete". By default, the system uses the discrete time semantics of the original nuXmv. Notice also that, depending on the specified semantics, the commands available to the user change to allow only the analyses supported for the chosen semantics.

- The parser now supports the specification of timed transition systems, via the definition of clock variables, the specification of urgent transitions and location invariants. In addition, it allows for the specification of $\text{MTL}_{0,\infty}$ properties and the LTL bounded operators now

can also contain complex expressions over clock variables. Fig. 10.2 reports a simple example showing some of the new language constructs.

- The symbol table now supports variables of type clock and we extended the type checker to properly handle the newly defined variables, expression types and language constructs.
- A new *TTS2ITS* module encodes the model checking problems on timed transition systems (TTS) into equivalent model checking problems on infinite-state transition systems with discrete time model (ITS). These problems can then be addressed using the existing algorithms of nuXmv. The module performs the translation of the model as in [60] and the specifications are discretized as described in Sec. 4.3.
- The trace representation of nuXmv has been extended to support timed traces where some clock variables may diverge.
- We modified the encoding for the loops in the bounded model checking algorithms to take into account that traces may contain diverging variables to allow for the verification and validation of LTL and $\mathrm{MTL}_{0,\infty}$ properties. This corresponds to a BMC encoding of the composition of an E-comp with diverging symbols, defined in Sec. 8.2, and one whose variables perform a lasso, hence the assignments to the variables keep repeating in the same order.

Chapter 11

Experimental Evaluation

This chapter reports the experimental evaluation of the approaches described in this thesis.

Sec. 11.1 describes the benchmarks we considered in the experiments. The state-of-the-art tools used to compare the effectiveness of our approaches are presented in Sec. 11.2. The same section also details to which benchmarks they have been applied based on the modelling and specification languages they support. We then organise the results we obtained along two directions. First, Sec. 11.3 discusses the results relative to the evaluation of the reduction-based approach for the verification of TTS that we detailed in Chapter 4. Then, in Sec. 11.4 we evaluate the falsification procedure described in Sec 7.2.

11.1 Benchmarks

The benchmarks are organised in 6 different categories based on their context of origin: LS, NS, ITS, TA, TTS and HS[1].

LS consists of 52 nonterminating linear software benchmarks taken from the C programs of the software termination competition [95]. We considered programs from the *C* and *C Integer* categories of the compe-

[1]all benchmarks can be downloaded from `https://github.com/enmag/F3`

tition that can be easily encoded as transition systems with Boolean, integer and real state variables. Therefore, we did not consider programs involving recursion or dynamic memory allocation.

NS contains 30 nonlinear software programs. 29 of them have been taken from [66]. We defined the remaining one such that it includes a variable to the power of 3 and a division by a constant. These are two elements not present in any of the other 29 benchmarks.

ITS encompasses 70 LTL falsification problems on infinite-state transition systems. 2 of these problems are proof obligations generated in the verification of a contract-based design, 29 come from scaling up to 30 processes a model of the bakery mutual exclusion protocol, other 29 instances are the scaling up to 30 process of a semaphore-based synchronisation protocol. The last 10 are instances we created and describe small systems that involve nondeterministic unbounded counters. We defined these models to test the capability of tools to deal with nondeterminism and identify infinite executions where the repeating pattern has non-constant length.

TA contains 174 timed automata. The models describe 6 different protocols, each of which has been scaled from 1 to 30 processes. The protocols are the *critical*, *csma*, *fddi*, *fischer*, *lynch* and *train* protocols from [84]. For every protocol we defined one true and one false property for each of the following specification languages: invariant, LTL and $\mathrm{MTL}_{0,\infty}$.

TTS consists of 120 LTL falsification problems on timed transition systems. 116 of them come from the scaling from 1 to 30 processes of 4 protocols (inspired by the *csma*, *fischer*, *lynch* and *token ring* protocols), and 4 are handcrafted instances. We now describe how we modified the 4 protocols and remark that the resulting systems can-

not be modelled as timed automata.

- The extended *csma* protocol introduces an adaptive backoff time for each process that increases every time a station encounters a collision and decreases each time it successfully communicates the whole message.
- We extended the *fischer* and *lynch* protocols by allowing each process to propose a wait time. Each process, before entering the critical section, must wait for at least the maximum of the proposed values.
- In the model of the *token ring* protocol we added a stopwatch that keeps track of the total amount of time spent while transmitting. In these models the LTL specification requires to verify whether the total transmission time is bounded by 10 subject to a fairness assumption on the token manager of the protocol.

These extensions require the comparison between a clock and an infinite-state variable, hence cannot be represented as TA. In fact, TA do not allow for infinite-state variables and clocks can be compared only with constants. Finally, we created the 4 remaining instances as follows. The first TTS is an abstracted version of the extended csma protocol. The system describes a single station and communication bus, while all other components have been abstracted as a non-deterministic environment. The second system is a sender-receiver protocol with acknowledgement of the messages based on a unique message identifier. The third model represents a modified stopwatch where the clock variable has a nonlinear reset. Finally, the last TTS describes a clock variable that is never reset and a nondeterministic state variable that must always be greater than the clock.

HS are 9 LTL falsification problems on hybrid systems. 5 of them have

been taken from the ARCH competition [86] and represent an adaptive cruise control system, a controller for the track permission of a train, a model of the flow of a liquid through some tanks and pipes and 2 different properties on a system describing the synchronisation of machines communicating via a ethernet network. The remaining 4 HS models describe a bouncing ball and they differ in the behaviour of the bounce. These models are meant to test the capability of the tools to handle the dynamic system described by the law of motion of the ball and a single discrete step used to represent the bounce.

11.2 Reference Tools

The state-of-the-art tools we use as reference for the evaluation of nuXmv and F3 are: Anant [66], AProVe [94], ATMOC [124, 125], CTAV [136, 135], DiVinE3 [104], iRankFinder [79], LTSmin [119], T2 [43], Ultimate [105] and Uppaal [75]. Unfortunately we could not obtain the software described in [28] to solve E-CHC problems.

We selected AProVe, iRankFinder and Ultimate because they are the tools that achieved the best results in the *C* and *C Integer* categories of the termination competition 2021[2]. We considered Anant because it implements techniques specialised to deal with nonlinear programs and we took most of the NS benchmarks from the work that describes them [66]. Finally, for software programs we considered also T2 which is a widely known tool for software analysis.

For the analysis of TA we selected a collection of tools we believe is representative of the techniques available in the state-of-the-art. Uppaal is a widely known model checker for timed automata and is used as a reference implementation by many works in this field. LTSmin at its core is a

[2]the results are available at https://termcomp.github.io/Y2021/

toolset for the manipulation of labelled transition systems. The toolset has many extensions that address different verification problems by relying on its *Partitioned Next-State Interface* (PINS). In this work we are interested in its *opaal* front-end that supports LTL verification on timed automata. We considered also version 3 of DiVinE as an additional tool supporting LTL verification on timed automata via region and zone abstraction. Unfortunately, the tool does not support the analysis of timed automata since version 4. CTAV is yet another tool that relies on simulating abstractions for timed automata, but supports also MTL specifications. To the best of our knowledge, CTAV is the only other tool capable of proving MTL properties on timed automata. Finally, ATMOC employs symbolic techniques closer to the ones we implemented in nuXmv and it is also capable of falsifying MTL specifications. The similarity of the approaches enables us to better understand the cost of the additional expressive power of our approach.

In the case of nuXmv we consider two different algorithms. Both algorithms were already available in the tool and, thanks to the reduction presented in Chapter 4, can now be applied to MTL verification problems on TTS. In addition, we enhanced their capability to identify counterexamples to temporal properties on timed systems (TA, TTS, HS) via the approach described in Sec. 8.2. Notice that this implies that in the LS, NS and ITS cases nuXmv can identify only lasso shaped counterexamples. We will refer to the first algorithm as nuXmv-IC3. This procedure is based on IC3 [40] and has been presented in [57] and [73]. nuXmv-IC3 is capable of both verifying and falsifying a specification. The second algorithm is based on bounded model checking [31] and will refer to it as nuXmv-BMC. nuXmv-BMC cannot conclude that a property holds, but only identify counterexamples.

Most of the tools we considered in our experiments are not able to handle

all the benchmarks. Therefore, we limit their application as described in the following and summarised in Table 11.1.

- We ran Anant, AProVe, iRankFinder and T2 only on the software nontermination problems (LS and NS groups).
- We ran ATMOC, CTAV, DiVinE3, LTSmin and Uppaal only on the timed automata (TA) benchmarks. CTAV, DiVinE3, LTSmin and Uppaal do not support MTL specifications, hence we considered them only for the invariant and LTL properties. LTSmin does not support clocks in the specifications, hence we could not use it to verify the true invariant property on the *csma* models. In addition, Uppaal supports only a fragment of LTL. In particular, it is not sufficient to express the false LTL properties of the *fischer* and *lynch* benchmarks and it supports the true LTL properties only of the *csma* benchmarks. For this reason, we could run it only on 29 [resp. 116] of the 174 TA instances for the true [resp. false] LTL specifications. Finally, ATMOC supports invariant, LTL and MTL specifications. However, it is capable of verifying only invariant properties, while in the LTL and MTL cases it supports only falsification via a BMC-based algorithm. Therefore, we run it on all the invariant verification and falsification problems and only on the false LTL and MTL specifications.
- Ultimate does not support nonlinear arithmetic, hence we could not run it on the NS family. In addition, since it supports LTL specifications but works on programs rather than transition systems, we translated the ITS benchmarks to LTL verification problems on software programs, using the same approach described in [73].
- We have not implemented the support for $\mathrm{MTL}_{0,\infty}$ specifications in F3 and it is only capable of falsifying properties. For this reason, we run it only on the LTL falsification problems.

- nuXmv is the only tool applicable to all the benchmarks. We executed nuXmv-IC3 on all benchmarks and nuXmv-BMC only on the falsification problems.

