Compositional Model-Checking of Multi-Properties

Ohad Goudsmid

Compositional Model-Checking of Multi-Properties

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Ohad Goudsmid

Submitted to the Senate
of the Technion — Israel Institute of Technology
Adar 5781 Haifa February 2021

This research was carried out under the supervision of Prof. Orna Grumberg and Dr. Sarai Sheinvald, in the Faculty of Computer Science.

Some results in this dissertation have been published in a paper by the author and collaborators in a conference (VMCAI 2021) during the course of this masters, the most up-to-date version of which being:

> Ohad Goudsmid, Orna Grumberg, and Sarai Sheinvald. Compositional model checking for multi-properties. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings*, volume 12597 of *Lecture Notes in Computer Science*, pages 55–80. Springer, 2021.

Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Orna Grumberg, for being an incredible supervisor, a mentor, a friend, and for inspiring me along the course of my studies. Thank you for our weekly meetings, for constantly encouraging and aiding me to improve my work, for introducing me to the phenomenal world of research, and for making it a fun experience. It has been a huge honor to work with you and to learn from you.

I would also like to thank my co-advisor, Dr. Sarai Sheinvald, for her immense part of this work. Thank you for your dedication, and for the ability to make every meeting fruitful with new ideas, humor and sharp thought. You are a great advisor and a great person, and I am very grateful for working with you during this research.

Last but not the least, I would like to thank my family, for their endless love and support. Thank you for believing in me, for encouraging me to be curious and to always aim high.

The generous financial help of the Technion, and Melvin R. Berlin Fellowship for Cyber Security Research are hereby gratefully acknowledged.

Contents

List of Figures

Abstract

Hyperproperties lift conventional trace properties in a way that describes how a system behaves in its entirety, and not just based on its individual traces, by allowing quantification over traces. Hyperproperties are very useful for describing security properties. Thus, performing hyperproperty model-checking is desired. We generalize this notion to *multi-properties*, which describe the behavior of not just a single system, but of a set of systems, which we call a *multi-model*.

We demonstrate the usefulness of our setting with practical examples. We show that model-checking multi-properties is equivalent to model-checking hyperproperties. However, our framework has the immediate advantage of being *compositional*, allowing to consider abstraction for each component separately. We introduce sound and complete compositional proof rules for model-checking multi-properties, based on over- and under-approximations of the systems in the multi-model.

We then describe methods of computing such approximations. The first is abstraction-refinement based, in which a coarse initial abstraction is continually refined using counterexamples, until a suitable approximation is found. The second, tailored for models with finite traces, finds suitable approximations via the L^* learning algorithm. We suggest improved algorithms for model-checking the $\forall^*\exists^*$ fragment of both methods, which utilize additional information from the multi-property.

Our methods can produce much smaller models than the original ones, and can therefore be used for accelerating model-checking for both multi-properties and hyperproperties.

Abbreviations and Notations

AP	:	a set of atomic propositions
P	:	a trace property
$\mathcal{P}$	:	a hyperproperty, usually given as a formula in HyperLTL or as an NFH
$\mathbb{P}$	:	a multi-property, usually given as a formula in MultiLTL or as an MNFH
	:	
$\mathcal{M}$	:	a model, given as a Kripke structure or as an NFA
$\mathbb{M}$	:	a multi-model, given as a tuple of Kripke structures or as an MNFA
$\mathcal{A}$	:	an approximation, given as a Kripke structure or as an NFA
$\mathbb{A}, \mathbb{B}$	:	a tuple of approximations
	:	
π, τ	:	a trace
$\mathcal{L}(\cdot)$	:	a language, hyper-language or a multi-language
$\mathcal{L}_f(\cdot)$	:	a prefix language
	:	
$[a, b]$	:	the set $\{a, a+1, \ldots, b\}$
$\mathbb{Q}$	:	a trace quantifier ($\exists$, $\forall$)
$I_\exists(\cdot)$	:	the set of indices of existential quantifiers in a formula
$I_\forall(\cdot)$	:	the set of indices of universal quantifiers in a formula

Chapter 1

Introduction

Temporal logics, such as LTL, are widely used for specifying program behaviors. An LTL property characterizes a set of traces, each of which satisfies the property. It has recently been shown that trace properties are insufficient for characterizing and verifying security vulnerabilities or their absence.

The notion of *hyperproperties* [23], a generalization of trace properties, provides a uniform formalism for specifying properties of *sets of traces*. Hyperproperties are particularly suitable for specifying security properties. For instance, secure information flow may be characterized by identifying low-security variables that may be observable to the environment, and high-security variables that should not be observable outside. Secure information flow is maintained in a system if for every two traces, if their low-security inputs are identical then so are their low-security outputs, regardless of the values of high-security variables. This property cannot be characterized via single traces.

While hyperproperties are highly useful, they are still limited: they can only refer to the system as a whole. Systems often comprise several components, and it is desired to relate traces from one component to traces of another. A prominent such example is *diversity* [46]. Diversity generalizes the notion of security policies by considering policies of a set of systems. The systems are all required to implement the same functionality but to differ in their implementation details. As noticed in [23], such a set of policies could, in principle, be modeled as a hyperproperty on a single system, which is a product of all the systems in the set. This, however, is both unnatural and highly inefficient.

We remedy this situation by presenting a framework which explicitly describes the system as a set of systems called a *multi-model*, and provides a specification language, MultiLTL, which explicitly relates traces from the different components in the multi-model. Our framework enables to directly and naturally describe properties like diversity, while avoiding the need for a complex translation.

Our framework also has the immediate advantage of being *compositional*. We thus suggest a sound and complete compositional model-checking rule. The rule is based on abstracting each of the components by over- and under-approximations, thus achieving additional gain.

We then suggest methods of computing such approximations. The first method is based on *abstraction-refinement*, in which a coarse initial abstraction is continuously refined by using

counterexamples, until a suitable approximation is found. The second, tailored for models with finite traces, finds suitable approximations via the L^* learning algorithm. Our methods can produce much smaller models than the original ones, and can therefore be used for accelerating model-checking for both multi-properties and hyperproperties.

We now describe our work in more detail. Our framework consists of multi-models, which are tuples of Kripke structures. The logic we focus on, called MultiLTL, is an extension of HyperLTL [22]. MultiLTL allows indexed quantifications, $\forall^i$ and $\exists^i$, referring to the i^{th} component-model in the multi-model.

We show that there is a two-way reduction between the model-checking problem for HyperLTL and the model-checking problem for MultiLTL. We emphasize, that even though the two model-checking problems are equivalent, our new framework is clearly more powerful as it enables a direct specification and verification of the whole system by explicitly referring to its parts.

We exploit this power by introducing two compositional proof rules, which are based on over- and under-approximations for each system component separately. These proof rules are capable of proving a MultiLTL property or its negation for a given multi-model.

We suggest two approaches to computing these approximations for the compositional proof rules. The first approach is based on *abstraction-refinement*. The approximations are computed gradually, starting from coarse approximations and are refined based on counterexamples. The abstraction-refinement approach is implemented using one of two algorithms. In both algorithms, when model-checking the abstract multi-model is successful, we conclude that model-checking for the original multi-model holds. Otherwise, a counterexample is returned.

The first algorithm is based on counterexamples coming from the multi-model only. For each component-model, we find a behavior that should be eliminated from an over-approximated component-model or added to an under-approximated component-model, and refine the components accordingly.

The second algorithm is applicable for a restricted type of MultiLTL properties, in which the quantification consists of a sequence of $\forall$ quantifiers followed by a sequence of $\exists$ quantifiers. In hyperproperties, this is a useful fragment which allows specifying noninterference and generalized noninterference, observational determinism, and more. The counterexamples in this case come directly from the unsuccessful model-checking process, and therefore refer both to the model and to the property. Notice that, since the abstract component-models are typically much smaller than the original component-models, their model-checking is much faster.

The logics of MultiLTL and the model of Kripke structure are designed for describing and modeling the behavior of on-going systems. However, to do the same for terminating programs with finite traces, a more suitable description is needed. Therefore, we turn our attention to multi-models and multi-properties with finite traces. In this context, we use nondeterministic finite automata (NFA) to describe a system, and a set of NFAs (*multi-NFA*) to describe a set of such systems. For the specification language, we use nondeterministic finite-word hyperautomata (NFH) suggested in [16]. NFH can be thought of as the regular-language counterpart of LTL, and are able to describe the regular properties of sets of finite-word languages, just as HyperLTL is able to describe the properties of a language of infinite traces. Also like HyperLTL, NFH can be easily adjusted to describe multi-properties, a model that we call *multi-NFH*.

We show that, as in the infinite-trace case, there is a two-way reduction between the model-checking problem for NFH and the model-checking problem for multi-NFH. We then proceed to present a compositional model-checking framework for multi-NFH. As in the case of infinite-traces, this framework is based on finding approximations for the NFAs in the multi-model. The method for finding these approximations for this case, however, is learning-based.

Learning-based model-checking [45] seeks candidate approximations by running an automata-learning algorithm such as L^* [3]. In the L^* algorithm, a *learner* constructs a finite-word automaton for an unknown regular language $\mathcal{L}$, through a sequence of *membership queries* ("is the word w in $\mathcal{L}$?") and *equivalence queries* ("is $\mathcal{A}$ an automaton for $\mathcal{L}$?"), to which it receives answers from a *teacher* who knows the language. The learner continually constructs and submits candidate automata, until the teacher confirms an equivalence query.

In our algorithm, the learner constructs a set of candidate automata in every iteration, one for every NFA in the multi-model. The key idea is treating these candidate automata as candidate approximations. When an equivalence query is submitted, we (as the teacher) check whether the NFAs that the learner submitted are suitable approximations. If they are not, we return counterexamples to the learner, based on the given multi-NFA, which it uses to construct the next set of candidates. If they are suitable approximations, we model-check the multi-NFA of the approximations against the multi-NFH. Since the automata that the learner constructs are relatively small, model-checking the candidates multi-model is much faster than model-checking the original multi-model.

In [45], the learning procedure aims at learning the *weakest assumption* W, which is a regular language that contains all the traces that under certain conditions satisfy the specification. The construction of W relies on counterexample words provided by the model checking. We can derive such counterexamples for a certain fragment of multi-NFH. Moreover, we define a suitable weakest assumption for this case, prove that it is regular, and use it as a learning goal in an improved algorithm. Both of these improvements – extracting counterexamples from the model-checker, and learning the weakest assumption rather than the model itself – allow for an even quicker convergence of the model-checking process for this type of multi-properties.

1.1 Related Work

Abstractions are used to reduce the state-space of a model by discarding details which are irrelevant for the model-checking, thus improving the running time of the model-checking algorithms [37, 27]. In many cases, the initial abstraction is too coarse for solving the model-checking problem. *Refinement* is used for achieving a more precise abstraction in an iterative process. Some refinements are guided by counterexamples from previous model-checking calls. Counterexamples, returned by model checking abstract models, may be spurious. Thus, counterexamples are analyzed, and spurious ones are used to guide the refinement of the models. This approach is usually referred to as *counterexamples-guided abstraction-refinement* (CEGAR) [18, 19]. Combining abstraction-refinement with game-based model-checking is suggested in [50]. This work creates an abstraction which is both an over- and an under-approximation simultaneously, by using *may-transitions* and *must-transitions*. This abstraction is refined using counterexamples extracted from a game-based model-checking procedure.

Hyperproperties, introduced in [23], provide a uniform formalism for specifying properties of sets of traces, by quantification over traces in the system. Hyperproperties are particularly suitable for specifying security properties, such as secure information flow and non-interference. Two logics for hyperproperties are introduced in [22]: HyperLTL and HyperCTL*, which generalize LTL and CTL*, respectively. Other logics for hyperproperties have been studied in [52, 28, 36, 44, 1, 9, 13, 25].

One of the first sound and complete methods for model-checking hyperproperties is *self-composition* [8]. Self-composition combines several disjoint copies of the same program, allowing to express relationships among multiple traces. This reduces the k-trace hyperproperty model-checking to trace property model-checking. Unfortunately, the size of the product model increases exponentially with the number of copies. Thus, reasoning directly on the product program is prohibitive.

Many approaches have been suggested for dealing with the high complexity of the self-composition. A possible approach is using a symbolic representation for the system and the specification as first-order formulae. This allows to use an SMT solver as part of the model-checking. However, using self-composition still increases drastically the size of the representation, and methods to increase the efficiency of SMT solvers for hyperproperty model-checking have been suggested. In [53, 7] type-directed transformations are suggested, aiming to move similar program segments next to one another. [7] extends this idea to an aligned-by-fragments program, which employs lock-step execution of loops. In this manner, many variables change together, aiding solvers to find a solution.

A generalization of Hoare triplets for safety-hyperproperties is presented in [51]. This is possible by considering k copies of the program and allowing formulae to refer to all executions simultaneously. This allows ignoring redundant parts of the product program in its reasoning. A game-based approach for liveness-hyperproperties is described in [26]. This method also allow to synthesize a model for such hyperproperties.

In [40], the bounded model-checking approach [12] is extended for the partial verification of hyperproperties. Although their approach is not complete, they suggest optimistic and pessimistic semantics for the bounded constraints, allowing to refute and prove the satisfaction of the hyper-property.

Different approaches to avoid the construction of the full product are presented in [54, 49]. The former exploits taint analysis or Bounded Model Checking. The latter infers a self-composition function together with an inductive invariant, suitable for verification.

An automata based algorithm for HyperLTL and HyperCTL is proposed in [35]. It combines self-composition with ideas from LTL model-checking using alternating automata. Alternating tree-automata are utilized in [30] for representing a proof for the verification of safety-hyperproperties. This proof restrict the possible interleavings of the self-composed model, reducing the state-space.

The L^* algorithm for automata learning is described in [3] and improved by [47]. This algorithm consists of interaction between a *learner* and a *teacher*. The learners aims to construct a *minimal deterministic finite automaton* (DFA) for an unknown regular language U, using two kinds of queries – "is $w \in U$?", and *equivalence queries* – "is A a DFA for U?" – which

are answered by the teacher. This idea is extended to other kinds of automata, including: alternating automata [4, 11], weighted automata [10, 6], infinite-alphabet automata [43], and regular-ω-languages [29, 5].

Automata learning algorithms are used for generating approximations for model-checking procedures. In the assume-guarantee framework [24, 45], a system consists of two models, and automata learning algorithms are used for generating an approximation for one of the model. The L^* algorithm is used in order to create the approximation. Its learning goal is set to be the *weakest assumption*. In essence, the weakest assumption is the most general language with whom the property can be satisfied. In [45] alphabet refinement is also implemented, allowing to consider only a subset of the alphabet of the system, further reducing the approximation.

Learning separating DFAs for compositional model-checking is suggested in [17]. In this work, the problem of finding the weakest assumption is reduced to learning a separating DFAs, improving the running time of the model-checking procedure.

A representation of hyperproperties in a form of finite-word automata is developed in [31]. This work introduces a canonical automata representation for regular-k-safety hyperproperties, which are only-universally-quantified safety-hyperproperties. This representation uses a k-bad-prefix-automaton, a finite-word automaton that recognizes sets of k-bad-prefixes as finite words. The authors present a learning algorithm for k-safety hyperproperties.

The first representation of general hyperproperties using finite automata is introduced in [16]. This representation, called *hyperautomata*, allows running multiple quantified words on an automaton. The authors show that hyperautomata can express regular hyperproperties and explore the decidability of nonemptiness (satisfiability) and membership (model-checking) problems. Additionally, they describe an L^*-based learning algorithm for some fragments of hyperautomata.

The problem of runtime verification and monitoring hyperproperties is studied in [2, 34, 14, 39] from both algorithmic and theoretical point of view. Synthesis of hyperproperties is discussed in [32, 33], where decidable fragments are identified, and algorithms for their synthesis are suggested. The repair problem for HyperLTL is investigated in [15], and a detailed complexity analysis is shown, under different restrictions on the Kripke structure.

Notions of asynchronous hyperproperties are explored in [38]. One uses a new automata model and another is based on a fixpoint-calculus, allowing to regard different traces in asynchronous manner. As the model-checking problem for asynchronous hyperproperties is highly-undecidable, the work suggests an approximative analyses for both models, which induce some decidable fragments.

Chapter 2

Preliminaries

Kripke Structures are a standard model for ongoing finite-state systems.

Definition 2.0.1. Given a finite set of atomic propositions AP, a *Kripke structure* is a 4-tuple $\mathcal{M} = (S, I, R, L)$, where S is a finite set of *states*, $I \subseteq S$ is a non-empty set of *initial states*, $R \subseteq S \times S$ is a total *transition relation* and $L : S \to 2^{AP}$ is a *labeling function*.

A *path* in $\mathcal{M}$ is an infinite sequence of states $p = s_0, s_1, s_2, \ldots$ such that $(s_i, s_{i+1}) \in R$ for every $i \in \mathbb{N}$. A *trace over* AP is an infinite sequence $\tau \in \left(2^{AP}\right)^{\omega}$. We sometimes refer to a trace as an (infinite) *word* over 2^{AP}.

The *trace that corresponds to a path* p is the trace $\tau(p) = \tau_0, \tau_1, \tau_2, \ldots$ in which $\tau_i = L(s_i)$ for every $i \in \mathbb{N}$. Notice that since R is total, there exists an infinite path from every state. We denote by τ^i the trace $\tau_i, \tau_{i+1}, \ldots$ and by ${}^i\tau$ the prefix $\tau_0, \tau_1, \ldots, \tau_{i-1}$.

Given a word $w = w_0, w_1, \cdots \in (2^{AP})^{\omega}$, a *run of* $\mathcal{M}$ *on* w is a path $p = s_0, s_1, \ldots$ in $\mathcal{M}$ such that $L(s_n) = w_n$ for every $n \in \mathbb{N}$.

Definition 2.0.2. The *language* $\mathcal{L}(\mathcal{M})$ of $\mathcal{M}$ is the set of all traces corresponding to paths in $\mathcal{M}$ that start in I. The *prefix language* $\mathcal{L}_f(\mathcal{M})$ of $\mathcal{M}$ is the set of all finite prefixes of traces in $\mathcal{L}(\mathcal{M})$.

For two Kripke structures $\mathcal{M}, \mathcal{M}'$, we write $\mathcal{M} \models \mathcal{M}'$ to denote that $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{M}')$.

Definition 2.0.3. A possibly-infinite *tree* T is a subset of $\mathbb{N}^*_{>0}$ such that for every node $t \in \mathbb{N}^*_{>0}$ and every $n \in \mathbb{N}_{>0}$:

- If $t \cdot n \in T$, then $t \in T$.
- If $t \cdot n \in T$, then $t \cdot m \in T$ for every $0 < m < n$.

The *root of* T is the empty sequence ϵ and for a node $t \in T$, we use $|t|$ to denote the length of t.

Definition 2.0.4. Let $\mathcal{M} = (S, I, R, L)$ be a Kripke structure, and let Δ be an prefix-closed infinite set of finite paths in $\mathcal{M}$. The *infinite unwinding tree of* Δ is a tuple (T, ℓ, p) where T is a tree, $\ell : T \to S$ is a mapping from the tree to states in $\mathcal{M}$ and $p : T \to \Delta$ is a mapping from the tree to paths in Δ. Such that:

- $\ell(\epsilon) \in I$

- For every node t with children $t_1, \ldots, t_k$ it holds that $0 \leq k \leq |S|$ and that $(\ell(t), \ell(t_i)) \in R$ for every $i \in [1, k]$.

- For every node $t \cdot n \in T$, the path $p(t \cdot n)$ is of length $|t \cdot n|$ and $p(t)$ is the prefix of length $|t|$ of $p(t \cdot n)$.

The following is a known result, which can be proven using König's Lemma and the definition of infinite unwinding tree.

Lemma 2.0.5. *For Kripke structures $\mathcal{M}$ and $\mathcal{M}'$, it holds that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$ iff $\mathcal{L}_f(\mathcal{M}) = \mathcal{L}_f(\mathcal{M}')$.*

Proof. For the first direction, assume that $\mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)$. By Definition 2.0.2:

$$\begin{aligned}\mathcal{L}_f(\mathcal{M}_1) &= \{w \in (2^{AP})^* \mid \exists \tau \in \mathcal{L}(\mathcal{M}_1), n \in \mathbb{N} \text{ s.t. } {}^n\tau = w\} \\ &= \{w \in (2^{AP})^* \mid \exists \tau \in \mathcal{L}(\mathcal{M}_2), n \in \mathbb{N} \text{ s.t. } {}^n\tau = w\} \\ &= \mathcal{L}_f(\mathcal{M}_2)\end{aligned}$$

For the second direction, assume that $\mathcal{L}_f(\mathcal{M}_1) = \mathcal{L}_f(\mathcal{M}_2)$. Let $\tau \in \mathcal{L}(\mathcal{M}_1)$ be a trace. Thus, for every $n \in \mathbb{N}$, ${}^n\tau \in \mathcal{L}_f(\mathcal{M}_1)$. From the assumption, for every $n \in \mathbb{N}$, it holds that ${}^n\tau \in \mathcal{L}_f(\mathcal{M}_2)$. Therefore, for every prefix ${}^n\tau$, there is a path $s_0, s_1, \ldots, s_{n-1}$ in $\mathcal{M}_2$ which corresponds to it. Since I_2 is a finite set, there exists a state $s_0 \in I_2$, which appears in infinite such paths. Let Δ be the set of all such paths from s_0, and consider the infinite unwinding tree T of Δ. Since S_2 is finite, the branching degree of T is finite, and since Δ is infinite, this tree is infinite.

By König's lemma, there is an infinite path p in T. By definition, all traces that correspond to paths of length i in T are equal. This means that for every $i \in \mathbb{N}$, it holds that $\tau({}^ip) = {}^i\tau$. Thus, $\tau(p) = \tau$. Since T is an unwinding tree for $\mathcal{M}_2$, we get that $\tau \in \mathcal{L}(\mathcal{M}_2)$.

The proof for the inclusion $\mathcal{L}(M_2) \subseteq \mathcal{L}(\mathcal{M}_1)$ is symmetric. ■

2.1 Linear Time Logic (LTL)

The logic LTL is a common logic for describing trace properties.

Definition 2.1.1. Given a set of atomic propositions AP, a *trace property* is a set of traces $P \subseteq (2^{AP})^\omega$.

The (path) formulae of LTL are given by the following grammar:

$$\psi ::= a \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi \mathsf{U} \psi \qquad \text{for every } \mathsf{a} \in \mathsf{AP}$$

Let $\tau = \tau_0, \tau_1, \ldots$ be a trace over AP. The semantics of LTL are defined as follows:

$$\begin{aligned}
&\tau \models a \quad \text{iff } a \in \tau_0 \\
&\tau \models \neg\varphi \quad \text{iff } \tau \not\models \varphi \\
&\tau \models \varphi_1 \vee \varphi_2 \quad \text{iff } \tau \models \varphi_1 \text{ or } \tau \models \varphi_2 \\
&\tau \models \mathsf{X}\varphi \quad \text{iff } \tau^1 \models \varphi \\
&\tau \models \varphi_1 \mathsf{U} \varphi_2 \quad \text{iff there exists } i \geq 0 \text{ such that } \tau^i \models \varphi_2 \\
&\qquad\qquad\qquad \text{and for all } 0 \leq j < i,\ \tau^j \models \varphi_1
\end{aligned}$$

Trivially, we can add the logical operators $\wedge, \rightarrow, \leftrightarrow$. Similarly, it is possible to include additional temporal operators, which are defined as follows.

$$\mathsf{F}\varphi \equiv \mathbf{true}\mathsf{U}\varphi \qquad \mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\varphi \qquad \varphi_1 \mathsf{R} \varphi_2 \equiv \neg(\neg\varphi_1 \mathsf{U} \neg\varphi_2)$$

Given a Kripke structure M and an LTL formula φ, we say that M satisfies φ, denoted $M \models \varphi$, if $\tau \models \varphi$ for every $\tau \in \mathcal{L}(\mathcal{M})$.

2.1.1 Negation Normal Form of LTL

The fragment of LTL, where negation is applied only to atomic propositions, is called the *negation normal form* of LTL ($\mathsf{LTL}_{\mathsf{NNF}}$).

Definition 2.1.2. An LTL formula is in *negation normal form* if every negation operator is used only on atomic propositions. We denote the set of negation normal form formulae of LTL by $\mathsf{LTL}_{\mathsf{NNF}}$. In this definition, we allow the use of X, U and R as temporal operators and the additional logical operator $\wedge$.

Lemma 2.1.3. *The logic* LTL *is equivalent to the logic* $\mathsf{LTL}_{\mathsf{NNF}}$.

Proof. For the first direction, let φ_{NNF} be an $\mathsf{LTL}_{\mathsf{NNF}}$ formula. We show, by induction on the structure of LTL, that there exists a formula $\varphi \in \mathsf{LTL}$ which is equivalent to φ_{NNF}.

Base: When $\varphi_{\mathsf{NNF}} = a$ or $\varphi_{\mathsf{NNF}} = \neg a$ for some $a \in AP$, it holds that $\varphi_{\mathsf{NNF}} \in \mathsf{LTL}$.

Step: Assume that $\psi_{\mathsf{NNF}}, \phi_{\mathsf{NNF}}$ are $\mathsf{LTL}_{\mathsf{NNF}}$ formulae such that their equivalent LTL formulae are ψ and ϕ respectively.

- If $\varphi_{\mathsf{NNF}} = \psi \circ \phi$ for $\circ \in \{\vee, \mathsf{U}\}$, then the formula $\varphi = \psi \circ \phi$ is an LTL formula, which is equivalent to φ_{NNF}.
- If $\varphi_{\mathsf{NNF}} = \mathsf{X}\psi_{\mathsf{NNF}}$, then the formula $\varphi = \mathsf{X}\psi$ is an LTL formula, which is equivalent to φ_{NNF}.
- If $\varphi_{\mathsf{NNF}} = \psi_{\mathsf{NNF}} \wedge \phi_{\mathsf{NNF}}$, then by de-Morgan's laws and the induction hypothesis, the formula $\varphi = \neg\left((\neg\psi) \vee (\neg\phi)\right)$ is an LTL formula, which is equivalent to φ_{NNF}.
- If $\varphi_{\mathsf{NNF}} = \psi_{\mathsf{NNF}} \mathsf{R} \phi_{\mathsf{NNF}}$, then by the definition of R and the induction hypothesis, $\varphi = \neg\left((\neg\psi)\,\mathsf{U}\,(\neg\phi)\right)$ is an LTL formula, which is equivalent to φ_{NNF}.

For the second direction, let φ be an LTL formula. We show, by induction on the structure of LTL, that there exists formulae $\varphi_{\mathsf{NNF}}, \bar{\varphi}_{\mathsf{NNF}} \in \mathsf{LTL}_{\mathsf{NNF}}$ such that, φ_{NNF} is equivalent to φ and $\bar{\varphi}_{\mathsf{NNF}}$ is equivalent to $\neg\varphi$.

Base: When $\varphi = a$ or $\varphi = \neg a$ for some $a \in AP$, it holds that $\varphi, \neg\varphi \in \mathsf{LTL}_{\mathsf{NNF}}$, since we identify $\neg\neg\varphi$ with φ, and they satisfy the requirements.

Step: Assume that ψ, ϕ are LTL formulae and let ψ_{NNF}, $\bar{\psi}_{\mathsf{NNF}}$, ϕ_{NNF} and $\bar{\phi}_{\mathsf{NNF}}$ be their equivalent and negated $\mathsf{LTL}_{\mathsf{NNF}}$ formulae respectively.

- If $\varphi = \psi \circ \phi$ for $\circ \in \{\vee, \mathsf{U}\}$, then $\varphi_{\mathsf{NNF}} = \psi_{\mathsf{NNF}} \circ \phi_{\mathsf{NNF}}$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula, which is equivalent to φ. Additionally, $\bar{\psi}_{\mathsf{NNF}}\ \bar{\circ}\ \bar{\phi}_{\mathsf{NNF}}$, where $\bar{\wedge} = \vee$ and $\bar{\mathsf{U}} = \mathsf{R}$, is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula, equivalent to $\neg\varphi$.

- If $\varphi = \mathsf{X}\psi$, then the formula $\varphi_{\mathsf{NNF}} = \mathsf{X}\psi_{\mathsf{NNF}}$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula, which is equivalent to φ. Additionally, $\mathsf{X}\bar{\psi}_{\mathsf{NNF}}$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula equivalent to $\neg\varphi$.

- If $\varphi = \neg(\psi \vee \phi)$, then by de-Morgan's laws and the induction hypothesis, the formula $\varphi_{\mathsf{NNF}} = \left(\bar{\psi}_{\mathsf{NNF}}\right) \wedge \left(\bar{\phi}_{\mathsf{NNF}}\right)$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula, which is equivalent to φ. Additionally, $\psi_{\mathsf{NNF}} \vee \phi_{\mathsf{NNF}}$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula equivalent to $\neg\varphi$.

- If $\varphi = \neg(\psi \mathsf{U} \phi)$, then by the definition of R and the induction hypothesis, the formula $\varphi_{\mathsf{NNF}} = \left(\bar{\psi}_{\mathsf{NNF}}\right) \mathsf{R} \left(\bar{\phi}_{\mathsf{NNF}}\right)$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula, which is equivalent to φ. Additionally, $\psi_{\mathsf{NNF}} \mathsf{U} \phi_{\mathsf{NNF}}$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula equivalent to $\neg\varphi$.

- If $\varphi = \neg\mathsf{X}\psi$, then by the semantics of X and the induction hypothesis, the formula $\varphi_{\mathsf{NNF}} = \mathsf{X}\left(\bar{\psi}_{\mathsf{NNF}}\right)$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula, which is equivalent to φ. Additionally, $\mathsf{X}\psi_{\mathsf{NNF}}$ is an $\mathsf{LTL}_{\mathsf{NNF}}$ formula equivalent to $\neg\varphi$.

Since every $\mathsf{LTL}_{\mathsf{NNF}}$ formula is finite, this process terminates in a finite number of steps.

Note that this construction is linear in the length of the formula. ■

2.2 Hyperproperties and HyperLTL

Trace properties and the logics that express them are commonly used to describe desirable system behaviors. However, some behaviors cannot be expressed by referring to each trace individually. This is especially the case for many security protocols, such as non-interference and observational determinism [22]. These properties refer to a system with low-security variables that may be observable to the environment, and high-security variables that should not be observable outside. Actions in this system, may refer to low- and high-security variables, separately. Non-interference is maintained in this system, if for every trace, there exists an additional trace, without actions that affect high-security variables, which is equivalent to the original trace, over the low-security variables. Observational determinism holds, if the system appears deterministic to an observer from the environment.

In [23], properties describing the behavior of a combination of traces are formalized as *hyperproperties*. Thus, a hyperproperty is a set of sets of traces: all sets that behave according to the hyperproperty.

Definition 2.2.1. Given a set of atomic propositions AP, a *hyperproperty* is a subset of the powerset of traces, $\mathcal{P} \subseteq 2^{(2^{AP})^\omega}$.

HyperLTL [22] is an extension of *linear temporal logic* (LTL) to hyperproperties. The formulae of HyperLTL are given by the following grammar:

$$\varphi ::= \exists \pi.\ \varphi \mid \forall \pi.\ \varphi \mid \psi \qquad \text{where } \pi \text{ is a trace variable}$$
$$\psi ::= a_\pi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi \mathsf{U} \psi \qquad \text{for every } a \in AP$$

Intuitively, $\exists \pi.\varphi$ means that there exists a trace that satisfies φ and $\forall \pi.\varphi$ means that φ holds for every trace. a_π means that a holds in the first state of π. The semantics of X, U and the Boolean operators are similar to those in LTL.

The semantics of HyperLTL is defined as follows. Let $T \subseteq (2^{AP})^\omega$ be a set of traces over AP, let $\mathcal{V}$ be a set of *trace variables*, and $\Pi : \mathcal{V} \to T$ be a trace assignment. Let $\Pi[\pi \to t]$ be the function obtained from Π, by mapping π to t. Let Π^i be the function defined by $\Pi^i(\pi) = (\Pi(\pi))^i$.

$$\Pi \models_T \exists \pi.\psi \quad \text{iff there exists } t \in T \text{ such that } \Pi[\pi \to t] \models_T \psi$$
$$\Pi \models_T \forall \pi.\psi \quad \text{iff for every } t \in T,\ \Pi[\pi \to t] \models_T \psi$$
$$\Pi \models_T a_\pi \quad \text{iff } a \in \Pi(\pi)[0]$$
$$\Pi \models_T \neg\varphi \quad \text{iff } \Pi \not\models_T \varphi$$
$$\Pi \models_T \varphi_1 \vee \varphi_2 \quad \text{iff } \Pi \models_T \varphi_1 \text{ or } \Pi \models_T \varphi_2$$
$$\Pi \models_T \mathsf{X}\varphi \quad \text{iff } \Pi^1 \models_T \varphi$$
$$\Pi \models_T \varphi_1 \mathsf{U} \varphi_2 \quad \text{iff there exists } i \geq 0 \text{ such that } \Pi^i \models_T \varphi_2 \text{ and for all } 0 \leq j < i,\ \Pi^j \models_T \varphi_1$$

Notice that when all trace variables of a HyperLTL formula $\mathcal{P}$ are in the scope of a quantifier (i.e. when $\mathcal{P}$ is *closed*), then the satisfaction is independent of the trace assignment, in which case we write $T \models \mathcal{P}$. Given a Kripke structure $\mathcal{M}$ and a HyperLTL formula $\mathcal{P}$, the *model-checking problem* is to decide whether $\mathcal{L}(\mathcal{M}) \models \mathcal{P}$ (which we denote by $\mathcal{M} \models \mathcal{P}$).

By abuse of notation, given traces $w_1, \ldots, w_k$ over AP, we write $\langle w_1, \ldots, w_k \rangle \models \mathbb{Q}_1 \pi_1 \ldots \mathbb{Q}_k \pi_k \psi(\pi_1, \ldots, \pi_k)$ if $\Pi \models \psi(\pi_1, \ldots, \pi_k)$, where $\Pi(\pi_i) = w_i$.

2.2.1 Negation Normal Form of HyperLTL

Definition 2.2.2. An HyperLTL formula is in *negation normal form* if every negation operator is used only on atomic propositions. We denote the set of negation normal form formulae of HyperLTL by $\mathsf{HyperLTL}_{\mathsf{NNF}}$.
Note that similarly to Definition 2.1.2, we allow the use of X, U, R and $\wedge$, as well as the trace quantifiers $\exists$ and $\forall$.

Lemma 2.2.3. *The logic* HyperLTL *is equivalent to the logic* $\mathsf{HyperLTL}_{\mathsf{NNF}}$.

Proof. For the first direction, let $\mathcal{P}_{\mathsf{NNF}} = \mathbb{Q}_1 \pi_1, \ldots, \mathbb{Q}_n \pi_n . \varphi_{\mathsf{NNF}}(\pi_1, \ldots, \pi_n)$ be an $\mathsf{HyperLTL}_{\mathsf{NNF}}$ formula. By the definition of $\mathsf{HyperLTL}_{\mathsf{NNF}}$, the formula $\varphi_{\mathsf{NNF}}(\pi_1, \ldots, \pi_n)$ is in $\mathsf{LTL}_{\mathsf{NNF}}$. By

applying Lemma 2.1.3, we can obtain an equivalent formula $\varphi(\pi_1, \ldots, \pi_n)$ in LTL. Thus, the formula $\mathcal{P} = \mathbb{Q}_1\pi_1, \ldots, \mathbb{Q}_n\pi_n.\varphi(\pi_1, \ldots, \pi_n)$ is in $\mathsf{HyperLTL}$ and is equivalent to $\mathcal{P}_{\mathsf{NNF}}$.

For the second direction, let $\mathcal{P} = \mathbb{Q}_1\pi_1, \ldots, \mathbb{Q}_n\pi_n.\varphi(\pi_1, \ldots, \pi_n)$ be an $\mathsf{HyperLTL}$ formula. By the definition of $\mathsf{HyperLTL}$, the formula $\varphi(\pi_1, \ldots, \pi_n)$ is in LTL. By applying Lemma 2.1.3, we can obtain an equivalent formula $\varphi_{\mathsf{NNF}}(\pi_1, \ldots, \pi_n)$ in $\mathsf{LTL}_{\mathsf{NNF}}$. Thus, the formula $\mathcal{P}_{\mathsf{NNF}} = \mathbb{Q}_1\pi_1, \ldots, \mathbb{Q}_n\pi_n.\varphi_{\mathsf{NNF}}(\pi_1, \ldots, \pi_n)$ is in $\mathsf{HyperLTL}_{\mathsf{NNF}}$ and is equivalent to $\mathcal{P}$.

Chapter 3

Multi-Models and Multi-Properties

We generalize hyperproperties to *multi-properties*, which reason about the connections between several models, which we call a *multi-model*.

Definition 3.0.1. Given $k \in \mathbb{N}$, a *k-multi-model* is a k-tuple $\mathbb{M} = \langle \mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k \rangle$ of Kripke structures over a common set of atomic propositions AP. The *multi-language* of a k-multi-model $\mathbb{M}$, denoted $\mathcal{L}(\mathbb{M})$, is a tuple of languages $\langle \mathcal{L}(\mathcal{M}_1), \ldots, \mathcal{L}(\mathcal{M}_k) \rangle$. A *$k$-multi-property* is a set of tuples $\mathbb{P} \subseteq (2^{(2^{AP})^\omega})^k$.

$\mathbb{M}$ is a *multi-model* if it is a k-multi-model for some k, and similarly $\mathbb{P}$ is a *multi-property*.

Intuitively, in a multi-property $\mathbb{P}$, every $T \in \mathbb{P}$ is a tuple of k sets of traces, each interpreted in a model.

3.1 MultiLTL

We now present MultiLTL, a logic for describing multi-properties. A MultiLTL formula is interpreted over a multi-model $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle$. We use $[a, b]$, where $a \leq b$ are integers, to denote the set $\{a, a+1, \ldots, b\}$. MultiLTL formulae are defined inductively as follows.

$$\varphi ::= \exists^j \pi.\ \varphi \mid \forall^j \pi.\ \varphi \mid \psi \qquad \text{where } j \in [1,k] \text{ and } \pi \text{ is a trace variable}$$
$$\psi ::= a_\pi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi \mathsf{U} \psi \qquad \text{for every } a \in AP$$

The only difference in syntax from HyperLTL is that trace quantifiers are now indexed. This index is taken from the set $[1, k]$ for some $k \in \mathbb{N}$. The formula $\exists^j \pi.\varphi$ means that there exists a trace in $\mathcal{M}_j$ that satisfies φ and $\forall^j \pi.\varphi$ means that φ holds for every trace in $\mathcal{M}_j$.

The semantics of MultiLTL is defined as follows. Let $\mathbb{T} = \langle T_1, \ldots T_k \rangle$ be a multi-language over AP, called *domain*. Let $\mathcal{V}$ be a set of trace variables, and let $\Pi : \mathcal{V} \to \bigcup_{i \in [1,k]} T_i$.

$$\Pi \models_{\mathbb{T}} \exists^i \pi.\psi \quad \text{iff there exists } t \in T_i \text{ such that } \Pi[\pi \to t] \models_{\mathbb{T}} \psi$$
$$\Pi \models_{\mathbb{T}} \forall^i \pi.\psi \quad \text{iff } \Pi[\pi \to t] \models_{\mathbb{T}} \psi \text{ for every } t \in T_i$$

The semantics of the temporal operators is defined as in HyperLTL. Since every MultiLTL formula describes a multi-property, we refer to the formulae themselves as multi-properties.

As with HyperLTL, when a MultiLTL formula $\mathbb{P}$ is closed, satisfaction is independent of Π, and we denote $\mathbb{T} \models \mathbb{P}$. Given a multi-model $\mathbb{M}$ and a MultiLTL formula $\mathbb{P}$, the *model-checking problem* for MultiLTL is to decide whether $\mathcal{L}(\mathbb{M}) \models \mathbb{P}$ (which we denote $\mathbb{M} \models \mathbb{P}$).

For a MultiLTL formula $\mathbb{P} = \mathbb{Q}_1^{i_1}\pi_1 \dots \mathbb{Q}_n^{i_n}\pi_n.\varphi$, we define the sets $I_\exists(\mathbb{P}) = \{j \mid \mathbb{Q}_j^i = \exists \text{ and } i \in [1,k]\}$, and $I_\forall(\mathbb{P}) = \{j \mid \mathbb{Q}_j^i = \forall \text{ and } i \in [1,k]\}$. We write $I_\exists$ and $I_\forall$ when $\mathbb{P}$ is clear from the context.

3.1.1 Examples

We demonstrate the usefulness of MultiLTL and multi-models with some examples. The multi-models we consider consist of models that interact with each other via an asynchronous communication channel (which is not modeled). This assumption is not necessary outside the scope of the examples, where other forms of interactions across models can take place (e.g., shared variables).

Example 3.1.1. Consider a multi-model consisting of a client model C and a server model S. We would like to check whether $\langle C, S\rangle \models \forall^C\pi_1\forall^S\pi_2.\mathsf{G}(r_{\pi_1} \rightarrow \mathsf{F}r_{\pi_2})$. In this formula, r_{π_1} means that a request is sent in C and r_{π_2} means that a request is received in S. The formula specifies that for every run of the client and for every run of the server, every request sent by the client is eventually received by the server. This is a form of a liveness property that specifies that messages are guaranteed to eventually arrive at their destination.

Example 3.1.2. Consider again the multi-model of Example 3.1.1. Assume that the interaction between the client and the server is as follows. At the beginning of the interaction, the client sends its username and password to the server. Immediately afterwards the server updates its authentication flag and informs the client whether the authentication was successful or not. The client gets this notification one clock cycle after the server authentication flag has been updated. Consider the specification $\mathbb{P}_2$.

$$\begin{aligned}\mathbb{P}_2 = \forall^S\pi_1\exists^C\pi_2\forall^C\pi_3.\\ ((userDB_{\pi_1} = user_{\pi_2}) \wedge ((passDB_{\pi_1} = pass_{\pi_2})) \wedge (\mathsf{X}aut_{\pi_1} \wedge \mathsf{XX}aut_{\pi_2})\\ \wedge\ ((userDB_{\pi_1} = user_{\pi_3}) \wedge ((passDB_{\pi_1} \neq pass_{\pi_3})) \rightarrow (\mathsf{X}\neg aut_{\pi_1} \wedge \mathsf{XX}\neg aut_{\pi_3})\end{aligned}$$

The first two lines of $\mathbb{P}_2$ states that for every trace of the server there is a trace of the client whose username and password match the username and password in the server database. If so, the authentication succeeds. The third line assures that for each username in the server database there is only one valid password with which the authentication succeeds.

Note that in this example, we describe a property which cannot be described using LTL. Further, it cannot be expressed naturally in HyperLTL. MultiLTL, which explicitly refers to traces in different models within a multi-model, naturally expresses it.

Example 3.1.3. We demonstrate again the power of MultiLTL to *naturally* express properties that are not naturally expressible in HyperLTL. *Diversity* [46] refers to security policies of a set of systems. The systems constitute different implementations of the same high-level program.

They differ in their implementation details[1], but are equivalent with respect to the input-output they produce. In [46], diversity has been advocated as a successful way to resist attacks that exploit memory layout or instruction sequence specifics.

Assume that we are given a high-level program P and two low-level implementations M_1 and M_2. The following MultiLTL properties describe the fact that all implementations are equivalent to P.

$$\begin{aligned}\mathbb{P}_1 = \forall^{P}\pi\exists^{M_1}\pi_1\exists^{M_2}\pi_2.(input_\pi = input_{\pi_1} = input_{\pi_2}) \wedge \\ \mathsf{G}(end_\pi \wedge end_{\pi_1} \wedge end_{\pi_2} \rightarrow output_\pi = output_{\pi_1} = output_{\pi_2}) \\ \mathbb{P}_2 = \forall^{M_1}\pi_1\exists^{P}\pi.(input_{\pi_1} = input_\pi) \wedge \mathsf{G}(end_{\pi_1} \wedge end_\pi \rightarrow output_{\pi_1} = output_\pi) \\ \mathbb{P}_3 = \forall^{M_2}\pi_2\exists^{P}\pi.(input_{\pi_2} = input_\pi) \wedge \mathsf{G}(end_{\pi_2} \wedge end_\pi \rightarrow output_{\pi_2} = output_\pi)\end{aligned}$$

Note that these properties cannot naturally be expressed in HyperLTL since they require an explicit reference to the models from which the related traces are taken.

3.1.2 Negation Normal Form of MultiLTL

Definition 3.1.4. A MultiLTL formula is in *negation normal form* if every negation operator is used only on atomic propositions. We denote the set of negation normal form formulae of MultiLTL by $\mathsf{MultiLTL}_{\mathsf{NNF}}$.
Note that similarly to Definition 2.1.2 and Definition 2.2.2, we allow the use of X, U, R and $\wedge$, and the trace quantifiers $\exists^j$ and $\forall^j$.

Lemma 3.1.5. *The logic* MultiLTL *is equivalent to the logic* $\mathsf{MultiLTL}_{\mathsf{NNF}}$.

Proof. The proof is almost identical to the proof of Lemma 2.2.3. ■

3.2 Model-Checking MultiLTL

We now show that although MultiLTL is a generalization of HyperLTL, the model-checking problems for these logics are equivalent.

3.2.1 Reduction from HyperLTL Model-Checking to MultiLTL Model-Checking

For the first direction, it is easy to see that the model-checking problem for a model $\mathcal{M}$ and a HyperLTL formula $\mathcal{P}$ is equivalent to the model checking problem for $\langle\mathcal{M}\rangle$ and the MultiLTL formula obtained from $\mathcal{P}$ by indexing all of its quantifiers with the same index, 1.

Theorem 3.1. *The model-checking problem for* HyperLTL *is polynomially reducible to the model-checking problem for* MultiLTL.

[1]For instance, the call stack of procedures is obfuscated by changing the order of variables, the specific memory location of arguments and local variables, etc. The obfuscations differ in the different implementations.

Proof. Let $\mathcal{M}$ be a Kripke structure and let $\mathcal{P} = \mathbb{Q}_1\pi_1 \dots \mathbb{Q}_n\pi_n.\varphi(\pi_1, \dots, \pi_n)$ be a HyperLTL formula. For $\mathbb{M} = \langle \mathcal{M} \rangle$ and the MultiLTL formula $\mathbb{P} = \mathbb{Q}_1^1\pi_1 \dots \mathbb{Q}_n^1\pi_n.\varphi(\pi_1, \dots, \pi_n)$, we show that $\mathcal{M} \models \mathcal{P}$ iff $\mathbb{M} \models \mathbb{P}$.

Let $\Pi : \mathcal{V} \to \mathcal{L}(\mathcal{M})$ be an assignment, $T = \mathcal{L}(M)$ and $\mathbb{T} = \langle \mathcal{L}(\mathcal{M}) \rangle$. Since there was no change to φ, it is immediate that $\Pi \models_T \varphi(\pi_1, \dots, \pi_n)$ iff $\Pi \models_{\mathbb{T}} \varphi(\pi_1, \dots, \pi_n)$. Denote by $\mathcal{P}_k(\pi_1, \dots, \pi_{k-1})$ the formula $\mathbb{Q}_k\pi_k \dots \mathbb{Q}_n\pi_n\varphi(\pi_1, \dots, \pi_n)$ and by $\mathbb{P}_k(\pi_1, \dots, \pi_{k-1})$ the formula $\mathbb{Q}_k^1\pi_k \dots \mathbb{Q}_n^1\pi_n\varphi(\pi_1, \dots, \pi_n)$.

Assume that $\Pi \models_T \mathcal{P}_k$ iff $\Pi \models_{\mathbb{T}} \mathbb{P}_k$. Consider the quantifier $\mathbb{Q}_{k-1}$ in $\mathcal{P}$:

- If $\mathbb{Q}_{k-1} = \exists$, then $\Pi \models_T \exists\pi_{k-1}.\mathcal{P}_k$ iff there exists a trace $\tau \in \mathcal{L}(\mathcal{M})$ such that $\Pi[\pi_{k-1} \to \tau] \models_T \mathcal{P}_k$. By the induction hypothesis, this holds iff there exists a trace $\tau \in \mathcal{L}(\mathcal{M})$ such that $\Pi[\pi_{k-1} \to \tau] \models_{\mathbb{T}} \mathbb{P}_k$. This claim holds iff $\Pi \models_{\mathbb{T}} \exists^1\pi_{k-1}.\mathbb{P}_k$, since $\mathbb{T} = \langle T \rangle$.
- If $\mathbb{Q}_{k-1} = \forall$, then $\Pi \models_T \forall\pi_{k-1}.\mathcal{P}_k$ iff for every trace $\tau \in \mathcal{L}(\mathcal{M})$ it holds that $\Pi[\pi_{k-1} \to \tau] \models_T \mathcal{P}_k$. This is iff for every trace $\tau \in \mathcal{L}(\mathcal{M})$ it holds that $\Pi[\pi_{k-1} \to \tau] \models_{\mathbb{T}} \mathbb{P}_k$. This claim holds iff $\Pi \models_{\mathbb{T}} \forall^1\pi_{k-1}.\mathbb{P}_k$, since $\mathbb{T} = \langle T \rangle$.

Notice that by induction this also holds for $\mathcal{P} = \mathcal{P}_1$ and $\mathbb{P} = \mathbb{P}_1$, which do not contain free variables. Thus, $\mathcal{M} \models \mathcal{P}$ iff $T \models \mathcal{P}$ iff $\mathbb{T} \models \mathbb{P}$ iff $\mathbb{M} \models \mathbb{P}$ as required. ■

3.2.2 Reduction from MultiLTL Model-Checking to HyperLTL Model-Checking

For the other direction, we first introduce several definitions. We use the notation $\uplus$ for disjoint union.

Definition 3.2.1. Given a multi-model $\mathbb{M} = \langle \mathcal{M}_1, \dots, \mathcal{M}_k \rangle$ over AP, its *union model* is $\cup\mathbb{M} = (\uplus_{i=1}^n S_i, \uplus_{i=1}^n I_i, \uplus_{i=1}^n R_i, L)$, where $L(s) = L_i(s) \uplus \{\mathbf{i}\}$ for every i and $s \in S_i$.
The *indexing by* i of a trace $\tau = t_0, t_1, \dots$ over AP is the trace $\mathsf{ind}_i(\tau) = t_0 \cup \{\mathbf{i}\}, t_1 \cup \{\mathbf{i}\}, \dots$.
We define indexing of a word in a similar manner.
Note that ind_i is invertible. The inverse of ind_i is denoted ind_i^{-1}. Given an indexed trace τ', the trace $\mathsf{ind}_i^{-1}(\tau')$ is called the *unindexed trace* of τ'.

For a trace τ and a multi-model $\mathbb{M} = \langle \mathcal{M}_1, \dots, \mathcal{M}_k \rangle$, it holds that $\tau \in \mathcal{L}(\mathcal{M}_i)$ iff $\mathsf{ind}_i(\tau) \in \mathcal{L}(\cup\mathbb{M})$, as shown by the following lemma.

Lemma 3.2.2. *Let $\mathbb{M} = \langle \mathcal{M}_1, \dots, \mathcal{M}_k \rangle$ be a multi-model over AP. Then for every $i \in [1, k]$ it holds that $\tau \in \mathcal{L}(\mathcal{M}_i) \iff \mathsf{ind}_i(\tau) \in \mathcal{L}(\cup\mathbb{M})$.*

Proof. We first prove the claim for finite words, by induction on the structure of $(2^{AP})^*$.