Model	Spec	Result	Num	Anant	AProVe	ATMOC	CTAV	DiVinE3	F3	iRankFinder	LTSmin	nuXmv-IC3	nuXmv-BMC	T2	Ultimate	Uppaal
LS	**Term**	**False**	52	✓	✓	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✗
NS	**Term**	**False**	30	✓	✓	✗	✗	✗	✓	✓	✗	✓	✓	✓	✗	✗
ITS	**LTL**	**False**	70	✗	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✓	✗
TA	**INV**	**False**	174	✗	✗	✓	✓	✓	✗	✗	✓	✓	✓	✗	✗	✓
TA	**INV**	**True**	174	✗	✗	✓	✓	✓	✗	✗	✓[1]	✓	✗	✗	✗	✓
TA	**LTL**	**False**	174	✗	✗	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	✓[2]
TA	**LTL**	**True**	174	✗	✗	✗	✓	✓	✗	✗	✓	✓	✗	✗	✗	✓[3]
TA	**MTL**	**False**	174	✗	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗
TA	**MTL**	**True**	174	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗
TTS	**LTL**	**False**	120	✗	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗
HS	**LTL**	**False**	9	✗	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗

1. LTSmin does not handle the invariant specification of the *csma* protocol.
2. Uppaal supports only the false LTL specifications of the *fischer* and *lynch* protocols.
3. Uppaal supports only the true LTL specification of the *csma* protocol.

Table 11.1: Applicability of tools to the different verification tasks.

Experimental setup. We executed each tool on a single benchmark at a time. Each test has been executed on a machine running Ubuntu 20.04 equipped with an Intel(R) Xeon(R) Gold 6226R 2.90 GHz CPU. We run each experiment using a 1 hour limit on its CPU time and a maximum resident set size of 30 GB.

Interpretation of the results. We summarise the results in survival plots. For each tool t, the plots show a point $\langle x, y \rangle$ iff t required x seconds to solve its y simplest instances, i.e. the y instances for which t required less time. In addition, we do not distinguish between nuXmv-IC3 and

nuXmv-BMC in the computation of the number of instances that have been solved by only one tool. We do this because the 2 algorithms employ the same procedure to identify counterexamples and we are interested in evaluating the effectiveness of this technique and not its integration in the algorithms.

11.3 Model Checking Timed Systems

In this section we evaluate the effectiveness of our reductions from $\text{MTL}_{0,\infty}$ and LTL model checking on timed systems to the verification of LTL specifications on infinite-state transition systems.

We consider verification of invariants, LTL and $\text{MTL}_{0,\infty}$ specifications on timed automata. The invariant verification problems are useful to evaluate the effectiveness of the reduction without the additional complexity of the temporal and metric operators of LTL and $\text{MTL}_{0,\infty}$. Then, we consider a set of LTL verification problems to assess the cost associated with the temporal (non-metric) operators and, finally, the $\text{MTL}_{0,\infty}$ model checking problems allow the evaluation of the effectiveness of the complete reduction.

The instances in which the temporal properties do not hold are useful to evaluate the effectiveness of the technique we described in Sec. 8.2 to identify diverging *clocks* that has been implemented in the nuXmv algorithms.

In all cases we compare against state-of-the-art tools in the context of timed automata, hence tools that rely on decision procedures based on region and/or zone abstraction. Our approach supports a more expressive, but undecidable, modelling language and does not rely on such specialised techniques. From this perspective the comparison on invariant and LTL specifications has the objective of understanding the cost of the additional

expressive power in the common fragment.

11.3.1 Results

Table 11.2 reports the number of instances solved for each category by each tool. In addition, in the subscript we report the number of instances uniquely solved by each tool, if any. We highlight that in 3 cases the

Spec	Result	Num	nuXmv-BMC	nuXmv-IC3	F3 (no hints)	ATMOC	CTAV	DiVinE3	LTSmin	Uppaal
INV	True	174	–	**144**$_{38}$	–	95$_{6}$	44	43	33[1]	82$_{1}$
INV	False	174	**142**	137	–	74	137	42	88	99
LTL	True	174	–	39$_{19}$	–	–	14	48	**61**$_{16}$	6[2]
LTL	False	174	156	90	140	151	148	71	**163**	116[3]
MTL	True	174	–	**57**$_{45}$	–	–	13$_{1}$	–	–	–
MTL	False	174	147	121	–	148	**152**$_{26}$	–	–	–

Entries marked with "–" denote that the tool cannot handle the given benchmarks.

1. LTSmin supports only 145 true invariant specifications.
2. Uppaal supports only 29 true LTL specifications.
3. Uppaal supports only 116 false LTL specifications.

Table 11.2: Number of solved verification problems on TA.

procedures implemented in nuXmv solved the highest number of instances among all the tools, and also in the other cases it seems to be competitive. The class in which nuXmv seems to be less competitive is the one requiring to prove LTL specifications where it solved 39 instances, while the best result, achieved by LTSmin, is 61. However, nuXmv-IC3 solved 19 unique instances and LTSmin 16. This indicates a high degree of complementarity between the two tools. In the case of the LTL falsification benchmarks, nuXmv-BMC is the second best tool after LTSmin. In this case nuXmv-BMC solved 156 instances and LTSmin 163, with ATMOC and CTAV closely matching this result with 151 and 148 instances solved respectively.

The other setting in which nuXmv does not achieve the best result is the falsification of MTL specifications. In this case nuXmv-BMC solves 147 instances, only 5 less than CTAV and 1 less than ATMOC.

We now inspect one class at a time and for each of them we report the survival plot summarising the results.

Invariant specifications

Fig. 11.1 reports the number of invariant specifications verified by each tool in the given amount of time. It can be observed that the two fully symbolic techniques, ATMOC and nuXmv-IC3, follow similar trends, with the algorithm of nuXmv being faster. Uppaal seems to be more effective than nuXmv-IC3 in solving from 20 to 60 instances (the distance between the two lines decreases) and they appear to solve the 60 simplest instances in roughly the same amount of time.

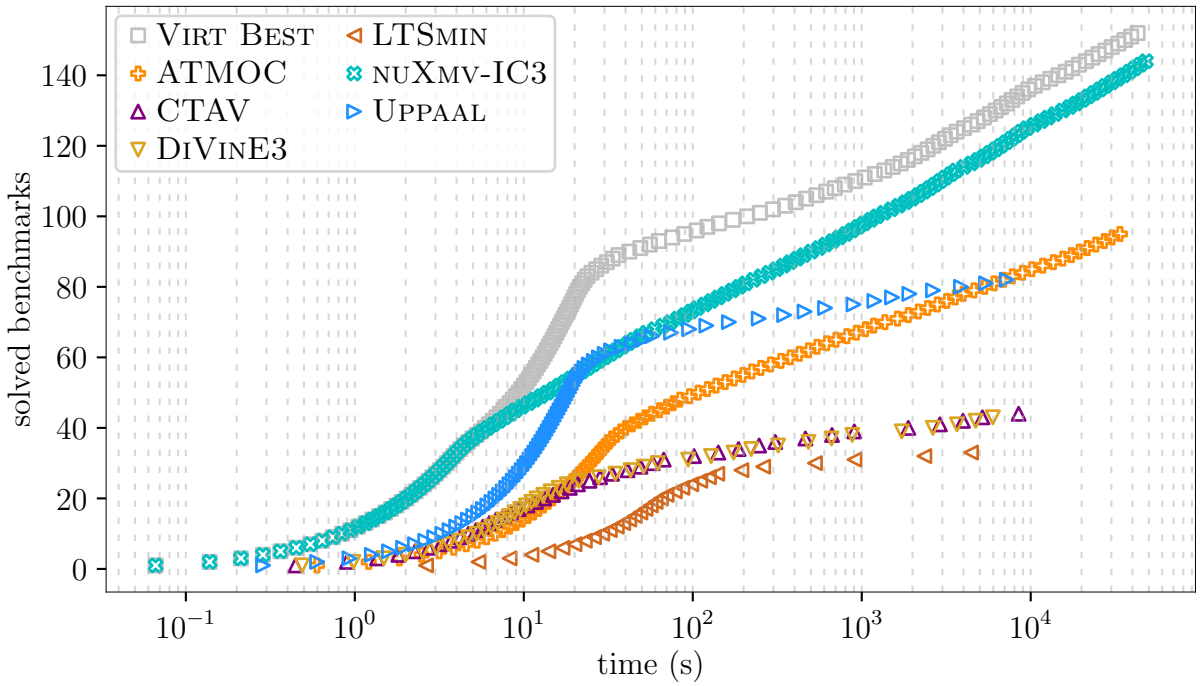

Figure 11.1: Survival plot, verification of true invariants on TA.

However, the performance of UPPAAL quickly degrades after that, while nuXMV-IC3 manages to solve many more instances and it appears to be relatively close to the virtual best after 110 problems. In addition, nuXMV proved 38 properties that all other tools failed to verify.

Fig. 11.2 shows the results we obtained in the falsification of invariant properties on the 174 timed automata. In the survival plot, nuXMV appears to be the fastest; its BMC algorithm is the quickest in solving up to about 110 instances and also solves the highest number overall. CTAV is faster than nuXMV-BMC in solving the last few problems, from 110 to 137. However, our approach solves 5 more instances. All the other tools stop before solving 100 instances. UPPAAL solves a total of 99 instances and it is faster than CTAV until they solve 80 problems. After that, as in the previous case, the performance of UPPAAL quickly degrades.

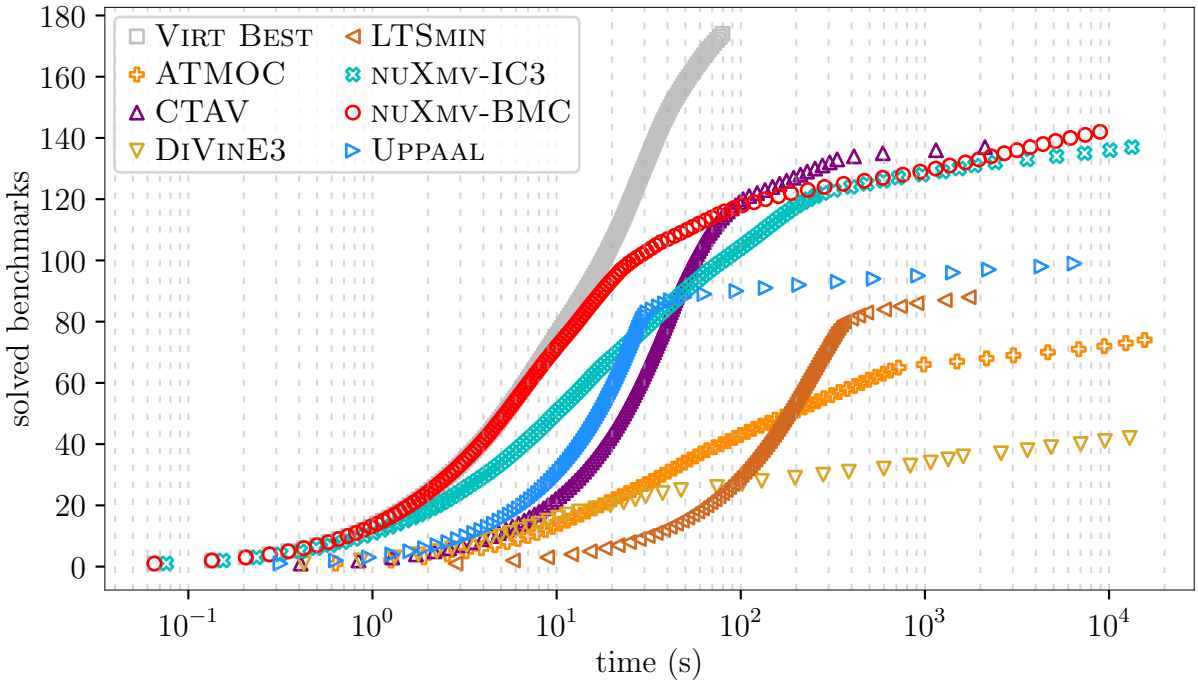

Figure 11.2: Survival plot, verification of false invariants on TA.