Base: $w = \epsilon$, by definition of Kripke structure, $w \in \mathcal{L}_f(M_i)$ and $\mathsf{ind}_i(w) = w \in \mathcal{L}_f(\cup\mathbb{M})$ for every $i \in \{1, \dots, k\}$.

Step: Assume that $w = w'\sigma$ for $w' \in (2^{AP})^*$. If $w \in \mathcal{L}_f(\mathcal{M}_i)$ for some i than there are states $s_i, s_i' \in S_i$ such that s_i is the end of the run of $\mathcal{M}_i$ over w and s_i' is the end of the same run over w'. Therefore, there exists a transition $(s_i', s_i) \in R_i$. Since Kripke structures are prefix-closed, $w' \in \mathcal{L}_f(\mathcal{M}_i)$, and by the induction hypothesis, $\mathsf{ind}_i(w) \in \mathcal{L}_f(\cup\mathbb{M})$. This means also that there is a transition $(s_i', s_i) \in R$ in $\cup\mathbb{M}$, which means that $\mathsf{ind}_i(w) \in \mathcal{L}_f(\cup\mathbb{M})$.

For the second direction, assume that $w \notin \mathcal{L}_f(\mathcal{M}_i)$, therefore, there is an index j such that ${}^j w$, the prefix of w of length j, is not in $\mathcal{L}_f(\mathcal{M}_i)$. As a result, by the induction hypothesis, $\mathsf{ind}_i\left({}^j w\right) \notin \mathcal{L}_f(\cup\mathbb{M})$, and therefore also $\mathsf{ind}_i\left(w\right) \notin \mathcal{L}_f(\cup\mathbb{M})$.

Consequently, this must also hold for infinite traces, since otherwise, there is some finite prefix for which it does not hold, contradicting the previous induction. ■

Note 3.2.3. Consider an input $\mathbb{M}, \mathbb{P}$ to the model-checking problem of MultiLTL. Without loss of generality, we can assume that each sub-model in $\mathbb{M}$ is quantified exactly once, and furthermore, that $\mathbb{P}$ is a $\mathsf{MultiLTL}_{\mathsf{NNF}}$ formula of the form $\mathbb{Q}_1^1\pi_1\mathbb{Q}_2^2\pi_2 \ldots \mathbb{Q}_n^n\pi_n.\varphi(\pi_1, \ldots, \pi_n)$. This can be achieved by duplicating models which are quantified several times and by reordering the models according to the order of the quantifiers.

We assume that $\mathbb{M}$ and $\mathbb{P}$ are in this form, for the rest of the section.

Definition 3.2.4. Given a model $\mathcal{M}$ and a HyperLTL formula $\mathcal{P}$, an assignment Π *respects the model* $\mathcal{M}$, if $\Pi(\pi_i) \in \mathcal{L}(\mathcal{M})$ for every $i \in [1, n]$.

Similarly, given a multi-model $\mathbb{M}$ and a MultiLTL formula $\mathbb{P}$, an assignment Π *respects the multi-model* $\mathbb{M}$, if $\Pi(\pi_i) \in \mathcal{L}(\mathcal{M}_i)$ for every $i \in [1, n]$.

Note that when checking whether a model (multi-model) satisfies a hyperproperty (multi-property), it is enough to consider only assignments that respect the model (multi-model) for every sub-formula of the property. This holds since there are no free trace variables in the formula.

Definition 3.2.5. Given a multi-model $\mathbb{M}$, a MultiLTL formula $\mathbb{P}$ and an assignment Π that respects $\mathbb{M}$, the *indexed assignment of* Π, denoted $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)$, is defined by $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)\left(\pi_i\right) = \mathsf{ind}_i\left(\Pi(\pi_i)\right)$ for every $i \in [1, n]$.

Given an indexed assignment Π, the *unindexed assignment of* Π, denoted $\mathsf{ind}^{-1}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)$, is defined by $\mathsf{ind}^{-1}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)\left(\pi_i\right) = \mathsf{ind}_i^{-1}\left(\Pi(\pi_i)\right)$.

Lemma 3.2.6. *Let* $\mathbb{M}$ *be a multi-model,* $\mathbb{P}$ *be a multi-property and* Π *be an assignment that respects* $\mathbb{M}$*. Then,* $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)$ *respects* $\cup\mathbb{M}$*.*

Proof. Since Π respects $\mathbb{M}$, it holds that $\Pi(\pi_i) \in \mathcal{L}(\mathcal{M}_i)$ for every $i \in [1, n]$. By Lemma 3.2.2, this implies that $\mathsf{ind}_i\left(\Pi(\pi_i)\right) \in \mathcal{L}(\cup\mathbb{M})$. Since $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)\left(\pi_i\right) = \mathsf{ind}_i\left(\Pi(\pi_i)\right)$, we also get that $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)\left(\pi_i\right) \in \mathcal{L}(\cup\mathbb{M})$. ■

Lemma 3.2.7. *Let* $\mathbb{M}$ *be a multi model,* $\mathbb{P}$ *be a multi property and* Π *an assignment that respects* $\mathbb{M}$*. For every* $j \in \mathbb{N}$*, it holds that* $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)^j = \mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi^j\right)$*.*

Proof. The following equations hold for every π_i and $j \in \mathbb{N}$, as required.

$$\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)^j\left(\pi_i\right) = \left(\mathsf{ind}_i\left(\Pi(\pi_i)\right)\right)^j = \mathsf{ind}_i\left(\Pi^j(\pi_i)\right) = \mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi^j\right)\left(\pi_i\right)$$ ■

Definition 3.2.8. Let $\mathbb{P} = \mathbb{Q}_1^1\pi_1 \ldots \mathbb{Q}_n^n\pi_n\varphi(\pi_1, \ldots, \pi_n)$ be a $\mathsf{MultiLTL}_{\mathsf{NNF}}$ formula. The *transformed formula of* $\mathbb{P}$, denoted $\mathsf{trans}\left(\mathbb{P}\right)$, is defined inductively as follows:

- If $\varphi = a_{\pi_i}$ or $\varphi = \neg a_{\pi_i}$ for $a \in AP$, we define $\mathsf{trans}(\varphi)$ to be $\mathbf{i}_{\pi_i} \rightarrow \varphi$ if $\mathbb{Q}_i^i = \forall^i$, and $\mathbf{i}_{\pi_i} \wedge \varphi$ otherwise.
- If $\varphi = \mathsf{X}\psi$, we define $\mathsf{trans}(\varphi) = \mathsf{X}\mathsf{trans}(\psi)$.
- If $\varphi = \psi \circ \phi$ for some $\circ \in \{\vee, \wedge, \mathsf{U}, \mathsf{R}\}$, we define $\mathsf{trans}(\varphi) = \mathsf{trans}(\psi) \circ \mathsf{trans}(\phi)$.
- If $\varphi = \mathbb{Q}^i \pi_i.\psi$ for $\mathbb{Q} \in \{\forall, \exists\}$, we define $\mathsf{trans}(\varphi) = \mathbb{Q}\pi_i.\mathsf{trans}(\psi)$.

Intuitively, $\mathsf{trans}(\mathbb{P})$ is obtained from $\mathbb{P}$ by the following changes:

1. For every $a \in AP$, every occurrence of a literal $l = a_\pi$ or $l = \neg a_\pi$ where π is universally quantified, is replaced with $\mathbf{i}_\pi \rightarrow l$. This change requires that the constraints on the atomic propositions are enforced only on traces that originate from the model on which the universal quantified was used in the multi-property. For traces from other models, the requirement is immediately satisfied.
2. For every $a \in AP$, every occurrence of a_π and for every occurrence of a literal $l = a_\pi$ or $l = \neg a_\pi$ where π is existentially quantified, is replaced with $\mathbf{i}_\pi \wedge l$. This requires that the trace used for the satisfaction of the existential quantifier is from the model which was quantified in the multi-property.
3. Every quantifier $\mathbb{Q}_i^i$ is replaced with $\mathbb{Q}_i$.

Note that when $\mathbb{P}$ is a $\mathsf{MultiLTL_{NNF}}$ formula, $\mathsf{trans}(\mathbb{P})$ is a $\mathsf{HyperLTL_{NNF}}$ formula.

Lemma 3.2.9. *Let $\mathbb{M}$ be a multi-model, $\mathbb{P} = \mathbb{Q}_1^1\pi_1 \ldots \mathbb{Q}_n^n\pi_n.\varphi(\pi_1, \ldots, \pi_n)$ be a $\mathsf{MultiLTL_{NNF}}$ formula and let Π be a trace assignment that respects $\mathbb{M}$. Then $\Pi \models \varphi(\pi_1, \ldots, \pi_n)$ iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi) \models \mathsf{trans}(\varphi(\pi_1, \ldots, \pi_n))$.*

Proof. By induction on the structure of $\mathsf{MultiLTL_{NNF}}$ without quantifiers.
Base: $\varphi = a_{\pi_i}$ or $\varphi = \neg a_\pi$. We show the proof for $\varphi = a_{\pi_i}$, the case where $\varphi = \neg a_{\pi_i}$ is similar.

- If $i \in I_\exists$, then $\mathsf{trans}(\varphi) = \mathbf{i}_{\pi_i} \wedge \varphi$. It holds that $\Pi \models a_{\pi_i}$ iff $a \in \Pi(\pi_i)$ iff $\{a, \mathbf{i}\} \subseteq \mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi)(\pi_i)$ iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi) \models \mathbf{i}_{\pi_i} \wedge a_{\pi_i}$.
- If $i \in I_\forall$, then $\mathsf{trans}(\varphi) = \mathbf{i}_{\pi_i} \rightarrow \varphi$. Note that Π respects $\mathbb{M}$, which means that $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi)$ respects $\cup\mathbb{M}$. Therefore, it holds that $\Pi \models a_{\pi_i}$ iff $a \in \Pi(\pi_i)$ iff $\{a, \mathbf{i}\} \subseteq \mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi)(\pi_i)$ iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi) \models \mathbf{i}_{\pi_i} \rightarrow a_{\pi_i}$. Note that in the last transition, we use the fact that $\mathbf{i} \in \mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi)(\pi_i)$ for every i.

Step: Let φ_1, φ_2 be $\mathsf{MultiLTL}$ formulae. Assume that $\Pi \models \varphi_i$ iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi) \models \mathsf{trans}(\varphi_i)$ for $i \in [1, 2]$. By the definition of $\mathsf{trans}(\cdot)$ the step follows:

- Let $\circ \in \{\wedge, \vee\}$. It holds that $\Pi \models \varphi_1 \circ \varphi_2$ iff $(\Pi \models \varphi_1) \circ (\Pi \models \varphi_2)$ iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi) \models \mathsf{trans}(\varphi_1)$ and/or $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi) \models \mathsf{trans}(\varphi_2)$ iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi) \models \mathsf{trans}(\varphi_1 \circ \varphi_2)$.
- $\Pi \models \mathsf{X}\varphi$ iff $\Pi^1 \models \varphi$ iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi^1) \models \mathsf{trans}(\varphi)$ iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi)^1 \models \mathsf{trans}(\varphi)$ iff $\mathsf{ind}_{\mathbb{P}}(\Pi) \models \mathsf{trans}(\mathsf{X}\varphi)$.

- $\Pi \models \varphi_1 \mathsf{U} \varphi_2$ iff there exists $k \geq 0$ such that $\Pi^k \models \varphi_2$ and for every $0 \leq i < k$ $\Pi^i \models \varphi_1$. This is iff there exists $k \geq 0$ such that $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi^k\right) \models \mathsf{trans}\left(\varphi_2\right)$ and for every $0 \leq i < k$ $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi^i\right) \models \mathsf{trans}\left(\varphi_1\right)$. This holds iff there exists $k \geq 0$ such that $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)^k \models \mathsf{trans}\left(\varphi_2\right)$ and for every $0 \leq i < k$ $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right)^i \models \mathsf{trans}\left(\varphi_1\right)$. This is iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right) \models (\mathsf{trans}\left(\varphi_1\right))\mathsf{U}(\mathsf{trans}\left(\varphi_2\right))$, which is iff $\mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi\right) \models \mathsf{trans}\left(\varphi_1 \mathsf{U} \varphi_2\right)$.
- The proof for R is similar to the proof for U.

For the cases $\mathsf{X}, \mathsf{U}, \mathsf{R}$ we use Lemma 3.2.7 in some of the transitions. ■

Definition 3.2.10. Given a MultiLTL formula $\mathbb{P}$, the *basic sub-formulae* of $\mathsf{trans}\left(\mathbb{P}\right)$ are all the sub-formulae of $\mathsf{trans}\left(\mathbb{P}\right)$ of the form: $\mathbf{i}_{\pi_i} \rightarrow a_{\pi_i}$ or $\mathbf{i}_{\pi_i} \wedge a_{\pi_i}$.

The following lemma shows that for every $\mathsf{MultiLTL}_{\mathsf{NNF}}$ formula $\mathbb{P}$, the formula $\mathsf{trans}\left(\mathbb{P}\right)$ is monotonic with respect to its basic sub-formulae. I.e. by changing the truth value of some basic sub-formulae to **true**, the truth value of the entire formula cannot become **false**, unless it was **false** before.

Lemma 3.2.11. *Let* $\mathbb{P} = \mathbb{Q}_1^1 \pi_1 \ldots \mathbb{Q}_n^n \pi_n . \varphi(\pi_1, \ldots, \pi_n)$ *be a* $\mathsf{MultiLTL}_{\mathsf{NNF}}$*, such that* $\mathbb{Q}_k = \forall$*. Let* $\mathbb{M}$ *be a multi-model and* Π *be an assignment for* $\mathsf{trans}\left(\varphi\right)$*. If* $\Pi \models \mathsf{trans}\left(\varphi\right)$ *and if* τ *is a trace that is not marked by the index* $\mathbf{k}$*, then* $\Pi[\pi_k \rightarrow \tau] \models \mathsf{trans}\left(\varphi\right)$*.*

Proof. Let Π and τ be as in the lemma. We show a proof by induction on the structure of a $\mathsf{MultiLTL}_{\mathsf{NNF}}$ formula without quantifiers.

Base: If there is no a_{π_k} in the formula, then the assignment is independent from $\Pi[\pi_k]$, which means that if $\Pi \models \mathsf{trans}\left(\varphi\right)$, then also $\Pi[\pi_k \rightarrow \tau] \models \mathsf{trans}\left(\varphi\right)$. Otherwise, $\varphi = \mathbf{k}_{\pi_k} \rightarrow a_{\pi_k}$. Since τ is not indexed by $\mathbf{k}$, then $\Pi[\pi_k \rightarrow \tau] \models \mathsf{trans}\left(\varphi\right)$ vacuously.

Step: Since all the operators in $\{\wedge, \vee, \mathsf{X}, \mathsf{U}, \mathsf{R}\}$ are monotonic, the step is immediate. ■

We now can show the reduction from MultiLTL model-checking problem to HyperLTL model-checking problem.

Theorem 3.2. *The model-checking problem for* MultiLTL *is polynomially reducible to the model-checking problem for* HyperLTL*.*

Proof. Let $\mathbb{P} = \mathbb{Q}_1^1 \pi_1 \ldots \mathbb{Q}_n^n \pi_n . \varphi(\pi_1, \ldots, \pi_n)$ be a $\mathsf{MultiLTL}_{\mathsf{NNF}}$ formula and let $\mathbb{M}$ be a multi-model $\langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle$, both over AP. Denote $\mathbb{T} = \langle \mathcal{L}(\mathcal{M}_1), \ldots, \mathcal{L}(\mathcal{M}_n) \rangle$ and $T = \mathcal{L}(\cup\mathbb{M})$. Let Π be an assignment that respects $\mathbb{M}$ and Λ be an assignment that respects $\cup\mathbb{M}$, such that $\mathsf{ind}_i^{-1}\left(\Lambda(\pi_i)\right) \in \mathcal{L}(\mathcal{M}_i)$ for every $i \in I_\exists$.. We show that $\Pi \models_{\mathbb{T}} \mathbb{P}$ iff $\Lambda \models_T \mathsf{trans}\left(\mathbb{P}\right)$.
Let $\mathbb{P}_k(\pi_1, \ldots, \pi_{k-1}) = \mathbb{Q}_k^k \pi_k \ldots \mathbb{Q}_n^n \pi_n . \varphi$ and $\mathsf{trans}\left(\mathbb{P}\right)_k(\pi_1, \ldots, \pi_{k-1}) = \mathsf{trans}\left(\mathbb{P}_k\right)$, for every $k \in [1, n+1]$. Note that $\varphi = \mathbb{P}_{n+1}$ and $\mathsf{trans}\left(\varphi\right) = \mathsf{trans}\left(\mathbb{P}\right)_{n+1}$.

First Direction: Assume that for every assignment Π that respects $\mathbb{M}$, it holds that $\Pi \models_{\mathbb{T}} \mathbb{P}$. Let Λ be an assignment that respects $\cup\mathbb{M}$, such that $\mathsf{ind}_i^{-1}\left(\Lambda(\pi_i)\right) \in \mathcal{L}(\mathcal{M}_i)$ for every $i \in I_\exists$. By induction on the number of quantifiers in $\mathbb{P}$, we show that if $\Pi \models_{\mathbb{T}} \mathbb{P}$ then $\Lambda \models_T \mathsf{trans}\left(\mathbb{P}\right)$. Note that since $\mathbb{P}$ and $\mathsf{trans}\left(\mathbb{P}\right)$ does not contain free trace variables, this means that this claim also holds for λ that does not fulfill the requirements, when considering the *entire* formula.

Base: When the formula does not contain quantifiers, we consider two cases. If $\Lambda = \mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi)$ for some Π that respects $\mathbb{M}$, then the base case holds according to Lemma 3.2.9.

Otherwise, Λ assigns at least one trace variable π_i, a trace that is not indexed by $\mathbf{i}$ for $i \in I_\forall$. In this case, every basic sub-formula that refers to those trace variables is vacuously satisfied. When considering an assignment Λ' which agrees with Λ on the assignment for every π_i, when $i \in I_\exists$, by the first case, $\Lambda' \models_t \mathsf{trans}(\varphi)$. By Lemma 3.2.11, this also holds for Λ, since it is can be obtained by changing the traces assigned by Λ' for universally quantified trace variables.

Step: Assume that the claim holds for $\mathbb{P}_{k+1}$ and $\mathsf{trans}(\mathbb{P})_{k+1}$. We now show that it holds for $\mathbb{P}_k$ and $\mathsf{trans}(\mathbb{P})_k$. We consider two cases.

- $\mathbb{P}_k = \exists^k \pi_k.\mathbb{P}_{k+1}$: This means that $\mathsf{trans}(\mathbb{P})_k = \exists \pi_k.\mathsf{trans}(\mathbb{P})_{k+1}$.
  Assume that $\Pi \models_\mathbb{T} \exists^k \pi_k.\mathbb{P}_{k+1}$. By the semantics of MultiLTL, this means that there exists $\tau \in \mathcal{L}(\mathcal{M}_k)$ such that $\Pi[\pi_k \to \tau] \models_\mathbb{T} \mathbb{P}_{k+1}$. Since $\tau \in \mathcal{L}(\mathcal{M}_k)$, the assignment $\Pi[\pi_k \to \tau]$ respects $\mathbb{M}$. Additionally, $\mathsf{ind}_k(\tau) \in \mathcal{L}(\cup\mathbb{M})$ by construction. By the induction hypothesis, $\Lambda[\pi_k \to \mathsf{ind}_k(\tau)] \models_T \mathsf{trans}(\mathbb{P})_{k+1}$, which means by the semantics of HyperLTL that $\Lambda \models_T \exists \pi_k.\mathsf{trans}(\mathbb{P})_{k+1}$.

- $\mathbb{P}_k = \forall^k \pi_k.\mathbb{P}_{k+1}$: This means that $\mathsf{trans}(\mathbb{P})_k = \forall \pi_k.\mathsf{trans}(\mathbb{P})_{k+1}$.
  Assume that $\Pi \models_\mathbb{T} \forall^k \pi_k.\mathbb{P}_{k+1}$. By the semantics of MultiLTL, this means that for every $\tau \in \mathcal{L}(\mathcal{M}_k)$, $\Pi[\pi_k \to \tau] \models_\mathbb{T} \mathbb{P}_{k+1}$. Thus, by the induction hypothesis, $\Lambda[\pi_k \to \mathsf{ind}_k(\tau)] \models_T \mathsf{trans}(\mathbb{P})_{k+1}$. By Lemma 3.2.11, since for every trace $\tau' \in \mathcal{L}(\cup\mathbb{M})$ such that $\mathsf{ind}_k^{-1}(\tau') \notin \mathcal{L}(\mathcal{M}_k)$, all the basic sub-formulae of $\mathsf{trans}(\mathbb{P})_k$ that refer to π_k are satisfied. Also $\Lambda[\pi_k \to \tau'] \models_T \mathsf{trans}(\mathbb{P})_{k+1}$.

Second Direction: Assume that $\Lambda \models_T \mathsf{trans}(\mathbb{P})$ for every assignment Λ that respects $\cup\mathbb{M}$. By Lemma 3.2.6, this means that for every assignment Λ such that $\Lambda = \mathsf{ind}_{\mathbb{M},\mathbb{P}}(\Pi)$, for Π that respects $\mathbb{M}$, it holds that $\Lambda \models \mathsf{trans}(\mathbb{P})$.

Let Π be an assignment that respects $\mathbb{M}$. By induction on the number of quantifiers in $\mathbb{P}$, we show that if $\Lambda \models_T \mathsf{trans}(\mathbb{P})$ then $\Pi \models_\mathbb{T} \mathbb{P}$.

Base: The base case holds according to Lemma 3.2.9, in a similar manner to the previous direction.

Step: Assume that the claim holds for $\mathbb{P}_{k+1}$ and $\mathsf{trans}(\mathbb{P})_{k+1}$. We now show that it holds for $\mathbb{P}_k$ and $\mathsf{trans}(\mathbb{P})_k$.

- $\mathbb{P}_k = \exists^k \pi_k.\mathbb{P}_{k+1}$: This means that $\mathsf{trans}(\mathbb{P})_k = \exists \pi_k.\mathsf{trans}(\mathbb{P})_{k+1}$.
  Assume that $\Lambda \models_T \exists \pi_k.\mathsf{trans}(\mathbb{P})_{k+1}$. By the semantics of HyperLTL, this means that there exists $\tau \in \mathcal{L}(\cup\mathbb{M})$ such that $\Lambda[\pi_k \to \tau] \models_T \mathsf{trans}(\mathbb{P})_{k+1}$. Since $\tau \in \mathcal{L}(\cup\mathbb{M})$, the assignment $\Lambda[\pi_k \to \tau]$ respects $\cup\mathbb{M}$.

  - If $\mathsf{ind}_k^{-1}(\tau) \in \mathcal{L}(\mathcal{M}_k)$, then $\Pi[\pi_k \to \mathsf{ind}_i^{-1}(\tau)]$ respects $\mathbb{M}$ and also $\Lambda[\pi_k \to \tau] = \mathsf{ind}_{\mathbb{M},\mathbb{P}}\left(\Pi[\pi_k \to \mathsf{ind}_i^{-1}(\tau)]\right)$. Thus, by the induction hypothesis, $\Pi[\pi_k \to \mathsf{ind}_k^{-1}(\tau)] \models_\mathbb{T} \mathbb{P}_{k+1}$. This means that $\Pi \models_\mathbb{T} \exists^k \pi_k.\mathbb{P}_{k+1}$.
  - Otherwise, $\mathsf{ind}_k^{-1}(\tau) \notin \mathcal{L}(\mathcal{M}_k)$. Additionally, every basic sub-formula that refers to π_k in $\mathsf{trans}(\mathbb{P})_{k+1}$, is of the form $\mathbf{k} \wedge a_{\pi_k}$ for some $a \in AP$. Since $\mathsf{ind}_k^{-1}(\tau) \notin \mathcal{L}(\mathcal{M}_k)$, we

get that $\tau \models \mathsf{G}\neg\mathbf{k}$. Thus, the every basic sub-formula that refers to π_k is not satisfied. By the monotonicity of $\mathsf{HyperLTL}_{\mathsf{NNF}}$ (following a similar proof to Lemma 3.2.11), $\Pi[\pi_k \to \tau'] \models_{\mathbb{T}} \mathbb{P}_{k+1}$ for some τ' such that $\mathsf{ind}_k^{-1}(\tau') \in \mathcal{L}(\mathcal{M}_k)$. Thus we return to the previous case.

- $\mathbb{P}_k = \forall^k \pi_k.\mathbb{P}_{k+1}$: This means that $\mathsf{trans}(\mathbb{P})_k = \forall \pi_k.\mathsf{trans}(\mathbb{P})_{k+1}$.
  Assume that $\Lambda \models_T \forall \pi_k.\mathsf{trans}(\mathbb{P})_{k+1}$. By the semantics of $\mathsf{HyperLTL}$, this means that for every $\tau \in \mathcal{L}(\cup\mathbb{M})$, it holds that $\Lambda[\pi_k \to \tau] \models_T \mathsf{trans}(\mathbb{P})_{k+1}$. Since $\tau \in \mathcal{L}(\cup\mathbb{M})$, this also holds for every τ such that $\mathsf{ind}_k^{-1}(\tau) \in \mathcal{L}(\mathcal{M}_k)$. By the induction hypothesis, $\Pi[\pi_k \to \mathsf{ind}_k^{-1}(\tau)]$ respects $\mathbb{M}$ and $\Pi[\pi_k \to \mathsf{ind}_k^{-1}(\tau)] \models_{\mathbb{T}} \mathbb{P}_{k+1}$. By the semantics of $\mathsf{MultiLTL}$, this results in $\Pi \models_{\mathbb{T}} \mathbb{P}_k$. ■

Note that this proof is stronger than a simple reduction. Theorem 3.2 implies that there is a linear-time reversible translation of the traces of $\mathbb{M}$ to the traces of $\mathcal{M}$. This is especially useful when handling counterexamples.

3.3 Direct Algorithm for MultiLTL Model-Checking

In [35], the authors present an algorithm for model-checking $\mathsf{HyperLTL}$ that can be easily adjusted for $\mathsf{MultiLTL}$. Thus, there is no need to use the reduction in Theorem 3.2 for $\mathsf{MultiLTL}$ model-checking. The algorithm relies roughly on the repeated intersection of the models under $\exists$ with an automaton for φ, the quantifier-free part of the formula, or, in the case of $\forall$ quantifiers, for $\neg\varphi$ (which involves complementation). Accordingly, the complexity is a tower in the number of models, and the size of the models greatly influences the run-time.

Definition 3.3.1. A *non-deterministic Büchi automaton* (NBW) is a tuple $A = (Q, q_0, \Sigma, \delta, \alpha)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and $\alpha \subseteq Q$ is a set of accepting states.

Definition 3.3.2. Let Q be a finite set. The set of *positive boolean formulae over* Q, denoted $\mathbb{B}^+(Q)$, is defined inductively as follows:

- $\mathbf{true}, \mathbf{false} \in \mathbb{B}^+(Q)$ and $q \in \mathbb{B}^+(Q)$ for every $q \in Q$.
- If $\varphi_1, \varphi_2 \in \mathbb{B}^+(Q)$ then also $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are in $\mathbb{B}^+(Q)$.

Let $Q' \subseteq Q$ and $\theta \in \mathbb{B}^+(Q)$. We say that Q' *satisfies* θ, denoted $Q' \models \theta$, if $z_{Q'} \models \theta$, where:

$$z_{Q'}(q) = \begin{cases} 0, & \text{if } q \notin Q' \\ 1, & \text{if } q \in Q' \end{cases}$$

Definition 3.3.3. An *alternating Büchi automaton* (ABW) is a tuple $(Q, q_0, \Sigma, \delta, \alpha)$, where Q, q_0, Σ and α are as in Definition 3.3.1 and $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is a transition function mapping each state and letter to a positive Boolean formula of states.

Note that, an NBW $\mathcal{A} = (Q, q_0, \Sigma, \delta, \alpha)$ can be seen as an ABW where $\delta(q, \sigma)$ is a disjunction, for every $q \in Q$ and $\sigma \in \Sigma$.

Definition 3.3.4. Let $\mathcal{A}$ be an ABW and $w = w_0, w_1, \cdots \in \Sigma^\omega$ be an inifinte-word. A *run of* $\mathcal{A}$ *on* w is a tuple (T, r), where T is a tree (see Definition 2.0.3), and $r : T \to Q$ is a mapping from nodes in the tree to states in $\mathcal{A}$, such that the following holds: $r(\epsilon) = q_0$ and for every node $t \in T$ with children $t_1, \ldots, t_k$ it holds that $\{r(t_1), \ldots, r(t_k)\} \models \delta\left(r(t), w_{|t|}\right)$.

A run (T, r) of $\mathcal{A}$ on w is *accepting* if one of the two holds.

- For every infinite path $t_0, t_1 \ldots$ in T, there are infinitely many i with $r(t_i) \in \alpha$.
- Every maximal finite path $t_0, \ldots, t_k$ ends in a **true** transition.

We say that w is accepted by $\mathcal{A}$ if there is an accepting run of A on w, and denote by $\mathcal{L}(\mathcal{A})$ the set of infinite words accepted by $\mathcal{A}$.

It is a known result that NBW and ABW are equivalent in their expressive power.

Theorem 3.3. *[42] For every ABW $\mathcal{A}$, there exists a NBW $\mathcal{B}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

Given an ABW $\mathcal{A}$ we denote by $\mathsf{nbw}(\mathcal{A})$ an NBW $\mathcal{B}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(B)$, which exists by Theorem 3.3.

We next present our model-checking algorithm for MultiLTL.

Our model-checking algorithm takes a multi-model $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle$, where $\mathcal{M}_i = (S_i, I_i, R_i, L_i)$ and a $\mathsf{MultiLTL}_{\mathsf{NNF}}$ formula $\mathbb{P} = \mathbb{Q}_1^1 \pi_1, \ldots, \mathbb{Q}_k^k \pi_k . \varphi(\pi_1, \ldots, \pi_k)$, and constructs an ABW $\mathcal{A}_\mathbb{P}$ over $\Sigma = S_1 \times S_2 \times \cdots \times S_k$, such that a word $\bar{w} \in \Sigma$ is in its language, iff the traces that correspond to the states traversed by the run of $\mathcal{A}_\mathbb{P}$ on $\bar{w}$ satisfy the multi-property $\mathbb{P}$.

Let $\Sigma_i = S_1 \times S_2 \times \cdots \times S_i$. Note that $\Sigma = \Sigma_k$ and Σ_0 is a unary alphabet. Given a i-tuple $\bar{s} \in \Sigma_i$ where $\bar{s} = (s_1, \ldots, s_i)$, we use $\bar{s}|_j$ to denote s_j. Given tuples $\bar{s}_1 = (s_1, \ldots, s_n), \bar{s_2} = (s'_1, \ldots, s'_m)$, we use $\langle \bar{s}_1, \bar{s}_2 \rangle = (s_1, \ldots, s_n, s'_1, \ldots, s'_m)$ to denote the $(n+m)$-tuple of their concatenation.

We first describe a construction of an ABW for the quantifier-free formula $\varphi(\pi_1, \ldots, \pi_k)$. The construction is by induction on the structure of the formula, as described below. Assume that $\mathcal{A}_{\psi_i} = (Q_i, q_0^i, \Sigma, \delta_i, \alpha_i)$ for $i \in \{1, 2\}$, are the ABWs for the subformulae ψ_1 and ψ_2.

- If $\psi = a_{\pi_k}$:
  $\mathcal{A}_\psi = (\{q_0\}, q_0, \Sigma, \delta, \emptyset)$ where $\delta(q_0, \bar{s}) =$ **true** if $a \in L_k(\bar{s}|_k)$ and $\delta(q_0, \bar{s}) =$ **false** otherwise.
- If $\psi = \neg a_{\pi_k}$:
  $\mathcal{A}_\psi = (\{q_0\}, q_0, \Sigma, \delta, \emptyset)$ where $\delta(q_0, \bar{s}) =$ **false** if $a \in L_k(\bar{s}|_k)$ and $\delta(q_0, \bar{s}) =$ **true** otherwise.
- If $\psi = \psi_1 \circ \psi_2$, where $\circ \in \{\vee, \wedge\}$:
  $\mathcal{A}_\psi = (Q_1 \uplus Q_2 \uplus \{q_0\}, q_0, \Sigma, \delta, \alpha_1 \uplus \alpha_2)$ where $\delta(q_0, \bar{s}) = \delta_1(q_0^1, \bar{s}) \circ \delta_2(q_0^2, \bar{s})$, and $\delta(q, \bar{s}) = \delta_i(q, \bar{s})$ for every $q \in Q_i$ and $i \in \{1, 2\}$.
- If $\psi = \mathsf{X}\psi_1$:
  $\mathcal{A}_\psi = (Q_1 \uplus \{q_0\}, q_0, \Sigma, \delta, \alpha_1)$, where $\delta(q_0, \bar{s}) = q_0^1$, and $\delta(q, \bar{s}) = \delta_1(q, \bar{s})$ for every $q \in Q_1$.
- If $\psi = \psi_1 \mathsf{U} \psi_2$:
  $\mathcal{A}_\psi = (Q_1 \uplus Q_2 \uplus \{q_0\}, q_0, \Sigma, \delta, \alpha_1 \uplus \alpha_2)$, where $\delta(q_0, \bar{s}) = \delta_2(q_0^2, \bar{s}) \vee (\delta_1(q_0^1, \bar{s}) \wedge q_0)$, and $\delta(q, \bar{s}) = \delta_i(q, \bar{s})$for every $q \in Q_i$.

- If $\psi = \psi_1 \mathsf{R} \psi_2$:
  $\mathcal{A}_\psi = (Q_1 \uplus Q_2 \uplus \{q_0\}, q_0, \Sigma, \delta, \alpha_1 \uplus \alpha_2 \uplus \{q_0\})$, where $\delta(q_0, \bar{s}) = \delta_2(q_0^2, \bar{s}) \wedge (\delta_1(q_0^1, \bar{s}) \vee q_0)$, and $\delta(q, \bar{s}) = \delta_i(q, \bar{s})$ for every $q \in Q_i$.

Now, we describe how to handle quantifiers. When we introduce a quantifier, we restrict the alphabet of the automaton. Specifically, when introducing the i$^{\text{th}}$ quantifier to the formula $\psi = \exists^i \pi.\psi_1$, we restrict the alphabet of the automaton from $\Sigma_{\psi_1} = \Sigma_i$ to $\Sigma_\psi = \Sigma_{i-1}$. The idea behind this is that the states for the i$^{\text{th}}$ trace, are hidden inside the automaton, and are not given as its input. We follow a path p in $\mathcal{M}_i$, by tracking the states it follows, using the new component in the tuple that represents a state from S_i. In this way, when we reach an automaton for the entire formula, we can follow the first existentially quantified traces in the constructed automaton.

- If $\psi = \exists^i \pi_i.\psi_1$:
  Let $\mathsf{nbw}(\mathcal{A}_{\psi_1}) = (Q', q_0', \Sigma_{\psi_1}, \delta', \alpha')$ be the NBW that is equivalent to $\mathcal{A}_{\psi_1}$, and let $\mathcal{M}_i = (S_i, I_i, R_i, L_i)$ be the i$^{\text{th}}$ model in $\mathbb{M}$.

  $\mathcal{A}_\psi = ((Q' \times S_i) \uplus \{q_0\}, q_0, \Sigma_\psi, \delta, \alpha' \times S_i)$, where

$$\delta(q_0, \bar{s}) = \{(q', s') \mid q' \in \delta'(q_0', \langle \bar{s}, s_i \rangle),\ (s_i, s') \in R_i \text{ and } s_i \in I_i\}$$
$$\delta((q, t), \bar{s}) = \{(q', s') \mid q' \in \delta'(q, \langle \bar{s}, t \rangle) \text{ and } (t, s') \in R_i\}$$

  After this construction is applied to the first quantifier $\mathbb{Q}_1^1$, the automaton $\mathcal{A}_\mathbb{P}$ is over a unary alphabet Σ_0.

For the universal quantifiers we use the fact that NBWs are closed for complementation [41]. Thus, for $\psi = \neg\psi_1$, we can use $\overline{\mathsf{nbw}(\mathcal{A}_{\psi_1})}$.

- If $\psi = \forall^i \pi_i.\psi_1$:
  Then ψ is equivalent to $\neg\exists^i \pi_i.\neg\psi_1$, and we set $\mathcal{A}_\psi = \overline{\mathsf{nbw}\,(\exists^i \pi_i.\neg\psi_1)}$.

Similarly to [35], when $\mathbb{P}$ begins with an existential quantifier, $\mathbb{M} \models \mathbb{P}$ iff $\mathcal{L}(\mathcal{A}_\mathbb{P})$ is *not empty*. The correctness follows directly from the correctness of the original algorithm.

3.3.1 Counterexamples from the Model-Checking Algorithm

In this section, we describe how counterexamples can be obtained from the aforementioned model-checking algorithm, and for which formulae. Let $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle$ be a multi-model, and $\mathbb{T} = \mathcal{L}(\mathbb{M})$.

We first consider a MultiLTL formula $\mathbb{P} = \mathbb{Q}_1^1 \pi_1 \ldots, \mathbb{Q}_k^k \pi_k.\varphi$, which begins with a sequence of n existential trace quantifiers. The *witness for the satisfaction* is a tuple $\langle w_1, \ldots, w_n \rangle$, such that for every trace assignment Π, if $\Pi(\pi_i) = w_i$ for $i \in [1, n]$, then $\Pi \models_\mathbb{T} \mathbb{Q}_{n+1}^{n+1} \pi_{n+1} \ldots, \mathbb{Q}_k^k \pi_k.\varphi$.

Since the construction of $\mathcal{A}_\mathbb{P}$ starts from the innermost sub-formulae, the n last quantifiers, for which the construction is applied are the first n quantifiers in the formula $\mathbb{Q}_1^1, \ldots, \mathbb{Q}_n^n$. This is due to the fact that we apply the construction of the quantifiers in a decreasing order – from $\mathbb{Q}_k$ towards $\mathbb{Q}_1$.

During each of the constructions for $\mathbb{Q}_n^n, \ldots, \mathbb{Q}_1^1$, the states of the automaton are labeled by the states of the models in $\mathbb{M}$, from which a trace is needed by the quantifier. This is done by the Cartesian product in the construction for existential quantifiers. Thus, each state in $\mathcal{A}_\mathbb{P}$ is a tuple $(q, s_n, \ldots, s_1)$, such that $s_i \in S_i$, for $i \in [1, n]$. Note that when considering the automaton $\mathcal{A}_\psi$ for $\psi = \mathbb{Q}_{n+1}^{n+1}\pi_{n+1} \ldots, \mathbb{Q}_k^k \pi_k.\varphi$, the states in the automaton do not correspond to states in the multi-model $\mathbb{M}$. This is due to the fact, that in the construction for universal quantifiers, a complementation is needed. Similarly, if there are no other quantifiers, the states in the automaton are independent from the states of the multi-model. Thus, the states of $\mathcal{A}_\mathbb{P}$ can be viewd as an $(n+1)$-tuple of states, where the first component is a state in the complemented NBW $\mathcal{A}_\psi$. Due to the nature of the complementation construction, this component does not correspond to a path in the previously quantified models. Therefore, we can obtain such paths only for the first (outermost) sequence of existential quantifiers.

Consequently, when $\mathbb{M} \models \mathbb{P}$, the language of $\mathcal{A}_\mathbb{P}$ is not empty. Thus, we get a non-emptiness witness, which is an infinite path $(q^0, s_n^0, \ldots, s_1^0), (q^1, s_n^1, \ldots, s_1^1), \ldots$, which corresponds to a word $w \in \mathcal{L}(\mathcal{A}_\mathbb{P})$. Each path $p_i = s_i^0, s_i^1, \ldots$ is a path in $\mathcal{M}_i$, for $i \in [1, n]$. By setting each π_i to p_i, for $i \in [1, n]$, the formula $\mathbb{Q}_{n+1}^{n+1}\pi_{n+1} \ldots \mathbb{Q}_k^k \pi_k.\varphi$ can be satisfied.

Symmetrically, if $\mathbb{P}$ begins with n universal trace quantifiers. When $\mathbb{M} \not\models \mathbb{P}$, we aim to obtain a counterexample. Since $\mathbb{M} \not\models \mathbb{P}$ iff $\mathbb{M} \models \neg\mathbb{P}$, we can obtain a witness for the satisfiability of $\neg\mathbb{P}$, as described before. This witness a *counterexample* for the satisfaction of $\mathbb{P}$ by $\mathbb{M}$. This means that a *counterexample* is a tuple $\langle w_1, \ldots, w_n \rangle$, such that for every trace assignment Π, if $\Pi(\pi_i) = w_i$ for $i \in [1, n]$, then $\Pi \not\models_\mathbb{T} \mathbb{Q}_{n+1}^{n+1}\pi_{n+1} \ldots, \mathbb{Q}_k^k \pi_k.\varphi$.

To summarize, we have the following.

Lemma 3.3.5. *1. There is a direct algorithm for model-checking $\mathbb{M} \models \mathbb{P}$.*

2. Let $\mathbb{P} \in$ MultiLTL with k quantifiers such that $\mathbb{Q}_i = \forall$ for every $i \in [1, n]$, let Π be an assignment that respects $\mathbb{M}$ and $\mathbb{T} = \mathcal{L}(\mathbb{M})$. If $\mathbb{M} \not\models \mathbb{P}$ then the model-checking algorithm can also extract a counterexample $\langle w_1, \ldots, w_n \rangle$ such that $w_i \in \mathcal{L}(\mathcal{M}_i)$ for every $i \in [1, n]$. Additionally, for every trace assignment Π such that $\Pi(\pi_i) = w_i$ for $i \in [1, n]$, it holds that $\Pi \not\models_\mathbb{T} \mathbb{Q}_{n+1}^{n+1}\pi_{n+1} \ldots \mathbb{Q}_k^k \pi_k.\varphi$.

Intuitively, when $\mathbb{Q}_{n+1} = \cdots = \mathbb{Q}_k = \exists$, (2) means that there is no extension of $\langle w_1, \ldots, w_n \rangle$ for k traces, from the appropriate models, that satisfies the formula.

Note 3.3.6. Although the model checking algorithm works for every quantification condition, counterexamples are defined only for the first sequence of universal quantifiers. Since, for existential quantifiers, there is no natural counterexample in the form of a single word. Indeed, a counterexample for this case would need to convince the lack of existence of an appropriate word.

3.4 Compositional Proof Rules for Model-Checking MultiLTL

We present two complementing compositional proof rules for the model-checking problem of MultiLTL.

Let $\mathbb{M}$ be a k-multi-model, and let $\mathbb{P} = \mathbb{Q}_1^{i_1}\pi_1 \dots \mathbb{Q}_m^{i_m}\pi_m\varphi$ be a MultiLTL formula. The rule (PR) aims at proving $\mathbb{M} \models \mathbb{P}$, and ($\overline{\text{PR}}$) aims at proving the contrary, that is, $\mathbb{M} \models \neg\mathbb{P}$. Every model $\mathcal{A}_i$ in the rules is an *abstraction*. Since some models may be multiply quantified, a model $\mathcal{M}_i$ may have several different abstractions, according to the quantifiers under which $\mathcal{M}_i$ appears in $\mathbb{P}$.

$$\frac{\forall i \in I_\forall.\ \mathcal{M}_{i_j} \models \mathcal{A}_i \quad \forall i \in I_\exists.\ \mathcal{A}_i \models \mathcal{M}_{i_j} \quad \langle \mathcal{A}_1, \dots, \mathcal{A}_m \rangle \models \mathbb{Q}_1^1\pi_1 \dots \mathbb{Q}_m^m\pi_m.\varphi}{\langle \mathcal{M}_1, \dots, \mathcal{M}_k \rangle \models \mathbb{Q}_1^{i_1}\pi_1 \dots \mathbb{Q}_m^{i_m}\pi_m.\varphi} \tag{PR}$$

$$\frac{\forall i \in I_\forall.\ \mathcal{A}_i \models \mathcal{M}_{i_j} \quad \forall i \in I_\exists.\ \mathcal{M}_{i_j} \models \mathcal{A}_i \quad \langle \mathcal{A}_1, \dots, \mathcal{A}_m \rangle \models \neg(\mathbb{Q}_1^1\pi_1 \dots \mathbb{Q}_m^m\pi_m.\varphi)}{\langle \mathcal{M}_1, \dots, \mathcal{M}_k \rangle \models \neg(\mathbb{Q}_1^{i_1}\pi_1 \dots \mathbb{Q}_m^{i_m}\pi_m.\varphi)} \tag{$\overline{\text{PR}}$}$$

Intuitively, in (PR), we use an over-approximation for every model under $\forall$, and an under-approximation for every model under $\exists$. The rule ($\overline{\text{PR}}$) behaves dually to (PR) for the negation of $\mathbb{P}$.

Lemma 3.4.1. *The proof rules (PR) and ($\overline{\text{PR}}$) are sound and complete.*

Proof. We first prove for (PR). Let $\mathbb{M} = \langle \mathcal{M}_1, \dots, \mathcal{M}_k \rangle$ and $\mathbb{P} = \mathbb{Q}_1^{i_1}\pi_1 \dots \mathbb{Q}_m^{i_m}\pi_m.\varphi(\pi_1, \dots, \pi_m)$.
Completeness: When $\mathbb{M} \models \mathbb{P}$, we can choose $\mathcal{A}_i = \mathcal{M}_{i_j}$ for every $i \in [1, m]$, which satisfy all the requirements.

Soundness: Let $\mathbb{A} = \langle \mathcal{A}_1, \dots, \mathcal{A}_m \rangle$ such that $\mathbb{M}, \mathbb{A}$ and $\mathbb{P}$ fulfill the premises of the proof rule. Let Π be an assignment for the trace variables, and ψ a sub-formula of $\mathbb{P}$ that contains φ. Denote $\mathbb{T} = \langle \mathcal{L}(\mathcal{M}_1), \dots, \mathcal{L}(\mathcal{M}_k) \rangle$ and $\mathbb{T}' = \langle \mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_k) \rangle$.

We show soundness by induction on the number of quantifiers in sub-formulae of $\mathbb{P}$, that if $\Pi \models_{\mathbb{T}'} \psi$ then $\Pi \models_{\mathbb{T}} \psi$.

Base: ψ is quantifier free, this means that $\psi = \varphi$. Therefore $\Pi \models_{\mathbb{T}'} \psi$ iff $\Pi \models_{\mathbb{T}} \psi$, since when there are no quantifiers, the domain does not affect the satisfaction.

Step: Let $\psi = \mathbb{Q}_n^{i_n}\pi.\psi'$ be a formula with $m - n + 1$ quantifiers, and ψ' be its sub-formula with $m - n$ quantifiers. By the induction hypothesis we know that if $\Pi \models_{\mathbb{T}'} \psi'$ then $\Pi \models_{\mathbb{T}} \psi'$.

- If $\mathbb{Q}_n^{i_n} = \exists^i$, then $\Pi \models_{\mathbb{T}'} \exists^i\pi.\psi'$ iff there exists $t \in \mathcal{L}(\mathcal{A}_i)$ such that $\Pi[\pi \to t] \models_{\mathbb{T}'} \psi'$. Since $\mathcal{A}_i \models \mathcal{M}_i$, the same t is also in $\mathcal{L}(\mathcal{M}_i)$. Therefore, there exists $t \in \mathcal{L}(\mathcal{M}_i)$ such that $\Pi[\pi \to t] \models_{\mathbb{T}'} \psi'$ which by the induction hypothesis means that there exists $t \in \mathcal{L}(\mathcal{M}_i)$ such that $\Pi[\pi \to t] \models_{\mathbb{T}} \psi'$. By the semantics of MultiLTL, we get that $\Pi \models_{\mathbb{T}} \exists^i\pi.\psi'$.

- If $\mathbb{Q}_n^{i_n} = \forall^i$, then $\Pi \models_{\mathbb{T}'} \forall^i\pi.\psi'$ iff for every $t \in \mathcal{L}(\mathcal{A}_i)$, it holds that $\Pi[\pi \to t] \models_{\mathbb{T}'} \psi'$. Since $\mathcal{M}_i \models \mathcal{A}_i$, for every $t \in \mathcal{L}(\mathcal{M}_i)$, it holds $\Pi[\pi \to t] \models_{\mathbb{T}'} \psi'$. By the induction hypothesis, this means that for every $t \in \mathcal{L}(\mathcal{M}_i)$, it holds that $\Pi[\pi \to t] \models_{\mathbb{T}} \psi'$. By the semantics of MultiLTL, we get that $\Pi \models_{\mathbb{T}} \forall^i\pi.\psi'$.

When $\psi = \mathbb{P}$, we get that if $\mathbb{A} \models \mathbb{P}$ then $\mathbb{M} \models \mathbb{P}$ as needed.

For ($\overline{\text{PR}}$), notice that $\neg\mathbb{P} \equiv \overline{\mathbb{Q}}_1^{i_1}\pi_1 \dots \overline{\mathbb{Q}}_1^{i_m}\pi_m\neg\varphi$, where $\overline{\forall} = \exists$ and $\overline{\exists} = \forall$, conforming to (PR). ∎

Chapter 4

Abstraction-Refinement Based Implementation of the Proof Rules

In this chapter, we present methods for constructing over- and under-approximations using an abstraction-refinement based approach. We first define the notion of *simulation*.

Definition 4.0.1. Let $\mathcal{M}_1 = (S_1, I_1, R_1, L_1)$ and $\mathcal{M}_2 = (S_2, I_2, R_2, L_2)$ be Kripke structures over AP. A *simulation from* $\mathcal{M}_1$ *to* $\mathcal{M}_2$ is a relation $H \subseteq S_1 \times S_2$ such that for every $(s_1, s_2) \in H$:

- $L_1(s_1) = L_2(s_2)$
- For every $(s_1, s_1') \in R_1$, there exists a state $s_2' \in S_2$ such that $(s_2, s_2') \in R_2$ and $(s_1', s_2') \in H$.

If additionally, for every $s_0 \in I_1$ there exists $s_0' \in I_2$ such that $(s_0, s_0') \in H$, we denote $\mathcal{M}_1 \leq_H \mathcal{M}_2$. We denote $\mathcal{M}_1 \leq \mathcal{M}_2$ if $\mathcal{M}_1 \leq_H \mathcal{M}_2$ for some simulation H.

The following Lemma is a well-known property of a simulation relation.

Lemma 4.0.2. *Let $\mathcal{M}_1, \mathcal{M}_2$ be two Kripke structures such that $\mathcal{M}_1 \leq \mathcal{M}_2$. Then $\mathcal{M}_1 \models \mathcal{M}_2$.*

Next, we describe how to construct sequences of over- and under-approximations for a given model $\mathcal{M}$. Each approximation in these sequences is closer to the original model than its previous. We later incorporate these sequences in a MultiLTL abstraction-refinement based model-checking algorithm using our proof rules.

4.1 Constructing a Sequence of Over-Approximations

Given a Kripke structure $\mathcal{M} = (S, I, R, L)$ over AP, we aim to construct a sequence of over-approximations $\mathcal{A}_0 \geq \mathcal{A}_1 \geq \cdots \mathcal{A}_k \geq \mathcal{M}$, where $\mathcal{A}_{i+1}$ is a *refinement* of $\mathcal{A}_i$, which we calculate by using counterexamples. A *counterexample* is a word $w \in \mathcal{L}(\mathcal{A}_i)$ yet $w \notin \mathcal{L}(\mathcal{M})$. By Lemma 2.0.5, it suffices to consider finite prefixes of w, since there is an index j for which $w_0, w_1, \ldots, w_{j-1} \in \mathcal{L}(\mathcal{A}_i) \setminus \mathcal{L}(\mathcal{M})$.