LTL specifications

The results we obtained in the verification of 174 true LTL specifications are reported in the survival plot of Fig. 11.3. The plot confirms our initial observation that nuXmv is not very effective in this setting. In this context both LTSmin and DiVinE3 appear to be capable of solving more instances and in less time. We remark that Uppaal has limited support for LTL specifications, hence we could run it only on 29 of the 174 instances. In this case nuXmv-IC3 was able to prove 19 properties that no other tool managed to verify, while LTSmin solved 16 unique instances.

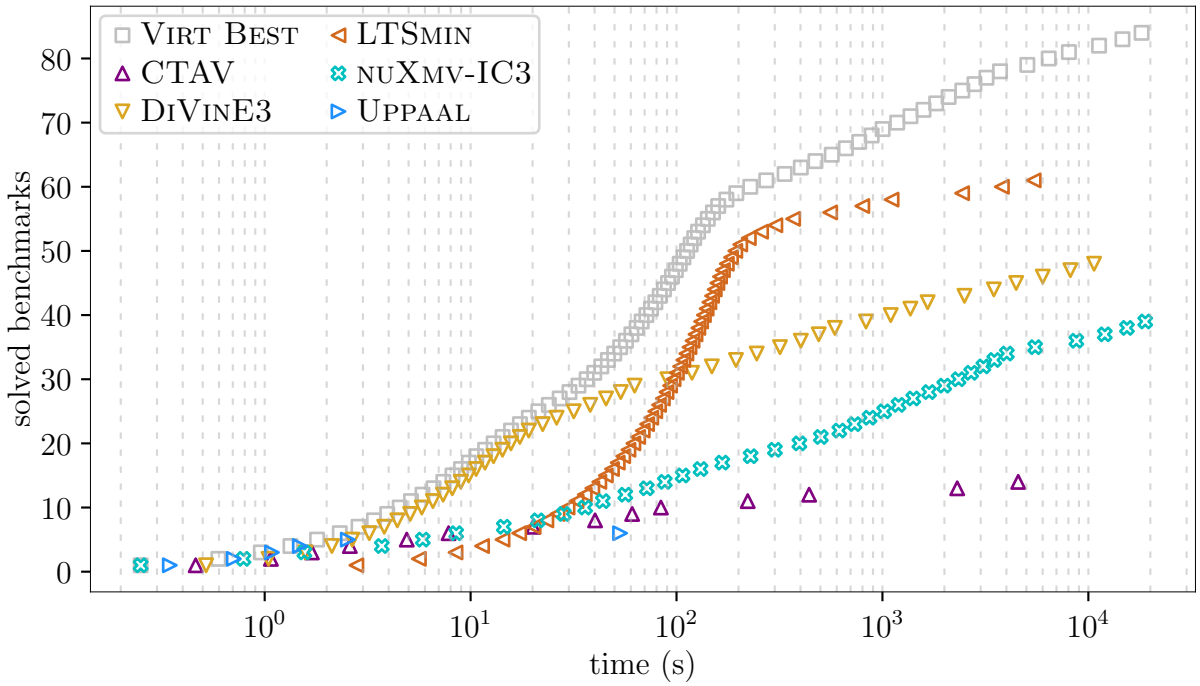

Figure 11.3: Survival plot, verification of true LTL on TA.

Fig. 11.4 shows the survival plot summarising the results obtained in the falsification of LTL specifications on timed automata. nuXmv-BMC appears to be the fastest up to about 100 instances, where its execution time starts to increase more steeply. Our approach is then overtook by Uppaal, CTAV and LTSmin in this order. However, it manages to solve

a higher number of instances than the first two tools, while LTSMIN solves 7 additional problems.

NUXMV-BMC and ATMOC show similar results. They solve a comparable number of instances, 156 and 151 respectively, and ATMOC appears to catch up to the execution time of NUXMV near the end. Both of them employ a fully symbolic technique based on BMC. However, while ATMOC employs an ad-hoc encoding that exploits the region abstraction for timed automata, both NUXMV-IC3 and NUXMV-BMC rely on a more generic encoding suitable also to timed transition systems. This might be the source of the advantage that ATMOC appears to have on the harder instances.

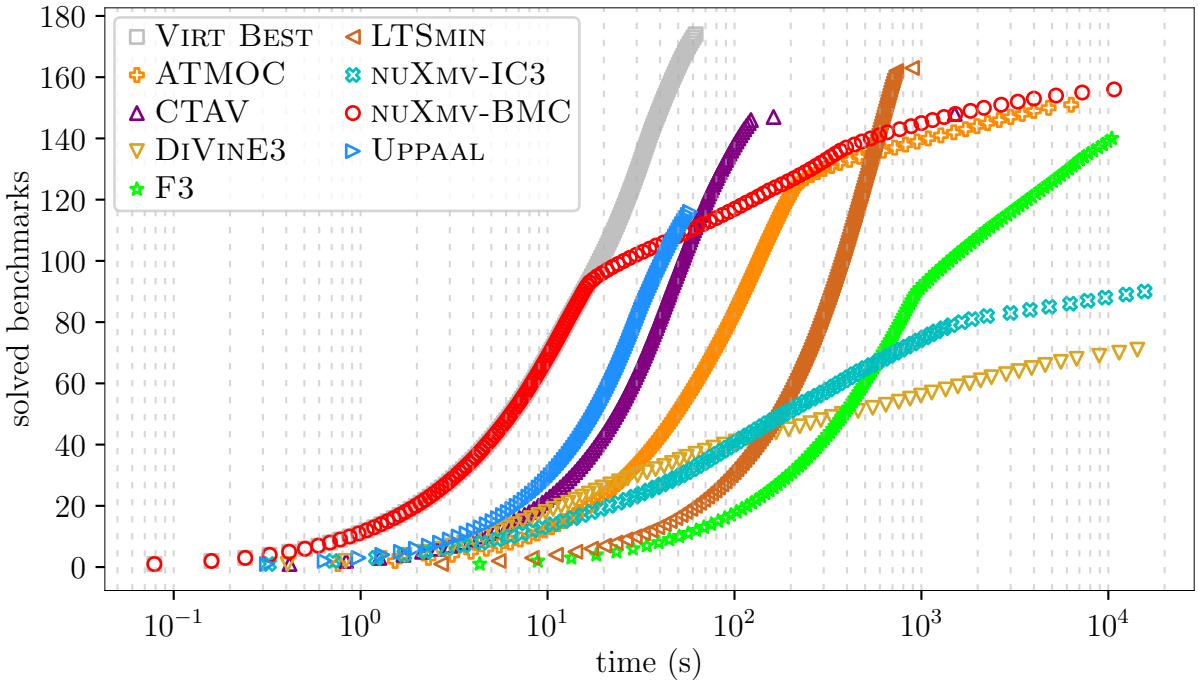

Figure 11.4: Survival plot, verification of false LTL on TA.

MTL specifications

Fig. 11.5 reports the results obtained in the verification of MTL specifications. The only two tools capable of handling such instances are CTAV

and nuXmv-IC3. CTAV appears to be very quick in solving the simplest instances. However, it is capable of solving only 13 in the allocated time, while nuXmv-IC3 solves 57. CTAV solved a single problem that nuXmv-IC3 failed to verify and nuXmv solved 45 unique instances. If we consider that there are 174 total instances, it is immediately apparent that the results are still not ideal. The two tools together managed to solve only one third (58) of the 174 problems. This could be a symptom of the inherent difficulty of verifying an expressive language such as MTL.

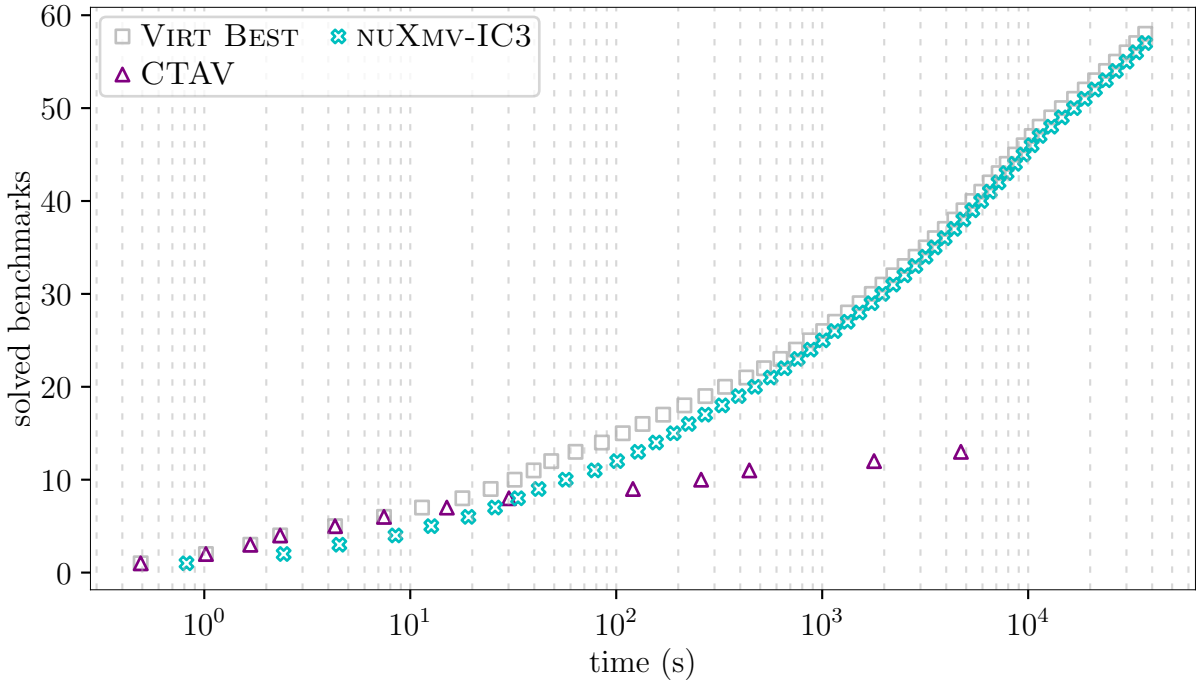

Figure 11.5: Survival plot, verification of true MTL on TA.