We use a sequence of *abstraction functions* $h_0, \ldots, h_k$, each defining an abstract model.

Definition 4.1.1. Let $\hat{S}$ be a finite set of abstract states. A function $h : S \to \hat{S}$ is an *abstraction function* if h is onto, and for every $\hat{s} \in \hat{S}$, it holds that $L(s_1) = L(s_2)$ for every $s_1, s_2 \in h^{-1}(\hat{s})$.

Definition 4.1.2. For an abstraction function $h : S \to \hat{S}$, the $\exists\exists$ *abstract model induced by* h is $\mathcal{A}_h = (\hat{S}, \hat{I}, \hat{R}, \hat{L})$, where $\hat{I} = \{\hat{s} \mid \exists s_0 \in I, h(s_0) = \hat{s}\}$, and for every $\hat{s} \in \hat{S}$ we set $\hat{L}(\hat{s}) = L(s)$ for some s such that $h(s) = \hat{s}$.[1] However, $(\hat{s}, \hat{s}') \in \hat{R}$ iff there exist $s, s' \in S$ such that $(s, s') \in R$ for which $h(s) = \hat{s}$ and $h(s') = \hat{s}'$.

Lemma 4.1.3. *Let $\mathcal{M} = (S, I, R, L)$ be a Kripke structure and $\mathcal{A}_h = (\hat{S}, \hat{I}, \hat{R}, \hat{L})$ be the $\exists\exists$ abstract model induced by an abstraction function $h : S \to \hat{S}$. Then, $\mathcal{M} \leq \mathcal{A}_h$.*

Proof. Let $H = \{(s, h(s)) | s \in S\}$. We show that H is a simulation relation. Let $(s, h(s)) \in H$.

1. $L(s) = \hat{L}(h(s))$ by the definition of abstraction function.
2. Let $(s, s') \in R$, we show that $(h(s), h(s')) \in \hat{R}$. By definition $(h(s), h(s')) \in \hat{R}$ iff there exist $s_{\mathcal{M}}, s'_{\mathcal{M}} \in S$ such that $(s_{\mathcal{M}}, s'_{\mathcal{M}}) \in R$ and $(s_{\mathcal{M}}, h(s_{\mathcal{M}})) \in H$ and $(s'_{\mathcal{M}}, h(s')) \in H$. Clearly for $s = s_{\mathcal{M}}$ and $s' = s'_{\mathcal{M}}$ this requirement is fulfilled.
3. Let $s_0 \in I$, by the construction of the abstract model, we know that $h(s_0) \in \hat{I}$ as needed.

■

Definition 4.1.4. Let $\mathcal{M}$ and $\mathcal{M}'$ be Kripke structures such that $\mathcal{M} \leq \mathcal{M}'$ by a simulation H, and let $r' = s'_0, s'_1, \ldots$ be a run of $\mathcal{M}'$ on w. The run $r = s_0, s_1, \ldots, s_j$ is a *maximal induced run* of r' in $\mathcal{M}$, if for every $i \in [0, j]$ it holds that $(s_i, s'_i) \in H$, and for every $i \in [0, j-1]$ it holds that $(s_i, s_{i+1}) \in R$. Moreover, there is no state $s^* \in S$ such that $(s^*, s'_{j+1}) \in H$ and $(s_j, s^*) \in R$. If no such j exists then r is infinite, and for every $i \geq 0$ it holds that $(s_i, s'_i) \in H$ and $(s_i, s_{i+1}) \in R$.

In the sequel, we fix a Kripke structure $\mathcal{M} = (S, I, R, L)$.

4.1.1 Over-approximation Sequence Construction

Initialization.

Define $\hat{S}_0 = \{s_P \mid P \subseteq AP \text{ and } \exists s \in S : \ L(s) = P\}$. That is, there is a state in $\hat{S}_0$ for every labeling in $\mathcal{M}$. The initial over-approximation $\mathcal{A}_0$ is the $\exists\exists$ model induced by $h_0 : S \to \hat{S}_0$ defined by $h_0(s) = s_{L(s)}$. Since h_0 is an abstraction function, by Lemma 4.1.3 we have that $\mathcal{M} \leq \mathcal{A}_0$.

Refinement.

Let $h_i : S \to \hat{S}_i$ be an abstraction function. Let $\mathcal{A}_i = (\hat{S}_i, \hat{I}_i, \hat{R}_i, \hat{L}_i)$ be the $\exists\exists$ model induced by h_i. By Lemma 4.1.3 we have that $\mathcal{M} \leq \mathcal{A}_i$. Let $w \in \mathcal{L}(\mathcal{A}_i) \backslash \mathcal{L}(\mathcal{M})$ be a counterexample. Let $\hat{r}_i = \hat{s}_0, \hat{s}_1 \ldots$ be a run of $\mathcal{A}_i$ on w, and $r = s_0 \ldots, s_j$ be a maximal induced run of $\mathcal{M}$ on w. Since $w \notin \mathcal{L}(\mathcal{M})$, we have that r is finite. We define $\mathcal{A}_{i+1}$ to be the $\exists\exists$ model induced by h_{i+1}, where $h_{i+1} : S \to \hat{S}_{i+1}$ for $\hat{S}_{i+1} = \hat{S}_i \uplus \{\hat{s}'\}$, defined as follows, for every $s \in S$.

[1] $\hat{L}$ is well defined since by Definition 4.1.1, only equilabeled states are mapped to the same abstract state.

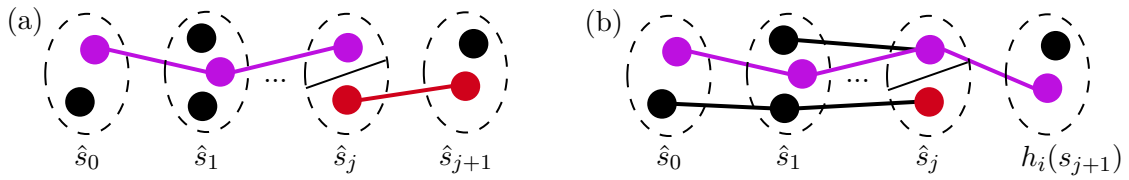

Figure 4.1: Illustration of Refinements: (a) $\exists\exists$, and (b) $\forall\exists$

.

$$h_{i+1}(s) = \begin{cases} h_i(s), & \text{if } h_i(s) \neq \hat{s}_j \\ h_i(s), & \text{if } h_i(s) = \hat{s}_j \text{ and } \exists s' \in S \text{ such that } h_i(s') = \hat{s}_{j+1} \text{ and } (s, s') \in R \\ \hat{s}', & \text{if } h_i(s) = \hat{s}_j \text{ and } \neg\exists s' \in S \text{ such that } h_i(s') = \hat{s}_{j+1} \text{ and } (s, s') \in R \end{cases}$$

The intuition for the refinement is presented in Figure 4.1 (a). Concrete states are the full circles and abstract states are the dashed ovals. The purple line is a maximal induced run of $\hat{s}_0, \hat{s}_1 \ldots$ in $\mathcal{M}$, which ends at $\hat{s}_j$. Since there is an infinite run in the abstract model, we can split $\hat{s}_j$ into two abstract states: one that includes all states that can continue to $\hat{s}_{j+1}$, and another that includes all the states with no such transitions. Clearly, the former set includes only states that are not reachable by the maximal induced run of $\hat{s}_0, \hat{s}_1, \ldots$, else the induced run would not have been maximal.

Lemma 4.1.5. *For every $i \in \mathbb{N}$, for every state $\hat{s} \in \hat{S}_i$, there exists a state $s \in S$ such that $h_i(s) = \hat{s}$.*

Proof. By induction on i.
Base: By construction, for every $\hat{s}_P \in \hat{S}_0$ there exists a state $s \in S$ such that $L(s) = P$, and so $h_0(s) = \hat{s}_P$.
Step: Assume towards contradiction that there is an abstract state $\hat{s} \in \hat{S}_{i+1}$ such that for every $s \in S$, it holds that $h_{i+1}(s) \neq \hat{s}$. Since $\mathcal{A}_i$ fulfills the required property, $\hat{s} \notin \hat{S}_i$. Then $\hat{s}$ is the new state $\hat{s}'$. Let $s_0, \ldots, s_j$ be a maximal induced run of $\mathcal{M}$ on the counterexample w. There is no state $s' \in h_i^{-1}(\hat{s}_{j+1})$ such that $(s_j, s') \in R$. Thus, by construction, $h_{i+1}(s_j) = \hat{s}'$, a contradiction. ■

The model $\mathcal{A}_{i+1}$ obtained from h_{i+1} is a refinement of $\mathcal{A}_i$, as stated in the following lemma:

Lemma 4.1.6. *For every $i \geq 0$, it holds that $\mathcal{M} \leq \mathcal{A}_{i+1} \leq \mathcal{A}_i$*

Proof. According to Lemma 4.1.3, it is left to show is that $\mathcal{A}_{i+1} \leq \mathcal{A}_i$. Consider the relation $H \subseteq \hat{S}_{i+1} \times \hat{S}_i$ defined by $H = \{(\hat{s}, \hat{s}') \mid h_{i+1}^{-1}(\hat{s}) \subseteq h_i^{-1}(\hat{s}')\}$. We show that H is a simulation relation. Let $(\hat{s}, \hat{s}') \in H$:

1. Since h_i and h_{i+1} are abstraction functions, we know that for every concrete states $s_1, s_2 \in h_i^{-1}(\hat{s}')$, it holds that $L(s_1) = L(s_2) = L(\hat{s}')$. Also for every states $s_1, s_2 \in h_{i+1}^{-1}(\hat{s})$, it holds that $L(s_1) = L(s_2) = L(\hat{s})$. Since $h_{i+1}^{-1}(\hat{s}) \subseteq h_i^{-1}(\hat{s}')$, we get that $L(\hat{s}) = L(\hat{s}')$.

2. Let $\hat{s}_1 \in \hat{S}_{i+1}$ such that $(\hat{s}, \hat{s}_1) \in \hat{R}_{i+1}$. This means that there are $s, s_1 \in S$ such that $h_{i+1}(s) = \hat{s}, h_{i+1}(s_1) = \hat{s}_1$ and $(s, s_1) \in R$. By construction $\hat{s}'_1 = h_i(s_1) \supseteq h_{i+1}(s_1) = \hat{s}_1$, thus $(\hat{s}_1, \hat{s}'_1) \in H$ as needed. Additionally, $(\hat{s}', \hat{s}'_1) \in \hat{R}_i$ by the definition of $\exists\exists$ abstract structure.

3. Let $\hat{s} \in \hat{I}_{i+1}$. By definition, there exists a state $s \in I$ such that $h_{i+1}(s) = \hat{s}$. By construction of the abstract model, $h_i(s) \supseteq h_{i+1}(s)$, which means that $h_i(s) \in \hat{I}_i$ and also $(\hat{s}, h_{i+1}(s)) \in H$. ■

Following Lemma 4.1.6, we have that $\mathcal{M} \leq \cdots \leq \mathcal{A}_1 \leq \mathcal{A}_0$. Thus, the refinements get more precise with every refinement step. Moreover, for $i > 0$, the model $\mathcal{A}_i$ is obtained from $\mathcal{A}_{i-1}$ by splitting a state. In a finite-state setting, this guarantees termination at the latest when reaching $\mathcal{A}_i = \mathcal{A}_{i+1}$.

Lemma 4.1.7. *Let $\mathcal{M}$ be a Kripke structure and let $\mathcal{A}_0 \geq \mathcal{A}_1 \cdots \geq \mathcal{M}$ be our sequence of over-approximations. Then, there exists $m \in \mathbb{N}$ for which $\mathcal{L}(\mathcal{A}_m) = \mathcal{L}(\mathcal{M})$.*

Proof. In each refinement step i, the approximation $\mathcal{A}_i$ is refined. In the process, an abstract state is split into two abstract states. According to Lemma 4.1.5, there is a concrete state mapped to every abstract state $\hat{s}$. When $|\hat{S}_i| = |S|$, every abstract state has exactly one concrete state mapped to it. Thus, by the definition of $\exists\exists$ induced model, we achieve a model isomorphic to $\mathcal{M}$. The process terminates when $|\hat{S}_i| = |S|$ or before. If it terminates beforehand, then an abstraction $\mathcal{A}_m$ was reached, which does not require another refinement. This means that there is no $w \in \mathcal{L}(\mathcal{A}_m) \setminus \mathcal{L}(\mathcal{M})$. Therefore, $\mathcal{L}(\mathcal{A}_m) \not\subset \mathcal{L}(\mathcal{M})$, and since $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{A}_m)$, we get that $\mathcal{L}(\mathcal{A}_m) = \mathcal{L}(\mathcal{M})$. ■

4.2 Constructing a Sequence of Under-Approximations

Given $\mathcal{M} = (S, I, R, L)$ over AP, we construct a sequence of under-approximations $\mathcal{A}_0, \mathcal{A}_1, \ldots$ such that $\mathcal{A}_i \leq \mathcal{M}$ for every $i \in \mathbb{N}$, via a sequence of abstraction functions using counterexamples. In this case, a counterexample is a word $w \notin \mathcal{L}(\mathcal{A})$, yet $w \in \mathcal{L}(\mathcal{M})$. Again, we can consider a prefix of w.

Definition 4.2.1. Given an abstraction function $h : S \to \hat{S}$, the $\forall\exists$ *abstract model induced by* h is $\mathcal{A}_h = (\hat{S}, \hat{I}, \hat{R}, \hat{L})$, where $\hat{I}$ and $\hat{L}$ are as in Definition 4.1.2, and $(\hat{s}, \hat{s}') \in \hat{R}$ iff for every $s \in S$ such that $h(s) = \hat{s}$ there exists $s' \in S$ such that $(s, s') \in R$ and $h(s') = \hat{s}'$.

Notice that the transition relation $\hat{R}$ of the $\forall\exists$ abstract model might not be total, i.e., there may exist a state with no outgoing transitions.

Lemma 4.2.2. *Let $\mathcal{M} = (S, I, R, L)$ be a Kripke structure and $\mathcal{A}_h = (\hat{S}, \hat{I}, \hat{R}, \hat{L})$ be the $\forall\exists$ abstract model induced by an abstraction function $h : S \to \hat{S}$. Then, $\mathcal{A}_h \leq \mathcal{M}$.*

Proof. Define $H = \{(h(s), s) \mid s \in S\}$. We show that H is a simulation relation. Let $(h(s), s) \in H$.

1. $L(s) = \hat{L}(h(s))$ by the definition of abstraction function.

2. Let $(h(s), \hat{s}) \in \hat{R}$. By the definition of $\hat{R}$, for every $s_1 \in S$ such that $h(s_1) = h(s)$ there exists $s_2 \in S$ such that $(s_1, s_2) \in R$ and $h(s_2) = \hat{s}$. Specifically, for $s_1 = s$ and $s_2 = s' \in S$ this premise holds.

3. Let $s_0 \in I$, by the construction of the abstract model, we know that $h(s_0) \in \hat{I}$ as needed.

■

4.2.1 Under-approximation Sequence Construction

Initialization.

Let $\hat{S}_0$ and h_0 be as in Subsection 4.1.1. We set the initial under-approximation $\mathcal{A}_0$ of $\mathcal{M}$ to be the $\forall\exists$ abstract model induced by h_0. By Lemma 4.2.2, we have $\mathcal{A}_0 \leq \mathcal{M}$.

Refinement.

As in the construction of the over-approximations sequence, we use counterexamples to guide our refinement. Let $\mathcal{A}_i = (\hat{S}_i, \hat{I}_i, \hat{R}_i, \hat{L}_i)$ be an $\forall\exists$ abstract model such that $\mathcal{A}_i \leq \mathcal{M}$ by an abstraction function $h_i : S \to \hat{S}_i$. Let $w \in \mathcal{L}(\mathcal{M}) \setminus \mathcal{L}(\mathcal{A}_i)$ be a counterexample. Let $r = s_0, s_1, \ldots$ be a run of $\mathcal{M}$ on w, and let $\hat{r} = \hat{s}_0, \ldots, \hat{s}_j$ be a maximal induced run of $\mathcal{A}_i$ on w. We define $\mathcal{A}_{i+1}$ to be the $\forall\exists$ abstract model induced by $h_{i+1} : S \to \hat{S}_{i+1}$ where $\hat{S}_{i+1} = \hat{S}_i \uplus \{\hat{s}'\}$, and where:

$$h_{i+1}(s) = \begin{cases} h_i(s), & \text{if } h_i(s) \neq \hat{s}_j \\ h_i(s), & \text{if } h_i(s) = \hat{s}_j \text{ and } \exists s' \in S \text{ such that } h_i(s') = h_i(s_{j+1}) \text{ and } (s, s') \in R \\ \hat{s}', & \text{if } h_i(s) = \hat{s}_j \text{ and } \neg\exists s' \in S \text{ such that } h_i(s') = h_i(s_{j+1}) \text{ and } (s, s') \in R \end{cases}$$

The idea behind this refinement is represented in Figure 4.1 (b). The purple states and lines represent the run in $\mathcal{M}$. Note that in $\hat{s}_j$ there is a red state with no transition to states in $h_i(s_{j+1})$. Thus there is no $\forall\exists$ abstract transition from $\hat{s}_j$ to $h_i(s_{j+1})$. To add such a transition, we split $\hat{s}_j$ into two states: one with all states that have a transition to a state in $h_i(s_{j+1})$, and another with all states that have no such transition. As a result, $\mathcal{A}_{i+1}$ includes a $\forall\exists$ transition from $\hat{s}_j$ to $h_i(s_{j+1})$.

As a result of Lemma 4.2.2, and since h_i is an abstraction function for every $i \in \mathbb{N}$, we get that the sequence of under-approximations $\mathcal{A}_0, \mathcal{A}_1, \ldots$ fulfills that $\mathcal{A}_i \leq \mathcal{M}$, for every $i \in \mathbb{N}$. Note that this is different from the over-approximation case, since we do not guarantee that the abstractions get more precise after every iteration. This property does not hold, since by splitting an abstract state $\hat{s}$, transitions entering $\hat{s}$ in $\mathcal{A}_i$ might not be present in $\mathcal{A}_{i+1}$. However, since the number of states increases by each refinement, we can still guarantee termination, in a similar manner to over-approximations.

Lemma 4.2.3. *For every $i \in \mathbb{N}$, for every state $\hat{s} \in \hat{S}_i$ there exists a state $s \in S$ such that $h_i(s) = \hat{s}$.*

Proof. **Base:** By construction, for every $\hat{s}_P \in \hat{S}_0$ there exists a state $s \in S$ such that $L(s) = P$, which means that $h_0(s) = \hat{s}_P$.

Algorithm 4.1 Abstraction-refinement based MultiLTL model-checking

Input: $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_n \rangle$, $\mathbb{P} = \mathbb{Q}_1^1 \pi_1 \ldots \mathbb{Q}_n^n \pi_n . \varphi(\pi_1, \ldots, \pi_n)$
Output: $\mathbb{M} \models \mathbb{P}$?

```
1: 𝔸, 𝔹 = INITIALIZE(𝕄, ℙ)
2: while true do
3:    res = MMC(𝔸, ℙ)
4:    if res == true then
5:       return 𝕄 ⊨ ℙ
6:    else
7:       ⟨w_1, ..., w_n⟩ = GET_CEX(𝔸, 𝕄, PR)
8:       𝔸 = REFINE(⟨w_1, ..., w_n⟩, 𝔸)
9:    end if
10:   res = MMC(𝔹, ¬ℙ)
11:   if res == true then
12:      return 𝕄 ⊭ ℙ
13:   else
14:      ⟨w_1, ..., w_n⟩ = GET_CEX(𝔹, 𝕄, PR̄)
15:      𝔹 = REFINE(⟨w_1, ..., w_n⟩, 𝔹)
16:   end if
17: end while
```

Step: Assume towards contradiction that there is an abstract state $\hat{s} \in \hat{S}_{i+1}$ such that for every $s \in S$, $h_{i+1}(s) \neq \hat{s}$. Since $\mathcal{A}_i$ fulfills the required property, $\hat{s} \notin \hat{S}_i$. This means that $\hat{s}$ is the new added state $\hat{s}'$. Let $s_0, \ldots, s_j$ be a maximal induced run of $\mathcal{M}$ on a word $w \in \mathcal{L}(\mathcal{A}_i)$, which caused the refinement to $\mathcal{A}_{i+1}$. Denote the run of $\mathcal{A}_i$ on w by $\hat{s}_0, \hat{s}_1, \ldots$. Since a refinement was performed on $\mathcal{A}_i$, we get that there is some state $s \in h_i^{-1}(\hat{s}_j)$ and $s' \in S$ such that $(s, s') \notin R$ and $h_{i+1}(s') = h(s_{j+1})$. Thus, by construction of $\mathcal{A}_{i+1}$, we get that $h_{i+1}(s) = \hat{s}'$, which achieves the contradiction. ■

For finite models, this guarantees termination of the refinement, similarly to the over-approximation case.

Lemma 4.2.4. *Let $\mathcal{M}$ be a finite-state Kripke structure and let $\mathcal{A}_0 \leq \mathcal{A}_1 \cdots \leq \mathcal{M}$ be our sequence of under-approximations. Then, there exists $m \in \mathbb{N}$ for which $\mathcal{L}(\mathcal{A}_m) = \mathcal{L}(\mathcal{M})$.*

Proof. Identical to the proof of Lemma 4.1.7. ■

4.3 Abstraction-Refinement Guided Model-Checking

Following Section 4.1 and Section 4.2, we present an abstraction-refinement inspired approach for model-checking multi-properties. We are given a MultiLTL formula $\mathbb{P} = \mathbb{Q}_1^1 \pi_1 \ldots \mathbb{Q}_n^n \pi_n . \varphi$ and a multi-model $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_n \rangle$ over AP. The model-checking procedure for $\mathbb{M} \models \mathbb{P}$ is described in Algorithm 4.1, which we detail next.

The procedure MMC$(\mathbb{M}, \mathbb{P})$ performs model-checking as per Lemma 3.3.5 (1) and returns **true** if $\mathbb{M} \models \mathbb{P}$, and **false** otherwise. REFINE refines every approximation $\mathcal{A}_i$ for which there is a counterexample w_i in the vector $\langle w_1, \ldots, w_n \rangle$ of counterexamples.

Initialization.

In INITIALIZE (Line 1), for every model $\mathcal{M}_i$ such that $\mathbb{Q}_i^i = \forall$, we initialize abstract models $\mathcal{A}_i$ and $\mathcal{B}_i$ as described in Section 4.1 and Section 4.2, respectively. Each $\mathcal{A}_i$ is an over-approximation of the model $\mathcal{M}_i$, and each $\mathcal{B}_i$ is an under-approximation. For every model $\mathcal{M}_i$ such that $\mathbb{Q}_i^i = \exists$, we initialize abstract models $\mathcal{A}_i$ and $\mathcal{B}_i$ as described in Section 4.2 and Section 4.1, respectively. This means that one of them is an over-approximation and the other is an under-approximation for the same model. Thus, $\mathcal{B}_i \leq \mathcal{M}_i \leq \mathcal{A}_i$ for every $i \in I_\forall$ and $\mathcal{A}_i \leq \mathcal{M}_i \leq \mathcal{B}_i$ for every $i \in I_\exists$. In Algorithm 4.1, $\mathbb{A} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$ is used for (PR) and $\mathbb{B} = \langle \mathcal{B}_1, \ldots \mathcal{B}_n \rangle$ for ($\overline{\text{PR}}$).

Abstraction-refinement.

Lines 3-8 apply the rule (PR). When reaching line 3, it is guaranteed that $\mathcal{M}_i \leq \mathcal{A}_i$ for every $i \in I_\forall$ and $\mathcal{A}_i \leq \mathcal{M}_i$ for every $i \in I_\exists$. Thus, we try to apply (PR). We model-check $\mathbb{A} \models \mathbb{P}$ (Line 3). If the result is **true**, then by the correctness of (PR), we have $\mathbb{M} \models \mathbb{P}$ (Line 5). Otherwise, $\mathbb{A} \not\models \mathbb{P}$. As noted in Section 3.3, for $\mathcal{A}_i$ where $i \in I_\exists$, no single word counterexample can be obtained from the model-checking. Instead, we call GET_CEX (Line 7), which returns a sequence of words that lead to more precise abstractions. For (PR), GET_CEX returns an arbitrary $w_i \in \mathcal{L}(\mathcal{M}_i) \setminus \mathcal{L}(\mathcal{A}_i)$ for every $i \in I_\forall$ and an arbitrary $w_i \in \mathcal{L}(\mathcal{A}_i) \setminus \mathcal{L}(\mathcal{M}_i)$ for every $i \in I_\exists$. For ($\overline{\text{PR}}$), GET_CEX behaves dually on $\mathbb{B}$ for $I_\forall$ and $I_\exists$. If for some i such a word w_i does not exist, GET_CEX returns **null** as w_i. REFINE uses $\langle w_1, \ldots, w_n \rangle$ to refine each abstraction in $\mathbb{A}$ as described in Section 4.1 and Section 4.2, obtaining closer abstractions to the original models.

Lines 10-14 apply the rule ($\overline{\text{PR}}$). When we reach line 10, it is guaranteed that $\mathcal{B}_i \leq \mathcal{M}_i$ for every $i \in I_\forall$ and $\mathcal{M}_i \leq \mathcal{B}_i$ for every $i \in I_\exists$. Thus, we try to apply ($\overline{\text{PR}}$) in a similar manner as before. We model-check $\mathbb{B} \models \neg\mathbb{P}$. If the result is **true**, then by the correctness of ($\overline{\text{PR}}$), we have $\mathbb{M} \models \neg\mathbb{P}$ which implies $\mathbb{M} \not\models \mathbb{P}$. Otherwise, we call GET_CEX (Line 14) and refine $\mathbb{B}$ using $\langle w_1, \ldots, w_n \rangle$ (Line 15).