Finally, Fig. 11.6 considers the false MTL specifications. CTAV identified the highest number of counterexamples, followed by ATMOC. nuXmv-BMC solves only 1 instance less than ATMOC and appears to be particularly fast. It is interesting to observe the number of instances solved by nuXmv-IC3 in the true and false MTL cases. In the previous case the procedure managed to prove 57 MTL specifications, while in the falsifica-

tion case it solved 121 instances. Similarly, CTAV previously solved only 13 cases, while it was capable of falsifying 152 specifications. This seems to highlight that, for current model checking approaches, the verification of MTL specifications is much harder than the corresponding falsification problem.

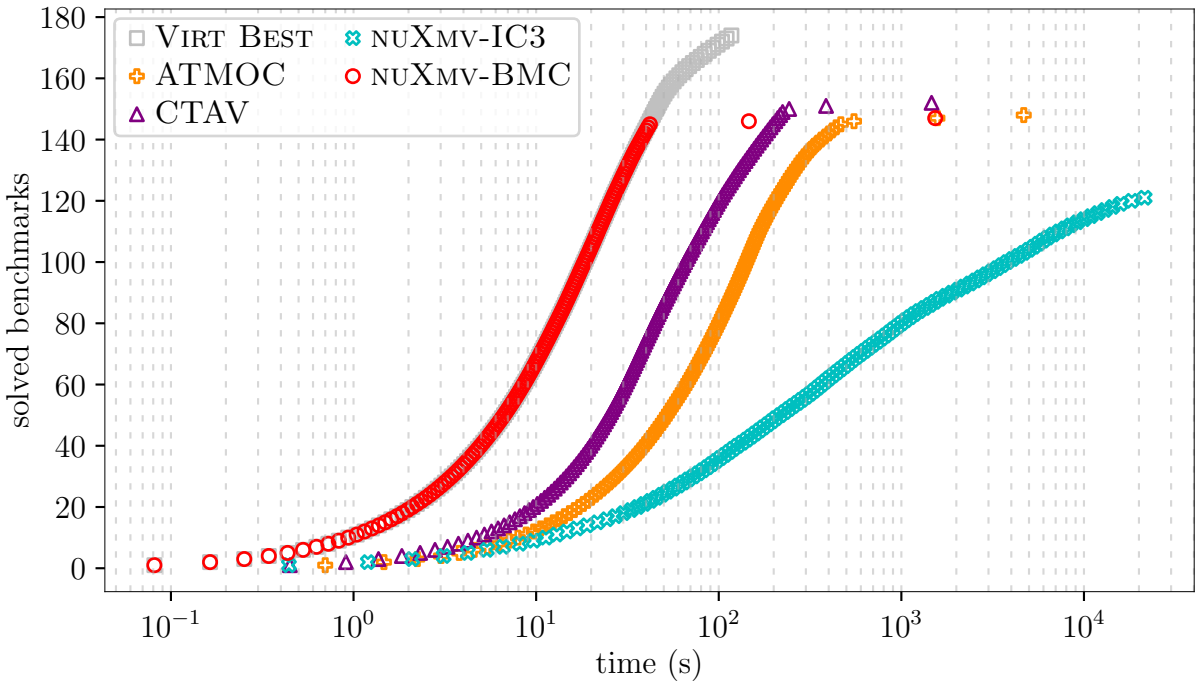

Figure 11.6: Survival plot, verification of false MTL on TA.

Concluding remarks

There is no clear winner on the 1044 model checking problems on timed automata. In the falsification instances nuXmv-BMC is the approach that identified the highest number of counterexamples in the invariant case, while also in the LTL and MTL cases it solved a competitive number of instances, respectively 7 and 5 instances less than the maximum.

Due to the similarities in their approach it is relevant to observe the difference in performance between nuXmv-BMC and ATMOC on the LTL and MTL falsification instances. They are the only fully symbolic

tools and both implement a BMC-based algorithm. However, ATMOC implements an ad-hoc BMC encoding for timed automata that exploits the region abstraction, while nuXmv-BMC employs a more general approach applicable to TTS. Our approach appears to be generally faster with ATMOC gaining on the hardest instances and identifying a few more counterexamples.

Consider now the true properties. In this case the results vary significantly depending on the specification language we consider. nuXmv-IC3 appears effective on invariant and MTL specifications. In both cases it solved the highest number of instances and it proved to be very competitive also in terms of the time required to solve them. However, it seems to struggle on the verification of LTL properties, where it is outperformed by both LTSmin and DiVinE3. In the case of invariant specifications, nuXmv does not incur in the additional complexity introduced by the reduction we presented in this work (Chapter 4). The generality of the rewriting comes into play when we consider temporal properties. This causes nuXmv-IC3 to explore a large space in order to prove such properties. It is relevant to notice that the only other tool that supports the verification of MTL properties (i.e. CTAV) also performed poorly in the verification of LTL specification. In both LTL and MTL verification benchmarks nuXmv-IC3 solved more instances in less time than CTAV.

These results prove the effectiveness of our approach, both in terms of the number of instances it solved and the time it required to do that. Our approach supports the model checking of invariant, LTL and MTL specifications on TTS. However, there is a lack of tools supporting this modelling formalism and, in our experiments, we considered benchmarks in the from of TA. Even in this restricted case, nuXmv and CTAV are the only tools capable of supporting the model checking of three types of specifications we considered. By considering TA, we compared nuXmv against tools

that are tailored to this verification scenario. Our approach still appears to be reasonably competitive in all settings. nuXmv is one of the fastest tools in all contexts apart from the verification of LTL specification. In addition, it showed to be the tool capable of solving the highest number of instances in 3 of the 6 settings we considered (verification and falsification of invariant specifications and verification of MTL properties).

11.4 Falsification

In this section we evaluate the effectiveness of the LTL falsification procedure implemented in F3. We consider a wide range of benchmarks: linear and nonlinear software programs, infinite-state transition systems, timed automata and hybrid systems. The objective of this evaluation is to assess the performance of the procedure across these contexts.

It is relevant to notice that many of the state-of-the-art tools are also capable of proving that a specification holds, while F3 can only identify counterexamples. In addition, tools for software analysis are often capable of analysing programs with recursive procedures and dynamic memory allocation. These features are specific to software programs and cannot be found in the other contexts, such as ITS, TA and HS. For this reason, we do not consider them in our evaluation and describe each software program via a corresponding infinite-state transition system using an explicit program counter *pc*. The (non)termination of the original program can be determined by verifying the LTL property $\mathbf{F}pc = end$ on the transition system, where *end* is the value assigned to the program counter upon completion of the procedure. Any counterexample for such property corresponds to a nonterminating execution for the program.

11.4.1 Results

Table 11.3 summarises the number of instances solved by each tool in the different contexts. In addition, in the subscript we report the number of instances uniquely solved by each tool, if any. F3 identified the highest number of counterexamples in 5 of the 6 contexts and it is the tool that solved the highest number of instances overall. In the table it is also possible to notice that F3 solved a significant number of instances that no other tool managed to address successfully.

Model	Num	F3 (no hints)	Anant	AProVe	ATMOC	CTAV	DiVinE3	iRankFinder	LTSmin	nuXmv-BMC	nuXmv-IC3	T2	Ultimate	Uppaal
LS	52	**52**	38	43	–	–	–	39	–	28	28	38	49	–
NS	30	$\mathbf{30}_{2}$	26	5	–	–	–	6	–	14	14	2	–	–
ITS	70	$\mathbf{65}_{57}$	–	–	–	–	–	–	–	4	4	–	8	–
TA	174	140	–	–	151	148	71	–	**164**	155	89	–	–	116[1]
TTS	120	$\mathbf{74}_{72}$	–	–	–	–	–	–	–	29_{27}	10	–	–	–
HS	9	$\mathbf{3}_{3}$	–	–	–	–	–	–	–	0	0	–	–	–
Total	455	**364**	64	48	151	148	71	45	164	230	145	40	56	116

Entries marked with "–" denote that the tool cannot handle the given benchmarks.

1. UPPAAL supports only 116 false LTL specifications.

Table 11.3: Number of falsified LTL specifications per benchmark family.

Linear software

Fig. 11.7 summarises, in a survival plot, the results we obtained on the linear software benchmarks. The figure shows that both algorithms of nuXmv quickly identify the cases in which there is a lasso-shaped non-terminating execution. However, they fail to solve the instances where no such execution exists. F3 is the only tool that solved all 52 instances and it is also the fourth fastest in solving up to 22 benchmarks and third fastest

afterwards. However, we have no way of measuring the time spent by the other tools in trying to prove termination. Therefore, we cannot draw any definite conclusion by comparing their execution times.

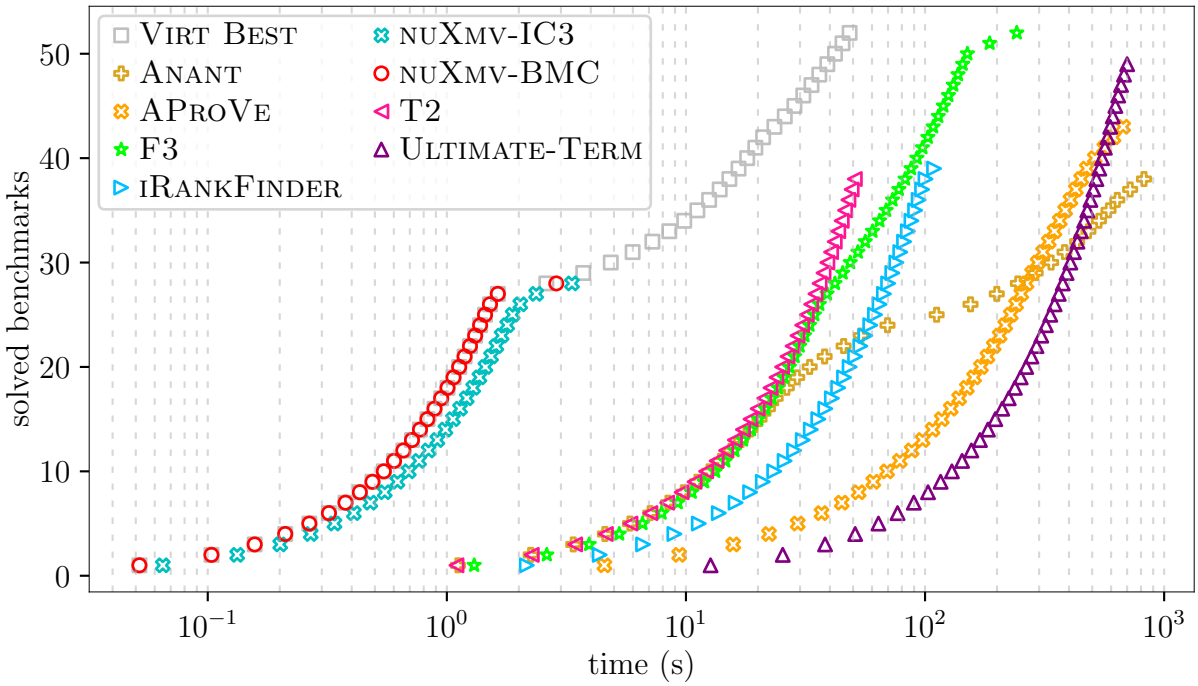

Figure 11.7: Survival plot, linear software nontermination.