In the worst case, all approximations converge to their respective models (as per Lemma 4.1.7 and Lemma 4.2.4), upon which no further counterexamples are found. Therefore, the run is guaranteed to terminate. Of course, the run terminates much earlier in case that appropriate approximations are found.

Correctness follows from the correctness of (PR) and ($\overline{\text{PR}}$). Hence, we have the following.

Lemma 4.3.1. *Algorithm 4.1 terminates with the correct result.*

Proof. In each iteration, either we terminated with $\mathbb{M} \models \mathbb{P}$, $\mathbb{M} \not\models \mathbb{P}$ or find a counterexample. In the case of counterexample, at least one of the over-approximations or under-approximations is refined, either from $\mathbb{A}$ or $\mathbb{B}$. Since we cannot refine an abstract model $\mathcal{A}_i$ and $\mathcal{B}_i$ more than $|\mathcal{M}_i|$ times each, after at most $2 \cdot \sum_{i=1}^{n} |\mathcal{M}_i|$ refinements, $\mathcal{A}_i = \mathcal{M}_i$ and $\mathcal{B}_i = \mathcal{M}_i$ for every $i \in [1, n]$. Then, the output is as performing model-checking on the original models, since no counterexamples can be found. Thus, the answer at this point must either be $\mathbb{M} \models \mathbb{P}$ or $\mathbb{M} \not\models \mathbb{P}$ (Lines 5,12).

Correctness follows from the correctness of the model-checking procedure and the correctness of the proof rules. ■

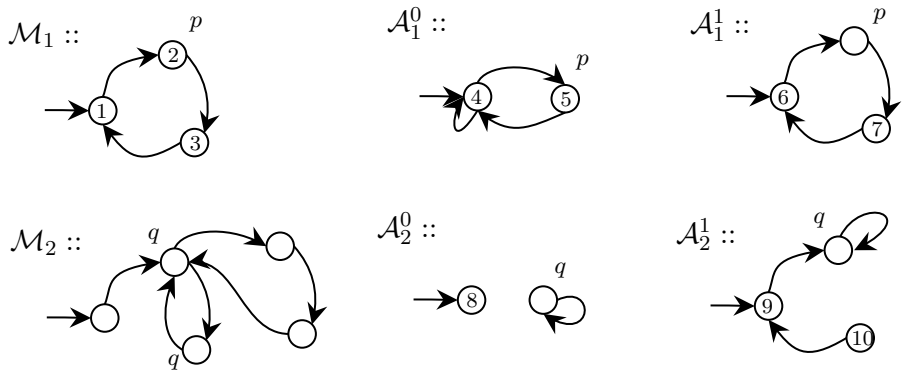

Figure 4.2: Example: Model-Checking for $\langle \mathcal{M}_1, \mathcal{M}_2 \rangle \models \mathbb{P}$

Example.

Consider $\mathcal{M}_1, \mathcal{M}_2$ (Figure 4.2) and $\mathbb{P} = \forall^1\pi\exists^2\tau.\ \mathsf{G}(\mathsf{p}_\pi \oplus \mathsf{X}\mathsf{p}_\pi \oplus \mathsf{X}\mathsf{X}\mathsf{p}_\pi) \wedge \mathsf{G}(\mathsf{p}_\pi \rightarrow \mathsf{q}_\tau)$, where $\oplus$ denotes XOR. For brevity, we ignore $\mathbb{B}$ since $\langle \mathcal{M}_1, \mathcal{M}_2 \rangle \models \mathbb{P}$. When running Algorithm 4.1 for $\langle \mathcal{M}_1, \mathcal{M}_2 \rangle \models \mathbb{P}$, we first construct $\mathcal{A}_1^0, \mathcal{A}_2^0$ as over- and under-approximations of $\mathcal{M}_1, \mathcal{M}_2$, respectively (Figure 4.2). Then, we check whether $\langle \mathcal{A}_1^0, \mathcal{A}_2^0 \rangle \models \mathbb{P}$. This does not hold, and MMC returns counterexamples $\langle \emptyset p \emptyset^\omega, \emptyset q^\omega \rangle$. We refine the abstractions according to these counterexamples.

Next, we find the maximal induced run of $\emptyset p \emptyset^\omega$ in $\mathcal{M}_1$, which is the path $\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{1}\perp$. Since the path for $\emptyset p \emptyset^\omega$ is $\mathbf{4}, \mathbf{5}, \mathbf{4}, \mathbf{4}^\omega$ in $\mathcal{A}_1^0$, we need to refine the state $\mathbf{4}$ in $\mathcal{A}_1^0$. By similar analysis of $\emptyset q^\omega$, state $\mathbf{8}$ is to be split in $\mathcal{A}_2^0$. Thus, we split state $\mathbf{4}$ from $\mathcal{A}_1^0$ to states $\mathbf{6}$,$\mathbf{7}$ in $\mathcal{A}_1^1$. In $\mathcal{A}_2^0$, we split state $\mathbf{8}$ to states $\mathbf{9}$,$\mathbf{10}$ in $\mathcal{A}_2^1$. Then, model-checking $\langle \mathcal{A}_1^1, \mathcal{A}_2^1 \rangle \models \mathbb{P}$, passes, and we return $\langle \mathcal{M}_1, \mathcal{M}_2 \rangle \models \mathbb{P}$.

4.4 Counterexample Guided Model-Checking Using (PR)

Algorithm 4.1 is guided by the difference between the abstract models and the original models. We now consider the $\forall^*\exists^*$ fragment of MultiLTL. By Lemma 3.3.5, when model-checking $\forall^*\exists^*$MultiLTL fails, we can get counterexamples for the models under $\forall$. We use these counterexamples to further improve our model-checking scheme for this fragment.

We are given a $\forall^*\exists^*$MultiLTL formula $\mathbb{P} = \forall_1^1\pi_1 \ldots \forall_k^k\pi_k\exists_{k+1}^{k+1} \ldots \exists_n^n\pi_n.\varphi$ and a multi-model $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_n \rangle$ over AP as input. Our model-checking procedure is described in Algorithm 4.2.

The procedure MMC$(\mathbb{M}, \mathbb{P})$ performs multi-property model-checking, and returns $(\mathbf{true}, \emptyset)$ if $\mathbb{M} \models \mathbb{P}$, and otherwise returns $(\mathbf{false}, cex)$, where cex is a counterexample vector $\langle w_1, \ldots, w_k \rangle$ such that $w_i \in \mathcal{L}(\mathcal{M}_i)$ for every $i \in [1, k]$ and there are no $w_i \in \mathcal{L}(\mathcal{M}_i)$ for $i \in [k+1, n]$ such that $\langle w_1, \ldots, w_n \rangle \models \mathbb{P}$, as per Lemma 3.3.5 (2). We fix every $\mathcal{A}_i$ under $\exists$ to be $\mathcal{M}_i$. Thus, it is guaranteed that the model-checking failure is not caused by words that are missing from the under-approximations, yet do exist in the concrete models. A counterexample w_i from $\langle w_1, \ldots, w_k \rangle$ is *spurious* if $w_i \in \mathcal{L}(\mathcal{A}_i)$ yet $w_i \notin \mathcal{L}(\mathcal{M}_i)$. That is, w_i cannot serve as proof that $\mathbb{M} \not\models \mathbb{P}$. REFINE refines every approximation $\mathcal{A}_i$ for which there is a tuple (i, w_i) in *spuriousList*, the list of spurious counterexamples, by removing w_i from $\mathcal{A}_i$.

Algorithm 4.2 CEGAR-based $\forall^*\exists^*$MultiLTL model-checking

Input: $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_n \rangle$, $\mathbb{P} = \forall_1^1 \pi_1 \ldots \forall_k^k \pi_k \exists_{k+1}^{k+1} \ldots \exists_n^n \pi_n . \varphi(\pi_1, \ldots, \pi_n)$.
Output: $\mathbb{M} \models \mathbb{P}$?

```
1:  𝔸 = INITIALIZE∀*∃*(𝕄, ℙ)
2:  while true do
3:     (res, cex) = MMC(𝔸, ℙ)
4:     if res == true then
5:        return 𝕄 ⊨ ℙ
6:     end if
7:     spuriousList = SPURIOUS(cex, 𝕄)
8:     if ISEMPTY(spuriousList) then
9:        return 𝕄 ⊭ ℙ
10:    end if
11:    𝔸 = REFINE(cex, spuriousList, 𝔸, 𝕄)
12: end while
```

Initialization.

In INITIALIZE$\forall^*\exists^*$ (Line 1), for every model $\mathcal{M}_i$ such that $\mathbb{Q}_i^i = \forall$, we initialize an abstract model $\mathcal{A}_i$ as described in Section 4.1. For every model $\mathcal{M}_i$ such that $\mathbb{Q}_i^i = \exists$, we fix $\mathcal{A}_i$ to be $\mathcal{M}_i$. Thus, $\mathcal{M}_i \leq \mathcal{A}_i$ for every $i \in [1, k]$ and $\mathcal{A}_i \leq \mathcal{M}_i$ (trivially) for every $i \in [k+1, n]$.

Model-Checking.

When we reach line 3, it is guaranteed that $\mathcal{M}_i \leq \mathcal{A}_i$ for every $i \in I_\forall$ (and $\mathcal{A}_i \leq \mathcal{M}_i$ for every $i \in I_\exists$, since $\mathcal{A}_i = \mathcal{M}_i$). Thus, we try to apply the proof rule (PR), and model-check $\langle \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle \models \mathbb{P}$ (Line 3) by running MMC. If the result is **true**, then by (PR), we have $\mathbb{M} \models \mathbb{P}$ (Line 5). Otherwise, we get a counterexample vector of the form $\langle w_1, \ldots, w_k \rangle$.

Counterexample Analysis.

(Lines 6-9). The procedure SPURIOUS iterates over the words in the counterexample $\langle w_1, \ldots, w_k \rangle$, and returns a list of tuples (i, w_i) such that $w_i \notin \mathcal{L}(\mathcal{M}_i)$. Note that since $\langle w_1, \ldots, w_k \rangle$ is a counterexample, it holds that $w_i \in \mathcal{L}(\mathcal{A}_i)$ for every $i \in [1, k]$. Thus, every w_i in the list of (i, w_i) is spurious. If there are no spurious counterexamples, then we return $\mathbb{M} \not\models \mathbb{P}$ (Line 8). Otherwise, we refine the approximations based on the spurious counterexamples.

In the worst case, the run iterates until $\mathcal{M}_i = \mathcal{A}_i$ for every $i \in [1, n]$, in which case there are no spurious counterexamples. Hence, we have the following.

Lemma 4.4.1. *Algorithm 4.2 terminates with the correct result.*

Proof. In each iteration, either we terminated with $\mathbb{M} \models \mathbb{P}$ or find a counterexample. In the case of counterexample, one of the over-approximations is refined. Since we cannot refine an abstract model $\mathcal{A}_i$ more than $|\mathcal{M}_i|$ times for $i \in [1, k]$, after at most $\sum_{i=1}^{k} |\mathcal{M}_i|$ refinements, $\mathcal{A}_i = \mathcal{M}_i$ for $i \in [1, k]$. Then, the output is as performing model-checking on the original models. Thus, the answer at this either be $\mathbb{M} \models \mathbb{P}$ or a non-spurious counterexample.

Correctness follows from the correctness of the model-checking procedure and the correctness of the proof rule. ■

Algorithm 4.2 improves Algorithm 4.1 in several ways. First, in order to compute the counterexamples there is no need to complement the models, to achieve a counterexample, which comes with an exponential price. Second, the counterexamples are provided by the model-checking process. As such, they are of "higher quality", in the sense that they take into account the checked property and are guaranteed to remove refuting parts from the abstractions. This, in turn, leads to faster convergence.

Chapter 5

Multi-Properties for Finite Traces

We now consider models whose traces are finite. This setting is natural, for example, when modeling terminating programs. In this case, a model is a finite-word language, and hyperproperties can be expressed by nondeterminisitic finite hyperautomata (NFH) [16]. To explain the idea behind NFH, we first review nondeterministic automata.

Definition 5.0.1. A *nondeterministic finite-word automaton* (NFA) is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where Σ is an alphabet, Q is a nonempty finite set of *states*, $Q_0 \subseteq Q$ is a set of *initial states*, $F \subseteq Q$ is a set of *accepting states*, and $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*.

Given a word $w = \sigma_1\sigma_2 \cdots \sigma_n$ over Σ, a *run of A on w* is a sequence of states $q_0, q_1, \ldots, q_n$, such that $q_0 \in Q_0$, and for every $i \in [1, n]$, it holds that $(q_{i-1}, \sigma_i, q_i) \in \delta$. The run is *accepting* if $q_n \in F$. The *language* of A, denoted $\mathcal{L}(A)$, is the set of all words on which A has an accepting run. A language $\mathcal{L}$ is called *regular* if there exists an NFA such that $\mathcal{L}(A) = \mathcal{L}$.

An NFA A is called *deterministic* (DFA) if $|Q_0| = 1$, and for every $q \in Q$ and $\sigma \in \Sigma$, there exists exactly one q' for which $(q, \sigma, q') \in \delta$. It is well-known that every NFA has an equivalent DFA.

We now turn to explain NFH. An NFH $\mathcal{P}$ consists of a set of *word variables*, an NFA $\mathsf{nfa}(\mathcal{P})$ that runs on words that are assigned to these variables (which is akin to the unquantified LTL formula in a HyperLTL formula), and a *quantification condition* that describes the requirements for these assignments (which is akin to the quantifiers in a HyperLTL formula). Thus, NFH can be thought of as the regular-language counterpart of HyperLTL.

Definition 5.0.2. A *nondeterministic finite-word hyperautomaton* (NFH) $\mathcal{P}$ is a 7-tuple $(\Sigma, X, Q, Q_0, F, \delta, \alpha)$ where $X = \{x_1, ..., x_k\}$ is a finite set of word variables, $\alpha = \mathbb{Q}_1 x_1 \ldots \mathbb{Q}_k x_k$ is a quantification condition s.t. $\mathbb{Q}_i \in \{\forall, \exists\}$ for every $i \in [1, k]$ and $\mathsf{nfa}(\mathcal{P}) = (\hat{\Sigma}, Q, Q_0, \delta, F)$ is an NFA, where $\hat{\Sigma} = (\Sigma \cup \{\#\})^X$. This means that every letter in $\hat{\Sigma}$ is a mapping from every variable in X to a letter in Σ.

We call $\mathsf{nfa}(\mathcal{P})$ the *NFA induced by* $\mathcal{P}$. Intuitively, $\mathsf{nfa}(\mathcal{P})$ is an NFA that recognizes all k-tuples of finite words that satisfy $\mathcal{P}$.

We now define the notion of acceptance by an NFH.

Definition 5.0.3. Let $\langle w_1, \ldots, w_k \rangle$ be a tuple of words, $m = \max_i\{|w_i|\}$ and $\mathcal{P}$ be an NFH. A *run of* $\mathsf{nfa}(\mathcal{P})$ *over* $\langle w_1, \ldots, w_k \rangle$ is a run over a word $\bar{w} \in \left((\Sigma \uplus \{\#\})^k\right)^*$ of length m, in which for each $i \in [1, m]$, $\bar{w}_i = \{(w_1)_{ix_1}, \ldots, (w_k)_{ix_k}\}$, where $(w_i)_j$ is the j^{th} letter in the word w_i if $|w_i| \geq j$ and # otherwise. That is, the run of $\mathsf{nfa}(\mathcal{P})$ that matches assigning the letters of w_i to x_i, and uses # as a padding at the end of w_i when $|w_i| \leq m$.
We denote $\langle w_1, \ldots, w_k \rangle \in \mathcal{L}(\mathsf{nfa}(\mathcal{P}))$ if the run of $\mathsf{nfa}(\mathcal{P})$ over $\langle w_1, \ldots, w_k \rangle$ is accepting.

Example 5.0.4. Let $\mathcal{P}$ be an NFH with $X = \{x_1, x_2, x_3\}$ and let $w_1 = aa$, $w_2 = b$ and $w_3 = aaa$ be words over $\Sigma = \{a, b\}$. A run of $\mathsf{nfa}(\mathcal{P})$ over $\langle w_1, w_2, w_3 \rangle$ is a run of $\mathsf{nfa}(\mathcal{P})$ over the word $\bar{w} = \{a_{x_1}, b_{x_2}, a_{x_3}\}\{a_{x_1}, \#_{x_2}, a_{x_3}\}\{\#_{x_1}, \#_{x_2}, a_{x_3}\}$.

Let $S \subseteq \Sigma^*$ be a language and let $v : X \to S$ be an assignment of the word variables of $\mathcal{P}$ with words in S. We denote by $v[x \to w]$ the assignment obtained from v by assigning the word $w \in S$ to $x \in X$. We represent v by the word $\langle v \rangle = \langle v(x_1), \ldots v(x_k) \rangle$. We now define the acceptance condition of a language S by an NFH $\mathcal{P}$. We first define the satisfaction relation $\models$ for $S, \mathcal{P}$, a quantification condition α, and an assignment $v : X \to S$, as follows:

- For $\alpha = \epsilon$, we write $S \models_v (\alpha, \mathcal{P})$ if $\langle v \rangle \in \mathcal{L}(\mathsf{nfa}(\mathcal{P}))$.
- For $\alpha = \exists x_i \alpha'$, we write $S \models_v (\alpha, \mathcal{P})$ if there exists $w \in S$, such that $S \models_{v[x_i \to w]} (\alpha', \mathcal{P})$.
- For $\alpha = \forall x_i \alpha'$, we write $S \models_v (\alpha, \mathcal{P})$ if for every $w \in S$, it holds that $S \models_{v[x_i \to w]} (\alpha', \mathcal{P})$.

Since the quantification condition of $\mathcal{P}$ includes all $x \in X$, the satisfaction is independent of the assignment v, and we denote $S \models \mathcal{P}$, in which case, we say that $\mathcal{P}$ accepts S.

Definition 5.0.5. Let $\mathcal{P}$ be an NFH, the *hyperlanguage of* $\mathcal{P}$, denoted $\mathcal{L}(\mathcal{P})$, is the set of all languages accepted by $\mathcal{P}$.

We demonstrate NFH and their acceptance conditions with an example.

Example 5.0.6. Consider the NFH $\mathcal{A}$ in Figure 5.1 (left) over the alphabet $\Sigma = \{a, b\}$ and two word variables x and y. The NFA part $\mathsf{nfa}(\mathcal{A})$ of $\mathcal{A}$ reads two words simultaneously: one is assigned to x and the other to y. Accordingly, the letters that $\mathsf{nfa}(\mathcal{A})$ reads are tuples of the form $\{\sigma_x, \sigma'_y\}$, where σ is the current letter in the word that is assigned to x, and similarly for σ' and y. The symbol # is used for padding at the end if one of the words is shorter than the other. In the example, for two words w_1, w_2 that are assigned to x and y, respectively, $\mathsf{nfa}(\mathcal{A})$ requires that (1) w_1, w_2 agree on their a positions, and (2) once one of the words has ended, the other must only contain b letters. Since the quantification condition of $\mathcal{A}$ is $\forall x \forall y$, in a language S that $\mathcal{A}$ accepts, every two words agree on their a positions. As a result, all the words in S must agree on their a positions. The *hyperlanguage* of $\mathcal{A}$ is then the set of all finite-word languages in which all words agree on their a positions.

Notation. Let $w_1, \ldots, w_k$ be k words in $(\Sigma \cup \{\#\})^*$ and $\mathcal{P}$ an NFH with k quantifiers in α. We denote $\langle w_1, \ldots, w_k \rangle \models \mathcal{P}$ if $\emptyset \models_v (\epsilon, \mathcal{P})$, where $v(x_i) = w_i$ for every $i \in [1, k]$.
Note that since the quantification condition is ϵ, we can substitute $\emptyset$ with every language over $(\Sigma \cup \{\#\})$.

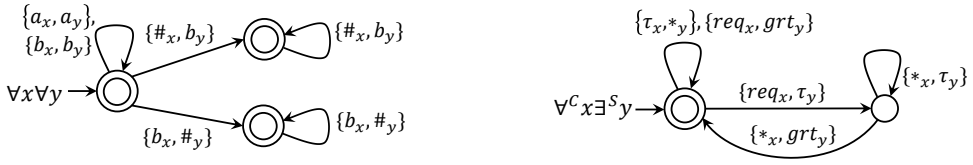

Figure 5.1: The NFH $\mathcal{A}$ (left) and the MNFH $\mathcal{B}$ (right).

Definition 5.0.7. The *model-checking problem for NFH* is to decide, given a language S and an NFH $\mathcal{P}$, whether $\mathcal{P}$ accepts S, in which case we denote $S \models \mathcal{P}$.

When S is given as an NFA, the model-checking problem is decidable (albeit, as for HyperLTL, by a nonelementary algorithm) [16].

5.1 Multi-Languages and Multi-NFH

As in the case of models with infinite traces, we generalize languages and NFH to *multi-languages* and *multi-NFH* (MNFH). Thus, a multi-language is a tuple $\mathbb{S} = \langle S_1, S_2, \dots S_k \rangle$ of finite-word languages, and an MNFH $\mathcal{A}$ is an NFH with indexed quantifiers.

Definition 5.1.1. A multi-NFH (MNFH) is a tuple $(\Sigma, X, Q, Q_0, F, \delta, \alpha)$ where Σ, X, Q, Q_0, F and δ are as in Definition 5.0.2, and $\alpha = \mathbb{Q}_1^{i_1} x_1 \dots \mathbb{Q}_k^{i_k} x_k$ is a quantification condition, where $\mathbb{Q}_j^{i_j} \in \{\forall, \exists\}$ and $i_j \in [1, k]$ for every $j \in [1, k]$.

The *NFA induced by* $\mathbb{P}$ is denoted $\mathsf{nfa}(\mathbb{P})$ and defined similarly to Definition 5.0.2.

Intuitively, the semantics are similar to that of Chapter 3, i.e., a quantifier $\mathbb{Q}^i$ in the quantification condition of $\mathcal{A}$ refers to S_i (rather than all quantifiers referring to the same language in the case of standard NFH).

Let $\mathbb{S} = \langle S_1, \dots, S_k \rangle$ be a multi-language where $S_i \subseteq \Sigma^*$ for every $i \in [1, k]$, and $v : X \to S$ be an assignment of the word variables of $\mathbb{P}$ to words in $S = \uplus_{i=0}^{k} S_i$. We now define the acceptance condition of a multi-language $\mathbb{S}$ by an MNFH $\mathbb{P}$. We first define the satisfaction relation $\models$ for $\mathbb{S}, \mathbb{P}$, with a quantification condition α, and an assignment $v : X \to \uplus_{i=0}^{k} S_i$, as follows:

- For $\alpha = \epsilon$, we denote $\mathbb{S} \models_v (\alpha, \mathbb{P})$ if $\langle v \rangle \in \mathcal{L}(\mathsf{nfa}(\mathbb{P}))$.
- For $\alpha = \exists^i x_j \alpha'$, we denote $\mathbb{S} \models_v (\alpha, \mathbb{P})$ if there exists $w \in S_i$, such that $\mathbb{S} \models_{v[x_j \to w]} (\alpha', \mathbb{P})$.
- For $\alpha = \forall^i x_j \alpha'$, we denote $\mathbb{S} \models_v (\alpha, \mathbb{P})$ if for every $w \in S_i$, it holds that $\mathbb{S} \models_{v[x_j \to w]} (\alpha', \mathbb{P})$.

As before, the satisfaction is independent of the assignment v, and we denote $\mathbb{S} \models \mathbb{P}$, in which case, we say that $\mathbb{P}$ accepts $\mathbb{S}$.

Definition 5.1.2. Let $\mathbb{P}$ be an MNFH. The *multi-language of* $\mathbb{P}$, denoted $\mathcal{L}(\mathbb{P})$, is the set of all multi-languages accepted by $\mathbb{P}$.

We consider multi-languages that consist of regular languages. We can express such a multi-language $\langle L_1, L_2, \dots, L_k \rangle$ by a tuple $\mathbb{M} = \langle \mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k \rangle$ of NFAs, where $\mathcal{L}(\mathcal{M}_i) = L_i$ for every $i \in [1, k]$. We call $\mathbb{M}$ a *multi-NFA* (MNFA).

Definition 5.1.3. The model-checking problem for MNFH is to decide, given a multi-language $\mathbb{S}$ and an MNFH $\mathbb{P}$, whether $\mathbb{P}$ accepts $\mathbb{S}$. In which case, we denote $\mathbb{S} \models \mathbb{P}$.

We define the model-checking problem for MNFA accordingly, and denote $\mathbb{M} \models \mathbb{P}$ if an MNFH $\mathbb{P}$ accepts $\mathcal{L}(\mathbb{M})$.

Example 5.1.4. Consider an MNFA $\langle S, C\rangle$, where S models a server and C models a client, and the MNFH $\mathcal{B}$ of Figure 5.1 (right) over $\Sigma = \{req, grt, \tau\}$, where req is a request sent to the server, grt is a grant given to the client and τ is a non-communicating action.

The multi-model $\langle S, C\rangle$ satisfies $\mathcal{B}$ iff for every run of S there exists a run of C such that every request by C is eventually granted by S. This means that the server does not starve the client.

From now on, we assume without loss of generality, in a similar manner to Note 3.2.3, that the quantification conditions of the MNFH that we consider are of the form $\mathbb{Q}_1^1 x_1 \mathbb{Q}_2^2 x_2 \dots \mathbb{Q}_k^k x_k$.

5.2 Equivalence of MNFH Model-Checking and NFH Model-Checking

We now show that the model-checking problem for MNFH is equivalent to the model-checking problem for NFH.