Nonlinear software

Fig. 11.8 reports the results on the nonlinear software instances. Notice that all but one of the nonlinear software benchmarks come from the paper that first introduced Anant [66]. We remark that Anant, AProVe, iRankFinder, nuXmv-IC3 and T2 are all capable of both proving and disproving termination, while F3 and nuXmv-BMC can only conclude nontermination. As in the previous case, nuXmv-BMC quickly identified all nonterminating instances for which there exists a lasso-shaped witness. However, it managed to identify such a counterexample in only 14 of the 30 model checking problems. Anant and F3 show similar performance in solving their first 15 instances. After that up to the 20^{th} instance Anant

appears faster, but it takes a lot of time in solving its 21^{th} instance and it is overtook by F3. F3 is the only tool that solved all of the instances and is the only tool that managed to solve 2 of them.

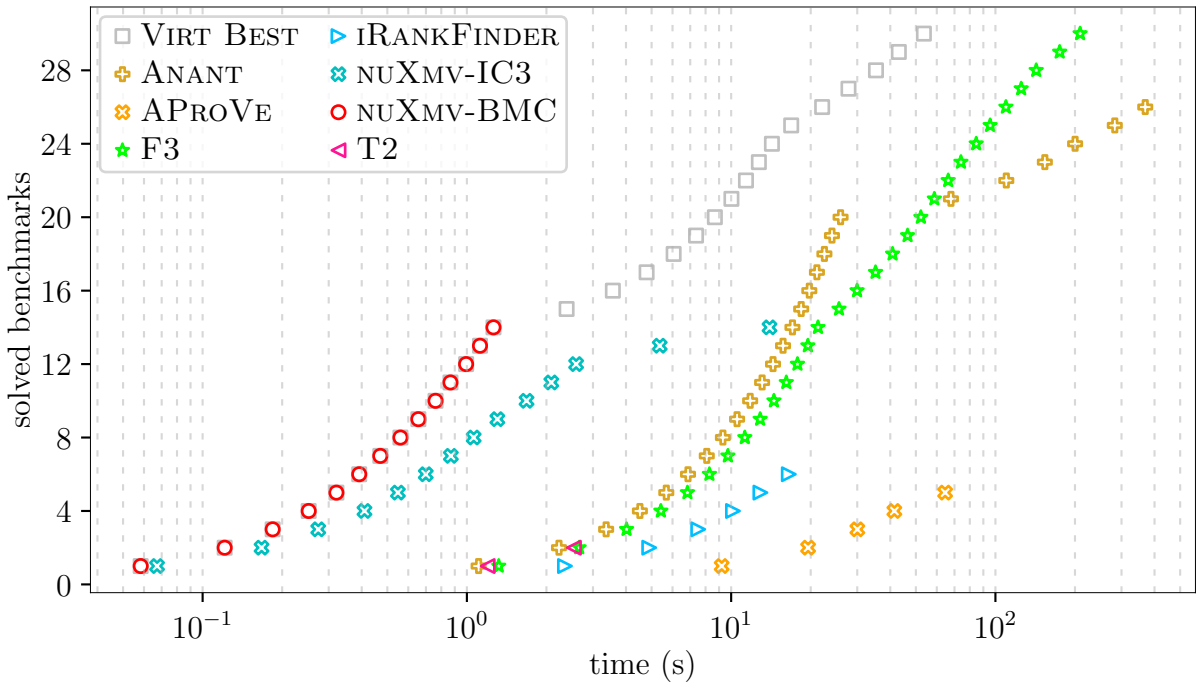

Figure 11.8: Survival plot, nonlinear software nontermination.

Infinite-state transition systems

Consider now the 70 LTL falsification problems on ITS. The results we obtained in this context are reported in Fig. 11.9. Also in this case, F3 solved the highest number of instances, while nuXmv-BMC identified only 4 counterexamples and Ultimate, considering both its LTL and termination configurations, solved only 8 instances.

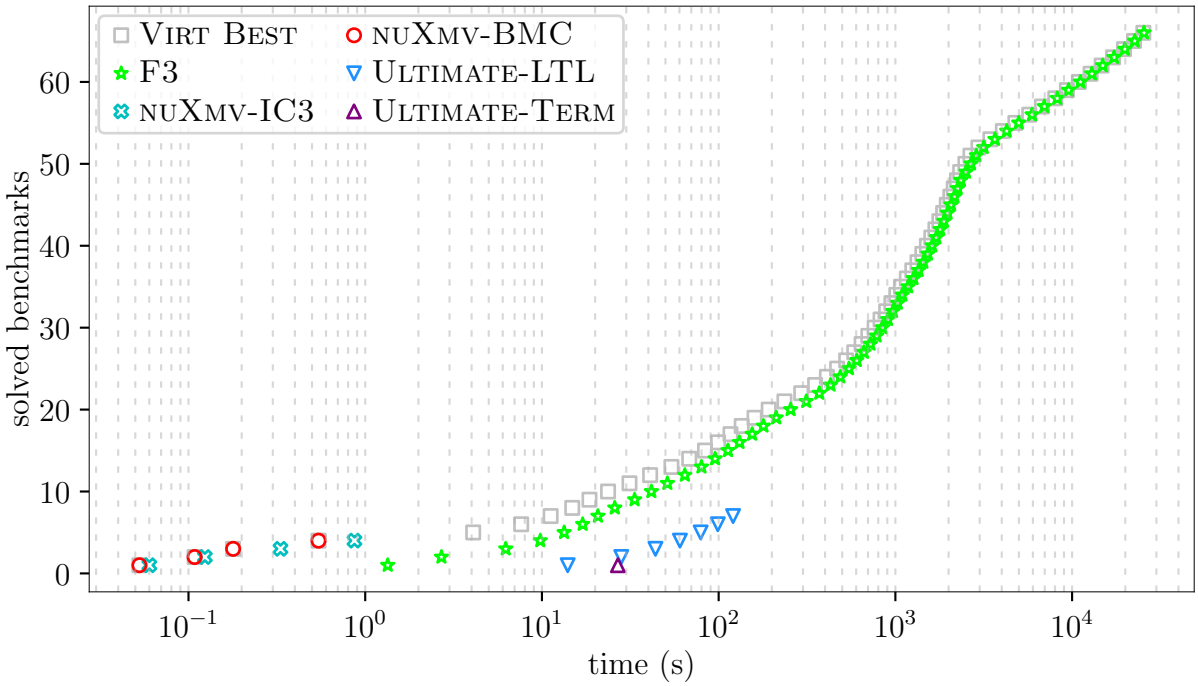

Figure 11.9: Survival plot, false LTL on ITS.

Timed automata

The results on the falsification of LTL specifications on TA are reported in Fig. 11.4. We already analysed such figure from the perspective of the approach we implemented in nuXmv-BMC. Now, instead, we consider the performance of F3 with respect to the other tools. The plot highlights that F3 is among the slowest tools in this context. This does not come as a surprise since most of the other tools have been specifically developed to analyse TA, while F3 implements a much more general technique. However, F3 solved a number of instances comparable to the other tools. It solved 140 problems outperforming DiVinE3, which solved 71, and getting relatively close to the highest value of 163, achieved by LTSmin.

Timed transition systems

Fig. 11.10 summarises the results we obtained on TTS. There is a limited number of tools that support verification of these models. The only two approaches that provided counterexamples for the properties we considered in our experiments are the ones we introduced in this thesis. Both managed to identify several counterexamples, but F3 appears to be both faster and capable of solving more instances. Notice that the virtual best solver solved 101 instances, while F3 stopped at 74 and nuXmv at 29. This indicates a high degree of complementarity. In fact, there are only 2 instances that have been solved by both techniques. The counterexample representation of F3 is strictly more expressive than the one used in nuXmv, nuXmv-BMC looks for counterexamples with a specific shape. This allows it to be much faster in exploring the space, while F3 requires more time to analyse and refute the candidates.

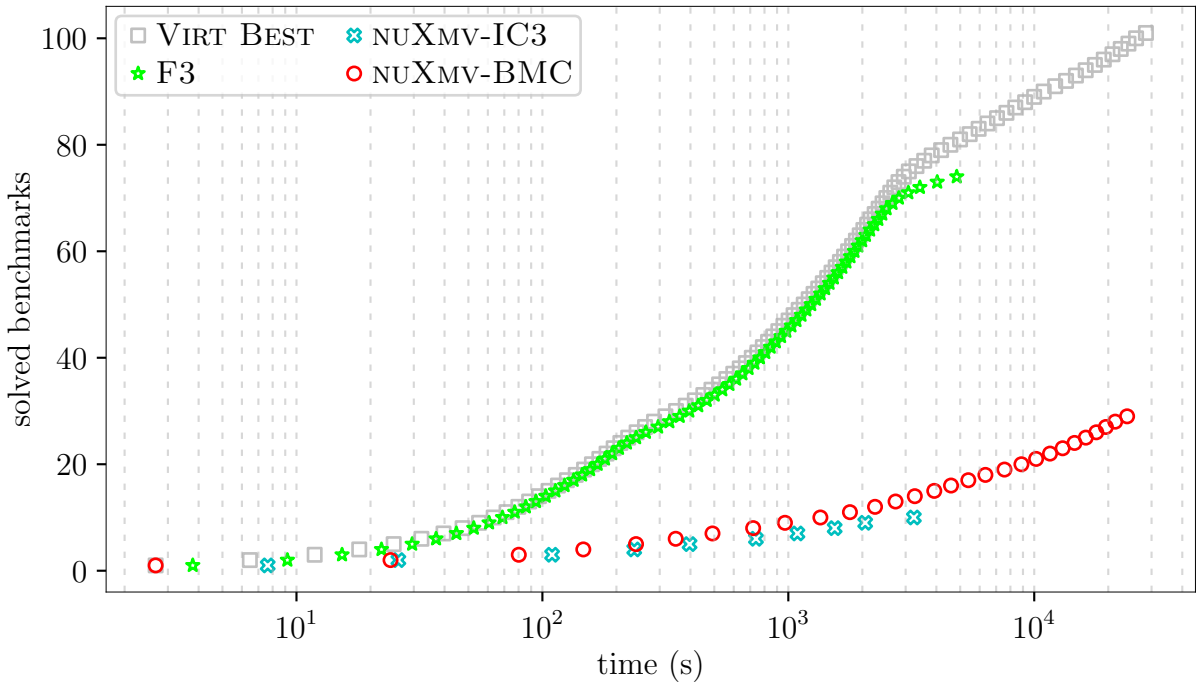

Figure 11.10: Survival plot, false LTL on TTS.

Hybrid system

We considered 9 LTL falsification problems on hybrid systems. Among all the models we considered these are the most difficult ones; they combine both nonlinear dynamics with timing constraints. F3 is the only tool that managed to solve at least one instance, and it solved only 3 in total. Up to now, in all our experiments we did not provide any hints to F3. In this case, we employed this capability and provided F3 with some user-defined hints in the form of E-comps. For each of the 6 unsolved instances we were able to define a single E-comp responsible for at most 2 symbols that allowed F3 to identify a counterexample. The definition of such hints was done manually and required the inspection of the model to identify the portion of the system that could be the hardest for F3 to automatically analyse. In most cases, we defined a hint that specifies the amount of time that needs to elapse at every transition. In our benchmarks, these hints greatly reduced the number of candidate loops generated by F3 and also effectively reduced the complexity of the formulae describing them by allowing the simplification of the nonlinear terms.

Concluding remarks

F3 proved to be effective in all the contexts we considered. In our experiments it solved a significant number of instances in every benchmark category. In 5 of the 6 categories we considered, F3 is the tool that solved the highest number of instances and, notably, it solved all instances in the two software categories. However, it is not the fastest when competing against ad-hoc techniques. F3 spends most of the time in the analysis of candidate loops. Therefore, the order in which the space of these candidates is explored greatly affects the overall execution time. Currently, the exploration is based on a BMC unrolling of the transition relation, hence

it favours shorter candidates and the space is explored in a breadth-first manner. This implies that F3 is less effective when the counterexample is deep and requires a long candidate. In this case F3 would have to analyse and discard many candidates before finding the desired one.

Chapter 12

Conclusions and Future Work

12.1 Conclusions

The complexity of systems developed by engineers has been steadily increasing over the years. As the system complexity grows it becomes harder to ensure its correct behaviour and the consistency of its design. Formal methods provide a number of mathematically precise languages to model, specify and reason about such systems. The development of the formal representation of the system is not an easy task and usually requires an iterative process involving successive refinements and inspections. Two aspects are of particular importance in tools that support this process. The first one is the expressiveness of their modelling language and the second is their capability of analysing and inspecting such models.

In this thesis, we first proposed a novel reduction-based approach to support the verification of expressive specification languages on timed systems. Then, we defined a compact and expressive structure to represent counterexamples. We employed this novel representation to develop a procedure applicable and effective in a wide range of contexts and also an approach tailored to identify counterexamples in a more specific context (timed transition systems). The experimental evaluation highlighted some degree of complementarity between the two approaches. The generality of

the first approach allows its application across different contexts and our experimental evaluation showed it to be competitive even against tools implementing context-specific algorithms. Although we were able to achieve good performance using only general assumptions on the verification problems, exploiting the prior information about the context is essential to achieve the best performance in the verification of a specific class of systems. Finally, our experimental evaluation demonstrates the relevance of the two novel approaches. They were able to solve 235 verification problems that none of the other tools we considered managed to address within the allocated resources.

12.2 Future work

This thesis opens several future research directions and further improvements to be explored.

The reduction based approach for the verification of MTL specifications defined in Chapter 4 is very general. The procedure can be tailored to generate simpler encodings in recurring cases. This could allow nuXmv to achieve better results in the verification of LTL specifications on timed automata. From the expressiveness point of view, the reduction could be extended to directly deal with the dynamical components described in terms of constraints over the derivatives with respect to time. This would enable the procedure to reason directly about the system of ordinary differential equations (ODEs) instead of relying on the existence of an explicit solution for the system.

The execution time of F3 is dominated by the analysis of the candidate loops it generates. Heuristics that allow the early pruning of such candidates have a great impact on the time required to identify a counterexample. The algorithm proposed in this thesis focuses on infinite be-

haviours generated by arithmetic operations over the reals and the integers and relies on a coarse abstraction of the nonlinear terms. The approach proved to be effective in many cases. However, we believe that improving the support for nonlinearities, for example by enabling the refinement of such abstraction, could lead to better results in contexts, such as hybrid systems, were they play a significant role. Furthermore, the approach could be extended to consider also fair paths whose infinite behaviour depends on other theories, for example strings, arrays and algebraic datatypes. In addition, as we move away from lasso-shaped counterexamples we are also loosing the capability of checking the correctness of the witness in polynomial time. In fact, checking the correctness of a funnel-loop requires to prove that all hypotheses of Th. 6 hold. Therefore, the procedures proposed in this thesis could be further extended to produce not only the final funnel-loop but also the proofs required to verify its correctness. Another direction for future work is to expand the ideas we employed in the design of F3 to tackle more general verification tasks defined as E-CHCs or some fragment of them. This would enable their use to solve many different existential problems in the domain of formal verification. Finally, the approach could be integrated with other techniques capable of proving a specification (and not only identifying a counterexample) and also with techniques such as the ones presented in [102] and [121, 67] to support the verification of branching-time logics, such as CTL and CTL*, on infinite-state systems.

Bibliography

[1] IEEE standard for system, software, and hardware verification and validation. *IEEE Std 1012-2016 (Revision of IEEE Std 1012-2012/ Incorporates IEEE Std 1012-2016/Cor1-2017)*, pages 1–260, 2017.

[2] Alkiviadis G. Akritas, Adam W. Strzebonski, and Panagiotis S. Vigklas. Implementations of a new theorem for computing bounds for positive roots of polynomials. *Computing*, 78(4):355–367, 2006.

[3] Matthias Althoff. An introduction to CORA 2015. In Goran Frehse and Matthias Althoff, editors, *1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2014, Berlin, Germany, April 14, 2014 / ARCH@CPSWeek 2015, Seattle, WA, USA, April 13, 2015*, volume 34 of *EPiC Series in Computing*, pages 120–151. EasyChair, 2015.

[4] Rajeev Alur. Formal verification of hybrid systems. In Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, editors, *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011*, pages 273–278. ACM, 2011.

[5] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.

[6] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.

[7] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1991.

[8] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.

[9] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6605 of *Lecture Notes in Computer Science*, pages 254–257. Springer, 2011.

[10] Ali Asadi, Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Mohammad Mahdavi. Polynomial reachability witnesses via stellensätze. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 20211*, pages 772–787. ACM, 2021.

[11] Eugene Asarin, Thao Dang, Oded Maler, and Olivier Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control, Third International Workshop, HSCC 2000, Pittsburgh, PA, USA, March 23-25, 2000, Proceedings*, volume 1790 of *Lecture Notes in Computer Science*, pages 20–31. Springer, 2000.

[12] Eugene Asarin, Venkatesh Mysore, Amir Pnueli, and Gerardo Schneider. Low dimensional hybrid systems - decidable, undecidable, don't know. *Inf. Comput.*, 211:138–159, 2012.

[13] Tomás Babiak, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. LTL to büchi automata translation: Fast and more deterministic. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7214 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2012.

[14] Roberto Bagnara, Fred Mesnard, Andrea Pescetti, and Enea Zaffanella. A new look at the automatic synthesis of linear ranking functions. *Inf. Comput.*, 215:47–67, 2012.

[15] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

[16] Christel Baier and Cesare Tinelli, editors. *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London,*

UK, April 11-18, 2015. Proceedings, volume 9035 of *Lecture Notes in Computer Science*. Springer, 2015.

[17] Alexey Bakhirkin, Josh Berdine, and Nir Piterman. A forward analysis for recurrent sets. In Sandrine Blazy and Thomas P. Jensen, editors, *Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings*, volume 9291 of *Lecture Notes in Computer Science*, pages 293–311. Springer, 2015.

[18] Alexey Bakhirkin and Nir Piterman. Finding recurrent sets with backward analysis and trace partitioning. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 17–35. Springer, 2016.

[19] Zuzana Baranová, Jiri Barnat, Katarína Kejstová, Tadeás Kucera, Henrich Lauko, Jan Mrázek, Petr Rockai, and Vladimír Still. Model checking of C and C++ with DIVINE 4. In Deepak D'Souza and K. Narayan Kumar, editors, *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, volume 10482 of *Lecture Notes in Computer Science*, pages 201–207. Springer, 2017.

[20] Anna Becchi and Enea Zaffanella. Revisiting polyhedral analysis for hybrid systems. In Bor-Yuh Evan Chang, editor, *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 183–202. Springer, 2019.

[21] Gerd Behrmann, Johan Bengtsson, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL implementation secrets. In Werner Damm and Ernst-Rüdiger Olderog, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 7th International Symposium, FTRTFT 2002, Co-sponsored by IFIP WG 2.2, Oldenburg, Germany, September 9-12, 2002, Proceedings*, volume 2469 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2002.

[22] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *STTT*, 8(3):204–215, 2006.

[23] Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, Luca Geretti, and Tiziano Villa. Assume–guarantee verification of nonlinear hybrid systems with Ariadne. *International Journal of Robust and Nonlinear Control*, 24(4):699–724, 2014.

[24] Béatrice Bérard and Catherine Dufourd. Timed automata and additive clock constraints. *Inf. Process. Lett.*, 75(1-2):1–7, 2000.

[25] Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Inform.*, 36(2-3):145–182, 1998.

[26] Marcello M. Bersani, Matteo Rossi, and Pierluigi San Pietro. An SMT-based approach to satisfiability checking of MITL. *Inf. Comput.*, 245:72–97, 2015.