For the first direction, in a similar manner to Theorem 3.1, a language S is accepted by an NFH $\mathcal{P}$ iff $\langle S\rangle$ is accepted by the MNFH $\mathbb{P}$ obtained from $\mathcal{P}$ by indexing all quantifiers in the quantification condition of $\mathcal{P}$ by the same index 1, as shown by the following lemma.

Theorem 5.1. *The model-checking problem for NFH is reducible to the model-checking problem for MNFH.*

Proof. Let $\mathcal{M} = (\Sigma, Q, Q_0, \delta, F)$ be an NFA and $\mathcal{P} = (\Sigma, X, Q, Q_0, F, \delta, \alpha)$ be an NFH. Consider the multi-NFA $\mathbb{M} = \langle \mathcal{M}\rangle$ and the MNFH $\mathbb{P} = (\Sigma, X, Q, Q_0, F, \delta, \beta)$, where β is the same as α, but with every quantifier $\mathbb{Q}_i$ replaced by $\mathbb{Q}_i^1$. Denote $S = \mathcal{L}(\mathcal{M})$, and $\mathbb{S} = \langle \mathcal{L}(\mathcal{M})\rangle$ and let $v : X \to \Sigma^*$ an assignment. By induction on the number of quantifiers in α, we show that $S \models_v (\alpha, \mathcal{P})$ iff $\mathbb{S} \models_v (\beta, \mathbb{P})$.

Base: When $\alpha = \epsilon$, also $\beta = \epsilon$. Thus, $S \models_v (\alpha, \mathcal{P})$ iff $\langle v\rangle \in \mathcal{L}(\mathsf{nfa}(\mathcal{P}))$. By the definition of $\mathbb{P}$, it holds that $\mathsf{nfa}(\mathcal{P}) = \mathsf{nfa}(\mathbb{P})$. Thus $\langle v\rangle \in \mathcal{L}(\mathsf{nfa}(\mathcal{P}))$ iff $\langle v\rangle \in \mathcal{L}(\mathsf{nfa}(\mathbb{P}))$. This holds iff $\mathbb{S} \models_v (\beta, \mathbb{P})$.

Step: Assume that $\alpha = \mathbb{Q}x.\alpha'$. By the construction, $\beta = \mathbb{Q}^1 x.\beta'$. Consider the different options for $\mathbb{Q}$:

- $\mathbb{Q} = \exists$, then $S \models_v (\alpha, \mathcal{P})$ iff there exists $w \in S$ such that $S \models_{v[x\to w]} (\alpha', \mathcal{P})$. By the induction hypothesis this holds iff there exists $w \in S$ such that $S \models_{v[x\to w]} (\beta', \mathbb{P})$. By the definition of $\exists^1$ and $\mathbb{S}$, this holds iff $\mathbb{S} \models_v (\beta, \mathbb{P})$.
- $\mathbb{Q} = \forall$, then $S \models_v (\alpha, \mathcal{P})$ iff for every $w \in S$ it holds that $S \models_{v[x\to w]} (\alpha', \mathcal{P})$. By the induction hypothesis this holds iff for every $w \in S$ it holds that $S \models_{v[x\to w]} (\beta', \mathbb{P})$. By the definition of $\forall^1$ and $\mathbb{S}$, this holds iff $\mathbb{S} \models_v (\beta, \mathbb{P})$.

Thus, we can conclude that $\mathcal{M} \models \mathcal{P}$ iff $\mathbb{M} \models \mathbb{P}$. ■

We now proceed to the second direction. The idea of the proof is similar to the reduction from MultiLTL model-checking to HyperLTL model-checking (Theorem 3.2). We aim to construct an NFA that contains all the NFAs in the multi-NFA, with their corresponding index. Then, we change the MNFH to an NFH, such that it refers also to the indices and the semantics of the acceptance condition.

Definition 5.2.1. Let $\mathcal{M} = (\Sigma, Q, Q_0, \delta, F)$ be an NFA. The *indexed NFA of* $\mathcal{M}$ is $\mathsf{ind}_i(\mathcal{M}) = (\Sigma_i, Q, Q_0, \delta_i, F)$, where $\Sigma_i = \Sigma \times \{\mathbf{i}\}$ and $(q, (\sigma, \mathbf{i}), q') \in \delta_i$ iff $(q, \sigma, q') \in \delta$ for every $q, q' \in Q$ and $\sigma \in \Sigma$.

Let $w \in \Sigma^*$ be a word such that $w = \sigma_0, \dots \sigma_k$. The *indexed word of* w, denoted $\mathsf{ind}_i(w)$, is $(\sigma_0, \mathbf{i}) \dots (\sigma_k, \mathbf{i})$.

Corollary 5.2. *By Definition 5.2.1, it holds that* $w \in \mathcal{L}(\mathcal{M})$ *iff* $\mathsf{ind}_i(w) \in \mathcal{L}(\mathsf{ind}_i(\mathcal{M}))$.

Definition 5.2.2. Given a tuple of words $\langle w_1, \dots, w_k \rangle$, the *indexed tuple* of $\langle w_1, \dots, w_k \rangle$ is $\mathsf{ind}(\langle w_1, \dots, w_k \rangle) = \langle \mathsf{ind}_1(w_1), \dots, \mathsf{ind}_k(w_k) \rangle$.

The following theorem describes the reduction from MNFH model-checking to NFH model-checking.

Theorem 5.3. *Let* $\mathbb{P}$ *be an MNFH, and let* $\mathbb{M} = \langle \mathcal{M}_1, \dots \mathcal{M}_k \rangle$ *be an MNFA. Then there exist an NFA* $\mathcal{M}$ *and an NFH* $\mathcal{P}$ *such that* $\mathbb{M} \models \mathbb{P}$ *iff* $\mathcal{M} \models \mathcal{P}$.

Proof. Let $\mathbb{M} = \langle \mathcal{M}_1, \dots, \mathcal{M}_k \rangle$ be a multi-NFA, where $\mathsf{ind}_i(\mathcal{M}_i) = (\Sigma_i, Q_i, Q_0^i, \delta_i, F_i)$, and $\mathbb{P} = (\Sigma', X, Q, Q_0, F, \delta, \alpha)$ be an MNFH with $\alpha = \mathbb{Q}_1^1 x_1 \dots \mathbb{Q}_k^k x_k$. Denote $\Sigma = \uplus_{i=1}^k \Sigma_i$.

Define $\mathcal{M} = (\Sigma, \uplus_{i=1}^k Q_i, \uplus_{i=1}^k Q_0^i, \uplus_{i=1}^k \delta_i, \uplus_{i=1}^k F_i)$. By Corollary 5.2, $w \in \mathcal{L}(\mathcal{M}_i)$ iff $\mathsf{ind}_i(w) \in \mathcal{L}(\mathcal{M})$. Therefore, we can use the index of the letters to determine which NFA they originated from.

Define $\mathcal{P} = (\Sigma, X, Q \uplus \{q_*\}, Q_0, F \uplus \{q_*\}, \delta_*, \beta)$, where β is as α without the indices on the quantifiers and q_* is a fresh accepting state. The transition relation δ_* is defined as follows:

1. For every $q, q' \in Q$ and every letter set $\hat{\sigma} = \{\sigma_{1_{x_1}}, \dots, \sigma_{k_{x_k}}\} \in \hat{\Sigma}$:
   Let $\mathsf{ind}(\hat{\sigma}) = \{(\sigma_1, \mathbf{1})_{x_1}, \dots, (\sigma_k, \mathbf{k})_{x_k}\}$. We define $(q, \mathsf{ind}(\hat{\sigma}), q') \in \delta_*$ iff $(q, \hat{\sigma}, q') \in \delta$.

2. For every $q \in Q_0$ and every letter set $\hat{\sigma} = \{\sigma_{1_{x_1}}, \dots, \sigma_{k_{x_k}}\} \in \hat{\Sigma}$, we define
   $(q, \{(\sigma_1, i_1)_{x_1}, \dots, (\sigma_k, i_k)_{x_k}\}, q_*) \in \delta_*$ iff $i_j \neq j$ for some $j \in [1, k]$.

3. $(q_*, \sigma^*, q_*) \in \delta_*$, for every letter-set $\sigma^* = \{\sigma^*_{1_{x_1}}, \dots, \sigma^*_{k_{x_k}}\} \in \hat{\Sigma}$.

Intuitively, (1) means that for every k-word $\langle w_1, \dots, w_k \rangle$ accepted by $\mathsf{nfa}(\mathbb{P})$, its indexed k-word $\mathsf{ind}(\langle w_1, \dots, w_k \rangle)$ is accepted by $\mathsf{nfa}(\mathcal{P})$. (2)+(3) mean that when x_i is assigned to a word not in $\mathcal{M}_i$, the NFA $\mathsf{nfa}(\mathcal{P})$ transitions into the accepting sink q_*.

Denote $\mathbb{S} = \langle \mathcal{L}(\mathcal{M}_1), \dots, \mathcal{L}(\mathcal{M}_k) \rangle$ and $S = \mathcal{L}(\mathcal{M})$. By induction over the number of quantifiers in α, we prove that for every assignment v that respects $\mathbb{S}$, and $\mathsf{ind}(v)$ that respects $\mathcal{M}$, it holds that: $\mathbb{S} \models_v (\alpha, \mathbb{P})$ iff $S \models_{\mathsf{ind}(v)} (\beta, \mathcal{P})$.

Base: By the construction, we get that for every $\langle w_1, \ldots, w_k \rangle$ such that $w_i \in \mathcal{L}(\mathcal{M}_i)$, it holds that $\langle w_1, \ldots, w_k \rangle \in \mathcal{L}(\mathsf{nfa}(\mathbb{P}))$ iff $\mathsf{ind}(\langle w_1, \ldots, w_k \rangle) \in \mathcal{L}(\mathsf{nfa}(\mathcal{P}))$.
Step: Let $\alpha = \mathbb{Q}_i^i x_i.\alpha'$, therefore, $\beta = \mathbb{Q}_i x_i.\alpha$.

- If $\mathbb{Q}_i^i = \exists^i$, then $\mathbb{S} \models_v (\alpha, \mathbb{P})$ iff there exists $w \in \mathcal{L}(\mathcal{M}_i)$ such that $\mathbb{S} \models_{v[x_i \to w]} (\alpha', \mathbb{P})$. By the induction hypothesis this holds iff that there exists $w \in \mathcal{L}(\mathcal{M}_i)$, such that $S \models_{\mathsf{ind}(v)[x_i \to \mathsf{ind}_i(w)]} (\beta', \mathbb{P})$. By the construction of $\mathcal{P}$, this means that there exists $w' \in \mathcal{L}(\mathcal{M})$, such that $S \models_{\mathsf{ind}(v)[x_i \to w']} (\beta', \mathbb{P})$, which by the semantics of MNFH holds iff $S \models_{\mathsf{ind}(v)} (\beta, \mathcal{P})$.
- If $\mathbb{Q}_i^i = \forall^i$, then $\mathbb{S} \models_v (\alpha, \mathbb{P})$ iff for every $w \in \mathcal{L}(\mathcal{M}_i)$ it holds that $\mathbb{S} \models_{v[x_i \to w]} (\alpha', \mathbb{P})$. By the induction hypothesis this holds iff for every $w \in \mathcal{L}(\mathcal{M}_i)$ it holds that $S \models_{\mathsf{ind}(v)[x_i \to \mathsf{ind}_i(w)]} (\beta', \mathbb{P})$. By the construction of $\mathcal{P}$, it also holds that $S \models_{\mathsf{ind}(v)[x_i \to \mathsf{ind}_j(w)]} (\beta', \mathbb{P})$ for every $w \in (\Sigma' \cup \{\#\})^*$ and $j \neq i$. Therefore, for every $w \in \mathcal{L}(\mathcal{M}_i)$ it holds that $S \models_{\mathsf{ind}(v)[x_i \to \mathsf{ind}_i(w)]} (\beta', \mathbb{P})$ iff for every $w \in \mathcal{L}(\mathcal{M})$ it holds that $S \models_{\mathsf{ind}(v)[x_i \to w]} (\beta', \mathbb{P})$, which by the semantics of MNFH holds iff $S \models_{\mathsf{ind}(v)} (\beta, \mathcal{P})$.

Therefore, $\mathbb{M} \models \mathbb{P}$ iff $\mathcal{M} \models \mathcal{P}$ as required. ■

The construction in the proof of Theorem 5.3 uses an alphabet whose size is polynomial in the original alphabet. The model $\mathcal{M}$ that we construct is linear in the size of $\mathbb{M}$, and the state space of $\mathcal{P}$ is linear in that of $\mathbb{P}$. However, since the size of the alphabet is larger, and the letters of $\mathcal{P}$ are set-letters, there may be exponentially many transitions in $\mathcal{P}$ compared with $\mathbb{P}$.

5.3 Direct Algorithm for MNFH Model-Checking

In this section, we describe a direct algorithm for model-checking MNFH, which is based on the algorithm for model-checking NFH in [16]. Additionally, when $\mathbb{M} \not\models \mathbb{P}$, it is possible to extract a counterexample $\langle w_1, \ldots, w_k \rangle$ when $\mathbb{Q}_i = \forall$ for $i \in [1, k]$, in a similar manner to Lemma 3.3.5.

Lemma 5.3.1. *There is a direct algorithm for model-checking MNFH.*

Proof. Let $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle$ be a multi-NFA, where $\mathcal{M}_i = (\Sigma, P_i, P_i^0, \rho_i, F_i)$ for every i, and let $\mathbb{P} = (\Sigma, \{x_1, \ldots, x_k\}, Q, Q_0, F, \delta, \mathbb{Q}_1^1 \ldots \mathbb{Q}_k^k)$ be an MNFH. Note that we assume, without loss of generality, that every NFA $\mathcal{M}_i$ is quantified exactly once, similarly to the MultiLTL case.

We first extend the alphabet of each $\mathcal{M}_i$ to $\Sigma \cup \{\#\}$, and extend its language to $\mathcal{L}(\mathcal{M}_i) \cdot \{\#\}^*$. This can be done by adding a new accepting state $q_\#$ and transitions labeled $\#$ from every accepting state $(F \cup \{q_\#\})$ to it. We describe a procedure for deciding whether $\mathbb{M} \models \mathbb{P}$.

For the case that $k = 1$, if $\alpha = \exists^1 x_1$, then $\mathbb{M} \models \mathbb{P}$ iff $\mathcal{L}(\mathcal{M}_1) \in \mathcal{L}(\mathbb{P})$ iff $\mathcal{L}(\mathcal{M}_1) \cap \mathcal{L}(\mathsf{nfa}(\mathbb{P})) \neq \emptyset$. Otherwise, if $\alpha = \forall^1 x_1$, then $\mathbb{M} \models \mathbb{P}$ iff $\mathcal{L}(\mathcal{M}_1) \in \mathcal{L}(\mathbb{P})$ iff $\mathcal{L}(\mathcal{M}_1) \notin \mathcal{L}(\overline{\mathbb{P}})$, where $\overline{\mathbb{P}}$ is the NFH for $\overline{\mathcal{L}(\mathbb{P})}$ (the complementation construction is found in [16]). The quantification condition for $\overline{\mathbb{P}}$ is $\exists x_1$, conforming to the base case.

For $k > 1$, we construct a sequence of NFA $\mathcal{A}_k, \mathcal{A}_{k-1} \ldots, \mathcal{A}_1$ as follows. Initially, $\mathcal{A}_k = \mathsf{nfa}(\mathbb{P})$. Let $\mathcal{A}_i = (\Sigma_i, Q_i, Q_i^0, \delta_i, \mathcal{F}_i)$.

If $\mathbb{Q}_i = \exists^j$, then we construct $\mathcal{A}_{i-1}$ as follows. The set of states of $\mathcal{A}_{i-1}$ is $Q_i \times P_i$, and the set of initial states is $Q_i^0 \times P_i^0$. The set of accepting states is $\mathcal{F}_i \times F_i$. For every $(q \xrightarrow{f} q') \in \delta_i$

and every $(p \xrightarrow{f(x_i)} p') \in \rho_i$, we have $((q,p) \xrightarrow{f \setminus \{\sigma_{i_{x_i}}\}} (q',p')) \in \delta_{i-1}$. We denote this construction by $\mathcal{M}_i \cap_{x_i} \mathcal{A}_i$. Then, $\mathcal{A}_{i-1}$ accepts a word assignment v iff there exists a word $u \in \mathcal{L}(M_i)$, such that $\mathsf{nfa}(\mathcal{A}_i)$ accepts $\langle v \cup \{x_i \mapsto u\}\rangle$.

If $\mathbb{Q}_i = \forall^i$, then we set $\mathcal{A}_{i-1} = \overline{\mathcal{M}_i \cap_{x_i} \overline{\mathcal{A}_i}}$ Notice that $\mathcal{A}_{i-1}$ accepts a word assignment v iff for every $u \in \mathcal{L}(\mathcal{M}_i)$, it holds that $\mathsf{nfa}(\mathcal{A}_i)$ accepts $\langle v \cup \{x_i \mapsto u\}\rangle$.

Let $\mathbb{P}_i$ be the MNFH whose quantification condition is $\alpha_i = \mathbb{Q}_1^1 x_1 \mathbb{Q}_2^2 x_2 \cdots \mathbb{Q}_i^i x_i$, and whose underlying NFA is $\mathcal{A}_i$. Then, according to the construction of $\mathcal{A}_{i-1}$, we have that $\mathbb{M} \models \mathbb{P}_i$ iff $\mathbb{M} \models \mathbb{P}_{i-1}$.

The NFH $\mathbb{P}_1$ has a single variable, and we can now apply the base case. ■

5.4 Compositional Proof Rules for Model-Checking MNFH

We present an adaptation of the proof rules (PR) and ($\overline{\text{PR}}$) for automata.

Let $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle$ be a multi-NFA, and let $\mathbb{P} = (\Sigma, X, Q, Q_0, F, \delta, \alpha)$ be an MNFH with $\alpha = \mathbb{Q}_1^{i_1} x_1 \ldots \mathbb{Q}_m^{i_m} x_m$. Similarly to Section 3.4, the rule ($\text{PR}_{\mathcal{A}}$) aims at proving $\mathbb{M} \models \mathbb{P}$, and ($\overline{\text{PR}_{\mathcal{A}}}$) aims at proving the contrary, that is, $\mathbb{M} \models \neg\mathbb{P}$. Every model $\mathcal{A}_i$ in the rules is an *abstraction*. Since some models may be multiply quantified, a model $\mathcal{M}_i$ may have several different abstractions, according to the quantifiers under which $\mathcal{M}_i$ appears in α.
Denote $\mathbb{P}' = (\Sigma, X, Q, Q_0, F, \delta, \beta)$, where $\beta = \mathbb{Q}_1^1 x_1 \ldots \mathbb{Q}_m^m x_m$.

$$\frac{\forall i \in I_\forall.\ \mathcal{M}_{i_j} \models \mathcal{A}_i \quad \forall i \in I_\exists.\ \mathcal{A}_i \models \mathcal{M}_{i_j} \quad \langle \mathcal{A}_1, \ldots, \mathcal{A}_m \rangle \models \mathbb{P}'}{\langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle \models \mathbb{P}} \qquad (\text{PR}_{\mathcal{A}})$$

$$\frac{\forall i \in I_\forall.\ \mathcal{A}_i \models \mathcal{M}_{i_j} \quad \forall i \in I_\exists.\ \mathcal{M}_{i_j} \models \mathcal{A}_i \quad \langle \mathcal{A}_1, \ldots, \mathcal{A}_m \rangle \models \overline{\mathbb{P}'}}{\langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle \models \overline{\mathbb{P}}} \qquad (\overline{\text{PR}_{\mathcal{A}}})$$

Lemma 5.4.1. *The proof rules ($PR_{\mathcal{A}}$) and ($\overline{PR_{\mathcal{A}}}$) are sound and complete.*

Proof. The proof is similar to the proof of Lemma 3.4.1, since the correctness of our rules stems only from the semantics of the quantifiers. ■

Chapter 6

Learning-Based Multi-Property Model-Checking

We describe how to use automata learning to find approximations according to the proof rules ($\mathrm{PR}_{\mathcal{A}}$) and ($\overline{\mathrm{PR}_{\mathcal{A}}}$) described in Section 5.4, for the multi-models of MNFA and multi-properties of MNFH of Chapter 5. The L^* algorithm [3] is a learning algorithm that finds a minimal DFA for an unknown regular language U. We exploit the fact that MNFAs consist of regular languages to introduce an L^*-based algorithm for constructing approximations for the languages in the MNFA and for model-checking MNFH. To explain the idea behind our method, we first describe the L^* algorithm.

The L^* algorithm.

L^* consists of two entities: a *learner*, whose goal is to construct a DFA for U, and a *teacher*, who helps the learner by answering *membership queries* – "is $w \in U$?", and *equivalence queries* – "is A a DFA for U?". In case that $\mathcal{L}(A) \neq U$, the teacher also returns a counterexample: a word which is accepted by A and is not in U, or vice versa.

The learner maintains an *observation table* T that contains words for which a membership query was issued, along with the answers the teacher returned for these queries. Once T fulfills certain conditions (in which case we say that T is *steady*), it can be translated to a DFA A_T whose language is consistent with T. If $\mathcal{L}(A_T) = U$ then L^* terminates. Otherwise, the teacher returns a counterexample with which the learner updates T, and the run continues.

In each iteration, the learner is guaranteed to steady T, and L^* is guaranteed to terminate successfully. The sizes of the DFAs that the learner produces grow from one equivalence query to the next (while never passing the minimal DFA for U). The runtime of L^* is polynomial in the size of a minimal DFA for U and in the length of the longest counterexample that is returned by the teacher.

The main idea behind learning-based model-checking algorithms is to use the candidates produced by the learner as potential approximations. Since these candidates may be significantly smaller than the original models, model-checking is accelerated.

We first introduce our algorithm for the general case, in which L^* aims to learn the models themselves. Then, we introduce an improved algorithm in case that the quantification condition

is of the type $\forall^1\exists^2$, in which case we can both define stronger learning goals, and use the counterexamples provided by the model-checker to reach these goals more efficiently.

6.1 Learning Assumptions for General Multi-Properties

Consider an MNFA $\mathbb{M} = \langle \mathcal{M}_1, \mathcal{M}_2, \dots \mathcal{M}_k \rangle$, and an MNFH $\mathbb{P}$ with a quantification condition $\alpha = \mathbb{Q}_1^1 x_1 \mathbb{Q}_2^2 x_2 \cdots \mathbb{Q}_k^k x_k$. Algorithm L^*_{MNFH}, described in Algorithm 6.1, computes an over-approximation for every $\mathcal{M}_i$ under $\forall$, and an under-approximation for every $\mathcal{M}_i$ under $\exists$. It does so by running L^* for every $\mathcal{M}_i$ in parallel, aiming to learn $\mathcal{M}_i$. Thus, the learner maintains a set $T_1, \dots T_k$ of observations tables, one for every $\mathcal{M}_i$. Whenever all tables are steady, the learner submits the DFAs $\mathcal{A}_{T_1}, \dots \mathcal{A}_{T_k}$ that it produces as candidates for the approximations via an equivalence query. The result of the equivalence query either resolves $\mathbb{M} \models \mathbb{P}$ according to $(\text{PR}_{\mathcal{A}})$ and $(\overline{\text{PR}_{\mathcal{A}}})$, or returns counterexamples with which the learner updates the tables to construct the next round of candidates.

In Algorithm 6.1, The methods INITIALIZE and STEADY are learner functions used for initializing an observation table, and reaching a steady observation table, respectively. The method ADDCEX updates the table when a counterexample is returned from an equivalence query.

Handling membership queries is rather straightforward: when the learner submits a query w for an NFA $\mathcal{M}_i$, we return **true** iff $w \in \mathcal{L}(\mathcal{M}_i)$. We now describe how to handle equivalence queries.

Equivalence Queries.

The learner submits its candidate $\mathbb{A}$, which includes its set of candidates. We first check that they are approximations for $(\text{PR}_{\mathcal{A}})$, by checking whether $\mathcal{M}_i \models \mathcal{A}_{T_i}$ for every over-approximation and $\mathcal{A}_{T_i} \models \mathcal{M}_i$ for every under-approximation.

If all checks pass, then we model-check $\mathbb{A} \models \mathbb{P}$. If the check passes, we return $\mathbb{M} \models \mathbb{P}$. If the candidates are not approximations for $(\text{PR}_{\mathcal{A}})$ but are approximations for $(\overline{\text{PR}_{\mathcal{A}}})$, we model-check $\mathbb{A} \models \neg\mathbb{P}$. If the check passes, we return $\mathbb{M} \not\models \mathbb{P}$.

If none of the above has triggered a return value, then there exists at least one candidate $\mathcal{A}_i$ such that $\mathcal{L}(\mathcal{A}_i) \neq \mathcal{L}(\mathcal{M}_i)$. We can locate these candidates during the over- and under-approximation checks, while computing a word $w \in \mathcal{L}(\mathcal{M}_i) \setminus \mathcal{L}(\mathcal{A}_i)$ (in case that we found $\mathcal{A}_i$ not to be an over-approximation), or a word $w \in \mathcal{L}(\mathcal{A}_i) \setminus \mathcal{L}(\mathcal{M}_i)$ (in the dual case). We then return the list of counterexamples according to the candidates for which we found a counterexample.

Since L^* is guaranteed to terminate when learning a regular language, Algorithm 6.1 is guaranteed to terminate. The correctness of $(\text{PR}_{\mathcal{A}})$ and $(\overline{\text{PR}_{\mathcal{A}}})$ guarantee that L^*_{MNFH} terminates correctly at the latest after learning $\mathbb{M}$ (and terminates earlier if it finds smaller appropriate approximations).