[27] Tewodros A. Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. A constraint-based approach to solving games on infinite graphs. In Suresh Jagannathan and Peter Sewell, editors, *The*

41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014, pages 221–234. ACM, 2014.

[28] Tewodros A. Beyene, Corneliu Popeea, and Andrey Rybalchenko. Solving existentially quantified horn clauses. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 869–882. Springer, 2013.

[29] Armin Biere, Cyrille Artho, and Viktor Schuppan. Liveness checking as safety checking. *Electron. Notes Theor. Comput. Sci.*, 66(2):160–177, 2002.

[30] Armin Biere and Roderick Bloem, editors. *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*. Springer, 2014.

[31] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Adv. Comput.*, 58:117–148, 2003.

[32] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In Rance Cleaveland, editor, *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Pro-*

ceedings, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

[33] Johannes Birgmeier, Aaron R. Bradley, and Georg Weissenbacher. Counterexample to induction-guided abstraction-refinement (CTIGAR). In Biere and Bloem [30], pages 831–848.

[34] Nikolaj Bjørner and Arie Gurfinkel. Property directed polyhedral abstraction. In Deepak D'Souza, Akash Lal, and Kim Guldstrand Larsen, editors, *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, volume 8931 of *Lecture Notes in Computer Science*, pages 263–281. Springer, 2015.

[35] Ahmed Bouajjani and Yassine Lakhnech. Temporal logic + timed automata: Expressiveness and decidability. In Insup Lee and Scott A. Smolka, editors, *CONCUR '95: Concurrency Theory, 6th International Conference, Philadelphia, PA, USA, August 21-24, 1995, Proceedings*, volume 962 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 1995.

[36] Patricia Bouyer. Untameable timed automata! In Helmut Alt and Michel Habib, editors, *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.

[37] Patricia Bouyer. Model-checking timed temporal logics. *Electron. Notes Theor. Comput. Sci.*, 231:323–341, 2009.

[38] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.

[39] Patricia Bouyer, François Laroussinie, Nicolas Markey, Joël Ouaknine, and James Worrell. Timed temporal logics. In Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare, editors, *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, volume 10460 of *Lecture Notes in Computer Science*, pages 211–230. Springer, 2017.

[40] Aaron R. Bradley. SAT-based model checking without unrolling. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *LNCS*, pages 70–87. Springer, 2011.

[41] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Linear ranking with reachability. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 491–504. Springer, 2005.

[42] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. MightyL: A compositional translation from MITL to timed automata. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 421–440. Springer, 2017.

[43] Marc Brockschmidt, Byron Cook, Samin Ishtiaq, Heidy Khlaaf, and Nir Piterman. T2: Temporal property verification. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Con-*

ference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings, volume 9636 of *Lecture Notes in Computer Science*, pages 387–393. Springer, 2016.

[44] J Richard Büchi. On a decision method in restricted second order arithmetic. In *The Collected Works of J. Richard Büchi*, pages 425–435. Springer, 1990.

[45] Gianpiero Cabodi and Satnam Singh, editors. *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*. IEEE, 2012.

[46] Augustin Louis Baron Cauchy. *Exercices de mathématiques*, volume 3. De Bure frères, 1828.

[47] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 334–342. Springer, 2014.

[48] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, and Dorde Zikelic. Proving non-termination by program reversal. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 20211*, pages 1033–1048. ACM, 2021.

[49] Hong Yi Chen, Byron Cook, Carsten Fuhs, Kaustubh Nimkar, and Peter W. O'Hearn. Proving nontermination via safety. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the*

Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings, volume 8413 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2014.

[50] Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control, Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 29-31, 1999, Proceedings*, volume 1569 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 1999.

[51] 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 - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2021.

[52] Alessandro Cimatti, Alberto Griggio, and Enrico Magnago. Proving the existence of fair paths in infinite-state systems. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings*, volume 12597 of *Lecture Notes in Computer Science*, pages 104–126. Springer, 2021.

[53] Alessandro Cimatti, Alberto Griggio, and Enrico Magnago. Ltl falsification in infinite-state systems. *Information and Computation*, page 104977, 2022.

[54] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, and Stefano Tonetta. Extending nuXmv with timed transition systems and timed temporal properties. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 376–386. Springer, 2019.

[55] 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.

[56] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 modulo theories via implicit predicate abstraction. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2014.

[57] 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 - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July*

18-22, 2014. Proceedings, volume 8559 of *Lecture Notes in Computer Science*, pages 424–440. Springer, 2014.

[58] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. HyComp: An SMT-based model checker for hybrid systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2015.

[59] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.

[60] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. SMT-based verification of hybrid systems. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012.

[61] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. Requirements validation for hybrid systems. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Pro-*

ceedings, volume 5643 of *Lecture Notes in Computer Science*, pages 188–203. Springer, 2009.

[62] Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. In Cabodi and Singh [45], pages 52–59.

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

[64] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[65] George E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Barkhage, editor, *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer, 1975.

[66] Byron Cook, Carsten Fuhs, Kaustubh Nimkar, and Peter W. O'Hearn. Disproving termination with overapproximation. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, pages 67–74. IEEE, 2014.

[67] Byron Cook, Heidy Khlaaf, and Nir Piterman. On automation of CTL* verification for infinite-state systems. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th*

International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I, volume 9206 of *Lecture Notes in Computer Science*, pages 13–29. Springer, 2015.

[68] Byron Cook, Heidy Khlaaf, and Nir Piterman. Verifying increasingly expressive temporal logics for infinite-state systems. *J. ACM*, 64(2):15:1–15:39, 2017.

[69] Byron Cook, Eric Koskinen, and Moshe Y. Vardi. Temporal property verification as a program analysis task - extended version. *Formal Methods Syst. Des.*, 41(1):66–82, 2012.

[70] 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.

[71] Patrick Cousot. Abstract interpretation based formal methods and future challenges. In Reinhard Wilhelm, editor, *Informatics - 10 Years Back. 10 Years Ahead.*, volume 2000 of *Lecture Notes in Computer Science*, pages 138–156. Springer, 2001.

[72] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977.

[73] 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 - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23,*

2016, Proceedings, Part I, volume 9779 of *Lecture Notes in Computer Science*, pages 271–291. Springer, 2016.

[74] Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for linear temporal logic. In Nicolas Halbwachs and Doron A. Peled, editors, *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 1999.

[75] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.*, 17(4):397–415, 2015.

[76] Leonardo Mendonça 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, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[77] David Déharbe, Pascal Fontaine, Daniel Le Berre, and Bertrand Mazure. Computing prime implicants. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 46–52. IEEE, 2013.

[78] Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.

[79] JJ Doménech and S Genaim. iRankFinder. *WST*, 18:83, 2018.

[80] Bruno Dutertre. Solving exists/forall problems with Yices. In *Workshop on satisfiability modulo theories*, 2015.

[81] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

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

[83] Fabian Emmes, Tim Enger, and Jürgen Giesl. Proving non-looping non-termination automatically. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 225–240. Springer, 2012.

[84] Rebeka Farkas and Gábor Bergmann. Towards reliable benchmarks of timed automata. In Béla Pataki, editor, *Proceedings of the 25th PhD Mini-Symposium*, pages 20–23. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2018.

[85] Martin Fränzle and Christian Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods Syst. Des.*, 30(3):179–198, 2007.

[86] Goran Frehse and Matthias Althoff, editors. *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systemsi, part of CPS-IoT Week 2019, Montreal, QC, Canada, April 15, 2019*, volume 61 of *EPiC Series in Computing*. EasyChair, 2019.

[87] Florian Frohn and Jürgen Giesl. Proving non-termination via loop acceleration. In Clark W. Barrett and Jin Yang, editors, *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, pages 221–230. IEEE, 2019.

[88] Florian Frohn and Jürgen Giesl. Termination of triangular integer loops is decidable. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*, volume 11562 of *Lecture Notes in Computer Science*, pages 426–444. Springer, 2019.

[89] Marco Gario and Andrea Micheli. PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *SMT Workshop 2015*, 2015.

[90] Paul Gastin and Denis Oddoux. Fast LTL to büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[91] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In Piotr Dembinski and Marek Sredniawa, editors, *Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1995.

[92] Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Termination proofs for string rewriting systems via inverse match-bounds. *J. Autom. Reason.*, 34(4):365–385, 2005.

[93] Dimitra Giannakopoulou, Kedar S. Namjoshi, and Corina S. Pasareanu. Compositional reasoning. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 345–383. Springer, 2018.

[94] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with AProVE. *J. Autom. Reason.*, 58(1):3–31, 2017.

[95] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In Dirk Beyer, Marieke Huisman, Fabrice Kordon, and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*, volume 11429 of *Lecture Notes in Computer Science*, pages 156–166. Springer, 2019.

[96] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. Proving and disproving termination of higher-order functions. In Bernhard Gramlich, editor, *Frontiers of Combining Systems, 5th International Workshop, FroCoS 2005, Vienna, Austria, September 19-21, 2005, Proceedings*, volume 3717 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 2005.

[97] Antoine Girard, Colas Le Guernic, and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In João P. Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings*, volume 3927 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2006.

[98] Sergey Grebenshchikov, Nuno P. Lopes, Corneliu Popeea, and Andrey Rybalchenko. Synthesizing software verifiers from proof rules. In Jan Vitek, Haibo Lin, and Frank Tip, editors, *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China - June 11 - 16, 2012*, pages 405–416. ACM, 2012.

[99] Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 540–554. Springer, 2009.

[100] Ashutosh Gupta, Thomas A. Henzinger, Rupak Majumdar, Andrey Rybalchenko, and Ru-Gang Xu. Proving non-termination. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 147–158. ACM, 2008.

[101] Arie Gurfinkel and Nikolaj Bjørner. The science, art, and magic of constrained horn clauses. In *21st International Symposium on Sym-*

bolic and Numeric Algorithms for Scientific Computing, SYNASC 2019, Timisoara, Romania, September 4-7, 2019, pages 6–10. IEEE, 2019.

[102] Zyad Hassan, Aaron R. Bradley, and Fabio Somenzi. Incremental, inductive CTL model checking. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 532–547. Springer, 2012.

[103] Zyad Hassan, Aaron R. Bradley, and Fabio Somenzi. Better generalization in IC3. In *FMCAD*, pages 157–164. IEEE, 2013.

[104] Jan Havlíček. *Untimed LTL Model Checking of Timed Automata.* PhD thesis, Master's thesis. Masaryk University, Faculty of Informatics, 2013., 2013.

[105] Matthias Heizmann, Jürgen Christ, Daniel Dietsch, Evren Ermis, Jochen Hoenicke, Markus Lindenmann, Alexander Nutz, Christian Schilling, and Andreas Podelski. Ultimate Automizer with SMTInterpol - (competition contribution). In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*, pages 641–643. Springer, 2013.

[106] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science,*

New Brunswick, New Jersey, USA, July 27-30, 1996, pages 278–292. IEEE Computer Society, 1996.

[107] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. In Orna Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer, 1997.

[108] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.

[109] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.

[110] Aaron Herman and Hoon Hong. Quality of positive root bounds. *J. Symb. Comput.*, 74:592–602, 2016.

[111] Yoram Hirshfeld and Alexander Moshe Rabinovich. An expressive temporal logic for real time. In Rastislav Kralovic and Pawel Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006, 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings*, volume 4162 of *Lecture Notes in Computer Science*, pages 492–504. Springer, 2006.

[112] Krystof Hoder and Nikolaj Bjørner. Generalized property directed reachability. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 -*

15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings, volume 7317 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2012.

[113] Hoon Hong. Bounds for absolute positiveness of multivariate polynomials. *J. Symb. Comput.*, 25(5):571–585, 1998.

[114] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition.* Pearson international edition. Addison-Wesley, 2007.

[115] Mehran Hosseini, Joël Ouaknine, and James Worrell. Termination of linear loops over the integers. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 118:1–118:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[116] Tobias Isenberg and Heike Wehrheim. Timed automata verification via IC3 with zones. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2014.

[117] Daisuke Ishii, Kazunori Ueda, and Hiroshi Hosobe. An interval-based SAT modulo ODE solver for model checking nonlinear hybrid systems. *International Journal on Software Tools for Technology Transfer*, 13(5):449–461, 2011.

[118] Shachar Itzhaky, Nikolaj Bjørner, Thomas W. Reps, Mooly Sagiv, and Aditya V. Thakur. Property-directed shape analysis. In Biere and Bloem [30], pages 35–51.

[119] Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: High-performance language-independent model checking. In Baier and Tinelli [16], pages 692–707.

[120] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham. Property-directed inference of universal invariants or proving their absence. *J. ACM*, 64(1):7:1–7:33, 2017.

[121] Yonit Kesten and Amir Pnueli. A compositional approach to CTL* verification. *Theor. Comput. Sci.*, 331(2-3):397–428, 2005.

[122] Yonit Kesten, Amir Pnueli, and Li-on Raviv. Algorithmic verification of linear temporal logic specifications. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1998.

[123] Yonit Kesten, Amir Pnueli, Li-on Raviv, and Elad Shahar. Model checking with strong fairness. *Formal Methods Syst. Des.*, 28(1):57–84, 2006.

[124] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. SMT-based induction methods for timed systems. In Marcin Jurdzinski and Dejan Nickovic, editors, *Formal Modeling and Analysis of Timed Systems - 10th International Conference, FORMATS 2012,*

London, UK, September 18-20, 2012. Proceedings, volume 7595 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2012.

[125] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Bounded model checking of an MITL fragment for timed automata. In Josep Carmona, Mihai T. Lazarescu, and Marta Pietkiewicz-Koutny, editors, *13th International Conference on Application of Concurrency to System Design, ACSD 2013, Barcelona, Spain, 8-10 July, 2013*, pages 216–225. IEEE Computer Society, 2013.

[126] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. dReach: δ-reachability analysis for hybrid systems. In Baier and Tinelli [16], pages 200–205.

[127] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real Time Syst.*, 2(4):255–299, 1990.

[128] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.

[129] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. SMT techniques for fast predicate abstraction. In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *LNCS*, pages 424–437. Springer, 2006.

[130] Tim Lange, Martin R. Neuhäußer, and Thomas Noll. IC3 software model checking on control flow automata. In Roope Kaivola and Thomas Wahl, editors, *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, pages 97–104. IEEE, 2015.

[131] Daniel Larraz, Kaustubh Nimkar, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Proving non-termination using Max-SMT. In Biere and Bloem [30], pages 779–796.

[132] Timo Latvala, Armin Biere, Keijo Heljanko, and Tommi A. Junttila. Simple is better: Efficient bounded model checking for past LTL. In Radhia Cousot, editor, *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings*, volume 3385 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2005.

[133] Jan Leike and Matthias Heizmann. Ranking templates for linear loops. *Log. Methods Comput. Sci.*, 11(1), 2015.

[134] Jan Leike and Matthias Heizmann. Geometric nontermination arguments. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*, volume 10806 of *Lecture Notes in Computer Science*, pages 266–283. Springer, 2018.

[135] Guangyuan Li. Checking timed büchi automata emptiness using LU-abstractions. In Joël Ouaknine and Frits W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2009.

[136] Guangyuan Li, Peter Gjøl Jensen, Kim Guldstrand Larsen, Axel Legay, and Danny Bøgsted Poulsen. Practical controller synthesis

for $\mathrm{MTL}_{0,\infty}$. In Hakan Erdogmus and Klaus Havelund, editors, *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, pages 102–111. ACM, 2017.

[137] Salvador Lucas and José Meseguer. Termination of just/fair computations in term rewriting. *Inf. Comput.*, 206(5):652–675, 2008.

[138] IUriĭ V Matiiasevich, Jurij V Matijasevič, Ûrij Vladimirovič Matiâsevič, Yuri V Matiyasevich, Yuri Vladimirovich Matiyasevich, Michael R Garey, and Albert Meyer. *Hilbert's tenth problem*. MIT press, 1993.

[139] Théodore Samuel Motzkin. Two consequences of the transposition theorem on linear inequalities. *Econometrica (pre-1986)*, 19(2):184, 1951.

[140] Sumit Nain and Moshe Y. Vardi. Branching vs. linear time: Semantical perspective. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007, Tokyo, Japan, October 22-25, 2007, Proceedings*, volume 4762 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2007.

[141] Truong Nghiem, Sriram Sankaranarayanan, Georgios E. Fainekos, Franjo Ivancic, Aarti Gupta, and George J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In Karl Henrik Johansson and Wang Yi, editors, *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010*, pages 211–220. ACM, 2010.

[142] James Jerson Ortiz, Axel Legay, and Pierre-Yves Schobbens. Memory event clocks. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, volume 6246 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2010.

[143] Étienne Payet. Loop detection in term rewriting using the eliminating unfoldings. *Theor. Comput. Sci.*, 403(2-3):307–327, 2008.

[144] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[145] Swaroop N. Prabhakar and Vikram Sharma. A lower bound for computing lagrange's real root bound. In Vladimir P. Gerdt, Wolfram Koepf, Werner M. Seiler, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania, September 19-23, 2016, Proceedings*, volume 9890 of *Lecture Notes in Computer Science*, pages 444–456. Springer, 2016.

[146] Swaroop N. Prabhakar and Vikram Sharma. Improved bounds on absolute positiveness of multivariate polynomials. In Michael A. Burr, Chee K. Yap, and Mohab Safey El Din, editors, *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2017, Kaiserslautern, Germany, July 25-28, 2017*, pages 381–388. ACM, 2017.

[147] Alessandro Previti, Alexey Ignatiev, António Morgado, and João Marques-Silva. Prime compilation of non-clausal formulae. In Qiang

Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1980–1988. AAAI Press, 2015.

[148] Jean-François Raskin and Pierre-Yves Schobbens. State clock logic: A decidable real-time logic. In Oded Maler, editor, *Hybrid and Real-Time Systems, International Workshop. HART'97, Grenoble, France, March 26-28, 1997, Proceedings*, volume 1201 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 1997.

[149] Jean-François Raskin and Pierre-Yves Schobbens. The logic of event clocks - decidability, complexity and expressiveness. *J. Autom. Lang. Comb.*, 4(3):247–282, 1999.

[150] Sriram Sankaranarayanan and Georgios Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In Thao Dang and Ian M. Mitchell, editors, *Hybrid Systems: Computation and Control (part of CPS Week 2012), HSCC'12, Beijing, China, April 17-19, 2012*, pages 125–134. ACM, 2012.

[151] Viktor Schuppan and Armin Biere. Liveness checking as safety checking for infinite state spaces. *Electr. Notes Theor. Comput. Sci.*, 149(1):79–96, 2006.

[152] Malcolm C. Smith. The general problem of the stability of motion : Translated and edited by a. t. fuller. taylor and francis, 1992. *Autom.*, 31(2):353–354, 1995.

[153] F. Somenzi. CUDD: Colorado University Decision Diagram package — release 2.4.1.

[154] Fabio Somenzi and Roderick Bloem. Efficient büchi automata from LTL formulae. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2000.

[155] Stefano Tonetta. Abstract model checking without computing the abstraction. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2009.

[156] Stavros Tripakis. Verifying progress in timed systems. In Joost-Pieter Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, Bamberg, Germany, May 26-28, 1999. Proceedings*, volume 1601 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 1999.

[157] Stavros Tripakis. Checking timed büchi automata emptiness on simulation graphs. *ACM Trans. Comput. Log.*, 10(3):15:1–15:19, 2009.

[158] Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.

[159] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, volume 1043 of *LNCS*, pages 238–266. Springer, 1995.

[160] Yakir Vizel, Orna Grumberg, and Sharon Shoham. Lazy abstraction and SAT-based reachability in hardware model checking. In Cabodi and Singh [45], pages 173–181.

[161] Tobias Welp and Andreas Kuehlmann. QF BV model checking with property directed reachability. In Enrico Macii, editor, *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 791–796. EDA Consortium San Jose, CA, USA / ACM DL, 2013.

[162] Shakiba Yaghoubi and Georgios Fainekos. Local descent for temporal logic falsification of cyber-physical systems. In Roger D. Chamberlain, Walid Taha, and Martin Törngren, editors, *Cyber Physical Systems. Design, Modeling, and Evaluation - 7th International Workshop, CyPhy 2017, Seoul, South Korea, October 15-20, 2017, Revised Selected Papers*, volume 11267 of *Lecture Notes in Computer Science*, pages 11–26. Springer, 2017.

[163] Harald Zankl, Christian Sternagel, Dieter Hofbauer, and Aart Middeldorp. Finding and certifying loops. In Jan van Leeuwen, Anca Muscholl, David Peleg, Jaroslav Pokorný, and Bernhard Rumpe, editors, *SOFSEM 2010: Theory and Practice of Computer Science, 36th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 23-29, 2010. Proceedings*, volume 5901 of *Lecture Notes in Computer Science*, pages 755–766. Springer, 2010.