Lemma 6.1.1. *Algorithm 6.1 terminated with the correct result.*

Proof. **Correctness:** The only two possibilities for outputs are $\mathbb{M} \models \mathbb{P}$ (Algorithm 11) and $\mathbb{M} \not\models \mathbb{P}$ (Algorithm 13). According to the equivalence query algorithm, we determine that $\mathcal{M} \models \mathbb{P}$ only when we can apply $(\text{PR}_{\mathcal{A}})$ on $\mathbb{A}, \mathbb{P}$, which is sound by the correctness of the proof

Algorithm 6.1 L^*_{MNFH}

Input: $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_k \rangle$, $\mathbb{P}$ with $\alpha = \mathbb{Q}_1^1 \pi_1 \ldots \mathbb{Q}_k^k \pi_k$.
Output: $\mathbb{M} \models \mathbb{P}$?

```
1: INITIALIZE(T_1, ... T_k)
2: while true do
3:    for i ∈ [1, k] do
4:       T_i = STEADY(T_i)
5:       Construct A_{T_i} from T_i
6:    end for
7:    𝔸 = ⟨A_{T_1}, A_{T_2}, ... A_{T_k}⟩
8:    (CexList, pass) = EQUIV(𝔸, 𝕄, ℙ)
9:    if CexList == null then
10:      if pass then
11:         return 𝕄 ⊨ ℙ
12:      else
13:         return 𝕄 ⊭ ℙ
14:      end if
15:   end if
16:   for (w_i, i) ∈ CexList do
17:      ADDCEX(T_i, w_i)
18:   end for
19: end while
```

rules. Similarly, we determine that $\mathcal{M} \not\models \mathbb{P}$ only when we can apply $(\overline{\text{PR}_{\mathcal{A}}})$ on $\mathbb{A}, \mathbb{P}$, which is sound by the correctness of the proof rules.

Termination: The goal of the L^* algorithm, is to learn the language of $\mathcal{M}_i$, for every $i \in [1, k]$. It is immediate that the membership queries are correct. Assume towards contradiction, that we do not halt due to equivalence queries. Since $\mathcal{M}_i$ is an NFA and in every equivalence query we obtain a counterexample which is correct according to $\mathcal{M}_i$, L^* terminates for $\mathcal{M}_i$ when $\mathcal{A}_i$ is isomorphic to $\mathsf{dfa}(\mathcal{M}_i)$, the equivalent DFA to $\mathcal{M}_i$. Eventually, this happens for every $i \in [1, k]$. In this case, the equivalence query cannot yield a spurious counterexample, resulting in termination. ■

6.2 Weakest Assumption for MNFH$_{\forall\exists}$

We introduce a *weakest assumption* in the context of multi-properties with a quantification condition $\forall\exists$. Intuitively, a weakest assumption is the most general language that can serve as an over-approximation. We prove that the weakest assumption is regular, and show how to incorporate it in a learning-based multi-property model-checking algorithm based on $(\text{PR}_{\mathcal{A}})$. We denote MNFH with a quantification condition of the form $\forall_1^1 x \exists_2^2 y$ by MNFH$_{\forall\exists}$. The weakest assumption is the goal of the learning Algorithm 6.2 below.

Definition 6.2.1. Let $\mathbb{M} = \langle \mathcal{M}_1, \mathcal{M}_2 \rangle$ be an MNFA and let $\mathbb{P}$ be an MNFH$_{\forall\exists}$. The *weakest assumption for* $\mathbb{P}$ *w.r.t.* $\mathcal{M}_2$ is as follows.

$$W^{\mathcal{M}_2:\mathbb{P}} = \bigcup_{\mathcal{A} \text{ s.t. } \langle \mathcal{A}, \mathcal{M}_2 \rangle \models \mathbb{P}} \mathcal{L}(\mathcal{A})$$

That is, $W^{\mathcal{M}_2:\mathbb{P}}$ is the union of all languages that along with $\mathcal{M}_2$ satisfy $\mathbb{P}$.

Lemma 6.2.2. *Let $\mathcal{A}$ and $\mathcal{M}_2$ be NFA, and $\mathbb{P}$ be an MNFH$_{\forall\exists}$. Then $\mathcal{L}(\mathcal{A}) \subseteq W^{\mathcal{M}_2:\mathbb{P}}$ iff $\langle \mathcal{A}, \mathcal{M}_2 \rangle \models \mathbb{P}$.*

Proof. If $\langle \mathcal{A}, \mathcal{M}_2 \rangle \models \mathbb{P}$ then the claim holds by the definition of $W^{\mathcal{M}_2:\mathbb{P}}$.

For the other direction, if $\mathcal{L}(\mathcal{A}) \subseteq W^{\mathcal{M}_2:\mathbb{P}}$, then for every $w \in \mathcal{L}(\mathcal{A})$ there exists an NFA $\mathcal{A}_w$, with $\mathcal{L}(\mathcal{A}_w) = \{w\}$ s.t. $\langle \mathcal{A}_w, \mathcal{M}_2 \rangle \models \mathbb{P}$. Therefore, for every $w \in \mathcal{L}(\mathcal{A})$, there exists a word $w' \in \mathcal{L}(\mathcal{M}_2)$ s.t. $\mathbb{P}$ accepts $\{w_x, w'_y\}$, and so by the semantics of MNFH, we have that $\langle \mathcal{A}, \mathcal{M}_2 \rangle \models \mathbb{P}$. ■

We note that a similar approach to Lemma 6.2.2 cannot work for general quantification conditions, since their satisfying assignments are generally not closed under union (See Appendix B).

6.2.1 Regularity of the Weakest Assumption

To justify using $W^{\mathcal{M}_2:\mathbb{P}}$ as the objective of a learning algorithm, we show that $W^{\mathcal{M}_2:\mathbb{P}}$ is regular.

Definition 6.2.3. Given two NFAs $\mathcal{A}_i = (Q_i, \Sigma_i, Q_0^i, \delta_i, F_i)$ for $i \in \{1, 2\}$, the *product automaton* of $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\mathcal{A}_1 \times \mathcal{A}_2 = (Q_1 \times Q_2, \Sigma_1 \times \Sigma_2, Q_0^1 \times Q_0^2, \delta_\times, F_1 \times F_2)$ where $\delta_\times((q, q'), (\sigma, \sigma')) = (\delta_1(q, \sigma), \delta_2(q', \sigma'))$.

Lemma 6.2.4. *Let $w, w' \in \Sigma^*$. Then, $w = \langle w_1, w_2 \rangle \in \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)$ iff $w_1 \in \mathcal{L}(\mathcal{A}_1)$ and $w_2 \in \mathcal{L}(\mathcal{A}_2)$.*

Proof. By induction on the length of $\langle w_1, w_2 \rangle$. ■

Definition 6.2.5. Let X be a set of variables, Σ be an alphabet and $\mathcal{A} = (Q, \hat{\Sigma}, Q_0, \delta, F)$ be an NFA. Denote $\mathcal{A}' = (Q, \Sigma, Q_0, \delta', F)$, where for every $q \in Q$ and $\sigma \in \Sigma$, the transition function is defined as follows.

$$\delta'(q, \sigma) = \bigcup_{\{(\sigma'_1)_{x_1}, \ldots, (\sigma'_k)_{x_k}\} \in \hat{\Sigma},\ (\sigma'_i)_{x_i} = \sigma} \delta(q, \{(\sigma'_1)_{x_1}, \ldots, (\sigma'_k)_{x_k}\})$$

The *projection of $\mathcal{A}$ to index $x_i \in X$*, denoted $\mathcal{A} \downarrow_{x_i}$, is an NFA for the language $\mathcal{L}(\mathcal{A}')/\{\#\}^*$.[1]

Lemma 6.2.6. *$w \in \mathcal{L}(\mathcal{A} \downarrow_{x_i})$ iff there exist words $w_1, \ldots, w_k \in \Sigma^*$ such that $w_i = w$ and $\langle w_1, \ldots, w_k \rangle \in \mathcal{L}(\mathcal{A})$.*

Proof. By definition. ■

Given a regular language $L \subseteq \Sigma^*$, $\mathcal{A}_L$ is an automaton that accepts L. Additionally, given NFAs $\mathcal{A}, \mathcal{B}$, the automaton $\mathcal{A} \cap \mathcal{B}$ is an NFA such that $\mathcal{L}(A \cap B) = \mathcal{L}(A) \cap \mathcal{L}(B)$ (such automaton exists since regular languages are closed for intersection).

[1] The *right quotient* L_1/L_2 is defined as the language $\{x \in \Sigma^* \mid \exists y \in L_2 \text{ s.t. } xy \in L_1\}$. Such NFA exists since regular languages are closed under right quotient.

Lemma 6.2.7. *Let $\mathbb{P}$ be an MNFH with a quantification condition $\alpha = \forall^1\exists^2$ and $\mathbb{M} = \langle \mathcal{M}_1, \mathcal{M}_2 \rangle$ be a multi-model. Denote by $\mathcal{M}'_2$ the NFA for $\mathcal{L}(\mathcal{M}_2) \cdot \{\#\}^*$. Then for every $w \in \Sigma^*$ it holds that:*

$$w \in W^{\mathcal{M}_2:\mathbb{P}} \text{ iff } w \in \mathcal{L}((\mathsf{nfa}\,(\mathbb{P}) \cap (\mathcal{A}_{\Sigma^* \cdot \{\#\}^*} \times \mathcal{M}'_2)) \downarrow_{x_1})$$

Proof.

$$\begin{aligned}
&w \in \mathcal{L}((\mathsf{nfa}\,(\mathbb{P}) \cap (\mathcal{A}_{\Sigma^* \cdot \{\#\}^*} \times \mathcal{M}'_2)) \downarrow_{x_1}) && \iff (\text{ Lemma 6.2.6})\\
&\exists w' \in \Sigma^* : \langle w, w' \rangle \in \mathcal{L}((\mathsf{nfa}\,(\mathbb{P}) \cap (\mathcal{A}_{\Sigma^* \cdot \{\#\}^*} \times \mathcal{M}'_2))) && \iff (\text{intersection})\\
&\exists w' \in \Sigma^* : \langle w, w' \rangle \in \mathcal{L}(\mathsf{nfa}\,(\mathbb{P})) \wedge \langle w, w' \rangle \in \mathcal{L}(\mathcal{A}_{\Sigma^* \cdot \{\#\}^*} \times \mathcal{M}'_2) && \iff (\text{Lemma 6.2.4})\\
&\exists w' \in \Sigma^* : \langle w, w' \rangle \in \mathcal{L}(\mathsf{nfa}\,(\mathbb{P})) \wedge w \in \Sigma^* \cdot \{\#\}^* \wedge w' \in \mathcal{L}(\mathcal{M}_2) && \iff\\
&\exists w' \in \Sigma^* : \langle w, w' \rangle \in \mathcal{L}(\mathsf{nfa}\,(\mathbb{P})) \wedge w \in \Sigma^* \wedge w' \in \mathcal{L}(\mathcal{M}_2) && \iff\\
&\exists w' \in \mathcal{L}(\mathcal{M}_2) : \langle w, w' \rangle \in \mathcal{L}(\mathsf{nfa}\,(\mathbb{P})) && \iff (\text{Definition 5.1.1})\\
&\exists w' \in \mathcal{L}(M_2) : \langle w, w' \rangle \models \mathbb{P} && \iff (\text{Lemma 6.2.2})\\
&\{w\} \subseteq W^{\mathcal{M}_2:\mathbb{P}} && \iff w \in W^{\mathcal{M}_2:\mathbb{P}}
\end{aligned}$$

That is, we can derive $W^{\mathcal{M}_2:\mathbb{P}}$ by taking the lefthand-side projection of the parallel run of $\mathsf{nfa}\,(\mathbb{P})$ with a multi-language consisting of an NFA that accepts all words in Σ^*, and $\mathcal{M}_2$ (while ignoring the # symbols). Intuitively, this projection includes all the words which can be matched with a word in $\mathcal{M}_2$ in a way that is accepted by $\mathsf{nfa}\,(\mathbb{P})$. We can therefore deduce the following.

Corollary 6.1. *$W^{\mathcal{M}_2:\mathbb{P}}$ is regular.*

6.3 Learning Assumptions for $\forall\exists$

Let $\mathbb{P}$ be an $\text{MNFH}_{\forall\exists}$ and let $\mathbb{M} = \langle \mathcal{M}_1, \mathcal{M}_2 \rangle$ be an MNFA. We now introduce our $L^*_{\forall\exists}$ learning-based algorithm for model-checking $\mathbb{M} \models \mathbb{P}$. As we have mentioned in Section 6.2, the learning goal in our $L^*_{\forall\exists}$ algorithm is $W^{\mathcal{M}_2:\mathbb{P}}$, as it is an over-approximation of $\mathcal{M}_1$. However, notice that every $\mathcal{A}$ such that $\mathcal{L}(\mathcal{M}_1) \subseteq \mathcal{L}(\mathcal{A}) \subseteq W^{\mathcal{M}_2:\mathbb{P}}$ suffices. $L^*_{\forall\exists}$ then runs L^* while using every DFA $\mathcal{A}$ that is produced by the learner during the run as a candidate for an over-approximation of $\mathcal{M}_1$.

We now describe our implementation for answering the membership and equivalence queries.

Membership Queries.

When the learner submits a membership query "$w \in^? \mathcal{L}(\mathcal{A})$", we model-check $\langle \mathcal{A}_w, \mathcal{M}_2 \rangle \models \mathbb{P}$, where $\mathcal{A}_w$ is a DFA whose language is $\{w\}$. If the check passes, then there exists a word $w' \in \mathcal{L}(\mathcal{M}_2)$ such that $\langle w, w' \rangle \models \mathbb{P}$. Therefore, we return **true**. Otherwise, $\langle w, w' \rangle \not\models \mathbb{P}$ for every $w' \in \mathcal{L}(\mathcal{M}_2)$, and thus we do not include w in $\mathcal{L}(\mathcal{A})$, and return **false**.

Equivalence Queries.

We first check that $\mathcal{A}$ is a potential over-approximation, by checking if $\mathcal{M}_1 \models \mathcal{A}$. If not, then we return a counterexample $w \in \mathcal{L}(\mathcal{M}_1) \setminus \mathcal{L}(\mathcal{A})$. Otherwise, we model-check $\langle \mathcal{A}, \mathcal{M}_2 \rangle \models \mathbb{P}$. If

the model-checking passed, then we can conclude $\mathbb{M} \models \mathbb{P}$. Otherwise, a counterexample w is returned for a word in $\mathcal{L}(\mathcal{M}_1)$ that has no matching word in $\mathcal{L}(\mathcal{M}_2)$. We now need to check if w is spurious. If $w \notin \mathcal{L}(\mathcal{M}_1)$, then we return w as a counterexample to the learner. Otherwise, we can conclude that $\mathbb{M} \not\models \mathbb{P}$.

Algorithm 6.2 $L^*_{\forall\exists}$

Input: An MNFH$_{\forall\exists}$ $\mathbb{P}$, an MNFA $\mathbb{M} = \langle \mathcal{M}_1, \mathcal{M}_2 \rangle$.
Output: $\mathbb{M} \models \mathbb{P}$?

```
1: INITIALIZE(T)
2: while true do
3:    T = STEADY(T)
4:    Construct A_T from T
5:    (cex, pass) = EQUIV(A_T, M, P)
6:    if cex then ADDCEX(T, cex)
7:    else
8:       if pass then
9:          return ⟨M1, M2⟩ ⊨ P
10:      else
11:         return ⟨M1, M2⟩ ⊭ P
12:      end if
13:   end if
14: end while
```

Since L^* is guaranteed to terminate when learning a regular language, $L^*_{\forall\exists}$ is guaranteed to terminate. In both cases, when $\mathbb{M} \models \mathbb{P}$ or $\mathbb{M} \not\models \mathbb{P}$, the correctness of Equation $\text{PR}_{\mathcal{A}}$ and the properties of $W^{\mathcal{M}_2:\mathbb{P}}$ guarantee that the algorithm terminates with a correct answer, at most after learning $W^{\mathcal{M}_2:\mathbb{P}}$ (and may terminate earlier if it finds a smaller appropriate over-approximation).

Lemma 6.3.1. *Algorithm 6.2 terminates with the correct result.*

Proof. In this algorithm, we aim to learn a DFA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{M}_1) \subseteq \mathcal{L}(\mathcal{A}) \subseteq W^{\mathcal{M}_2:\mathbb{P}}$. For a membership query on a word w, we add w to $\mathcal{A}$ iff $\langle \mathcal{A}_w, \mathcal{M}_2 \rangle \models \mathbb{P}$, which according to Lemma 6.2.2 holds iff $w \in W^{\mathcal{M}_2:\mathbb{P}}$. For an equivalence query, we check that $\mathcal{M}_1 \models \mathcal{A}$ and additionally, that $\langle \mathcal{A}, \mathcal{M}2 \rangle \models \mathbb{P}$. By Lemma 6.2.2, this means that $\mathcal{A} \subseteq W^{\mathcal{M}_2:\mathbb{P}}$. Thus, this query is also correct.

Correctness: The algorithm outputs that $\mathbb{M} \models \mathbb{P}$ when the equivalence query passes. This happens if $\mathcal{L}(\mathcal{M}_1) \subseteq W^{\mathcal{M}_2:\mathbb{P}}$, which implies that $\mathbb{M} \models \mathbb{P}$. The algorithm outputs $\mathbb{M} \not\models \mathbb{P}$, when the model checking algorithm on $\langle \mathcal{A}, \mathcal{M}_2 \rangle \models \mathbb{P}$, returns a non-spurious counterexample. This is a word $w \in \mathcal{L}(\mathcal{M}_1)$, for which there exists no word $w' \in \mathcal{M}_2$, with which $\langle w, w' \rangle \models \mathbb{P}$. Thus, by the semantics of MNFH, this means that $\mathbb{M} \not\models \mathbb{P}$.

Termination: Since the weakest assumption is regular, the learning algorithm terminates, at the latest, when reaching a minimal DFA for $W^{\mathcal{M}_2:\mathbb{P}}$. In this iteration, termination is guaranteed. ■

There are several advantages to using Algorithm 6.2 over Algorithm 6.1. First, $W^{\mathcal{M}_2:\mathbb{P}}$ may be smaller than $\mathcal{M}_1$ which leads to quicker convergence, and the algorithm may halt before reaching $W^{\mathcal{M}_2:\mathbb{P}}$, by finding an even smaller over-approximation. Second, there is no

need to complement $\mathcal{M}_1$ for the equivalence query, since we only check if $\mathcal{M}_1$ is contained in the candidate submitted by the learner (which is a DFA and can be easily complemented). Finally, in contrast to the previous algorithm, which cannot obtain counterexamples from the model-checking, by a similar reason mentioned Note 3.3.6, we can now use the more targeted counterexamples. These counterexamples are provided by the model-checking procedure, and they take into account the checked property. As such, the counterexamples are guaranteed to remove refuting parts from the abstractions. This, in turn, leads to faster convergence.

While we have defined the weakest assumption and Algorithm 6.2 for a quantification condition of the type $\forall\exists$, both can be easily extended to handle a sequence of $\exists$ quantifiers rather than a single one.

Chapter 7

Reducing the Alphabet Size

Often, different properties are checked on the same multi-model. This may cause the model to contain irrelevant information, which does not affect the satisfaction or refutation of a property.

In order to improve the approximation construction algorithms, we suggest a way to decrease the alphabet size. This may lead to smaller approximation in both abstraction-refinement based algorithm and automata-learning based algorithm.

The method described in the following sections may decrease the approximations in the following manner. In the abstraction-refinement based algorithm, several (previously disjoint) states may possibly be merged into one. Similarly, in the automata-learning based algorithm, when the alphabet of an automaton is decreased, we eliminate many transitions in the NFA, and consequently in the learned DFA.

In this chapter, we assume, without loss of generality, that each model is quantified exactly once, and that the quantification is in the same order as the models, as explained in Note 3.2.3.

7.1 Decreasing the Alphabet for MultiLTL

Let $\mathbb{P}$ be a $\mathsf{MultiLTL}_{\mathsf{NNF}}$ formula over AP and $\mathbb{M}$ be a multi-model over AP'.

Definition 7.1.1. Let $A_i \subseteq AP$ be the set of atomic propositions, which appears in $\mathbb{P}$, regarding the trace variable π_i. The *restricted model of $\mathcal{M}$ w.r.t. a trace variable π_i in $\mathbb{P}$* is $\mathcal{M}{\downarrow_{\pi_i:\mathbb{P}}} = (S, I, R, L{\downarrow_{\pi_i:\mathbb{P}}})$, where $L{\downarrow_{\pi_i:\mathbb{P}}}(s) = L(s) \cap A_i$, for every $s \in S$.

We now show that replacing the model $\mathcal{M}_i$ with $\mathcal{M}_i{\downarrow_{\pi_i:\mathbb{P}}}$ in the multi-model $\mathbb{M}$ does not affect the satisfaction of the formula $\mathbb{P}$.

Lemma 7.1.2. *Let $\mathbb{M}{\downarrow_i}$ be the multi-model obtained from $\mathbb{M}$ by replacing $\mathcal{M}_i$ with $\mathcal{M}_i{\downarrow_{\pi_i:\mathbb{P}}}$. Then, $\mathbb{M} \models \mathbb{P}$ iff $\mathbb{M}{\downarrow_i} \models \mathbb{P}$.*

Proof. Let $\mathbb{T} = \mathcal{L}(\mathbb{M})$ and $\mathbb{T}_i = \mathcal{L}(\mathbb{M}{\downarrow_{\pi_i:\mathbb{P}}})$. Given trace assignment Π, denote by $\Pi{\downarrow_i}$ the assignment that satisfies $\Pi{\downarrow_i}(\pi_i) = \Pi(\pi_i)_1 \cap A_i$ and is the same as Π for every other trace variable. We show by induction on the structure of $\mathbb{P}$, that for every assignment Π, it holds that $\Pi \models_{\mathbb{T}} \mathbb{P}$ iff $\Pi{\downarrow_i}(\pi_i) \models_{\mathbb{T}_i} \mathbb{P}$.

Base: For a_π, when $\pi \neq \pi_i$, it is immediate, since $\Pi(\pi) = \Pi{\downarrow_i}(\pi)$. For a_{π_i}, it holds that $a \in A_i$, which means that $\Pi \models_{\mathbb{T}} a_{\pi_i}$ iff $\Pi{\downarrow_i} \models_{\mathbb{T}_i} a_{\pi_i}$.

Step: For the operators $\neg, \wedge, \vee, \mathsf{X}, \mathsf{U}$ and R, the claim holds by their definitions. Additionally, the claim holds for quantifiers which do not refer to the i^{th} model. Therefore, it is enough to show only for $\exists^i$ and $\forall^i$ quantifiers.

- $\Pi \models_{\mathbb{T}} \exists^i \pi_i.\varphi(\pi_1, \ldots, \pi_i)$ iff there exists a trace $\tau \in \mathcal{L}(\mathcal{M}_i)$ such that $\Pi[\pi_i \to \tau] \models_{\mathbb{T}} \varphi(\pi_1, \ldots, \pi_i)$. This holds iff there exists a trace $\tau' \in \mathcal{L}(\mathcal{M}_i{\downarrow_{\pi_i:\mathbb{P}}})$, which is the restriction of τ to A_i, such that $\Pi{\downarrow_i}[\pi_i \to \tau'] \models_{\mathbb{T}_i} \varphi(\pi_1, \ldots, \pi_i)$. By the semantics of MultiLTL, this holds iff $\Pi{\downarrow_i} \models_{\mathbb{T}_i} \exists^i \pi_i.\varphi(\pi_1, \ldots, \pi_i)$.
- $\Pi \models_{\mathbb{T}} \forall^i \pi_i.\varphi(\pi_1, \ldots, \pi_i)$ iff for every trace $\tau \in \mathcal{L}(\mathcal{M}_i)$, it holds that $\Pi[\pi_i \to \tau] \models_{\mathbb{T}} \pi_i.\varphi(\pi_1, \ldots, \pi_i)$. This holds iff for every trace $\tau' \in \mathcal{L}(\mathcal{M}_i{\downarrow_{\pi_i:\mathbb{P}}})$, it holds that $\Pi{\downarrow_i}[\pi_i \to \tau'] \models_{\mathbb{T}_i} \varphi(\pi_1, \ldots, \pi_i)$. By the semantics of MultiLTL, this is iff $\Pi{\downarrow_i} \models_{\mathbb{T}_i} \forall^i \pi_i.\varphi(\pi_1, \ldots, \pi_i)$.

■

By applying Lemma 7.1.2 to every model $\mathcal{M}_i$ in the multi-model $\mathbb{M}$, we can decrease the size of the AP over which every model is defined, reducing the size of the approximations.

7.2 Decreasing the alphabet for MNFH

Let $\mathbb{P}$ be an MNFH, where the alphabet of $\mathbb{P}{\downarrow_{x_i}}$ is Σ_i, and let $\mathbb{M}$ be a MNFA.

Definition 7.2.1. The *restricted model of $\mathcal{M}$ w.r.t. a trace variable x_i in $\mathbb{P}$* is $\mathcal{M}{\downarrow_{x_i:\mathbb{P}}} = (\Sigma_i, Q, Q_o, \delta{\downarrow_i}, F)$, where for every $q, q' \in Q$ and $\sigma \in \Sigma_i$, it holds that $(q, \sigma, q') \in \delta{\downarrow_i}$ iff $(q, \sigma, q') \in \delta$.

We now show that replacing $\mathcal{M}_i$ with $\mathcal{M}_i{\downarrow_{x_i:\mathbb{P}}}$ in the multi-model $\mathbb{M}$ does not change the satisfaction of the MNFH.

Lemma 7.2.2. *Assume that $\mathcal{M}_i$ is existentially quantified in α. Let $\mathbb{M}{\downarrow_i}$ denote the multi-model obtained from $\mathbb{M}$ by replacing $\mathcal{M}_i$ with $\mathcal{M}_i{\downarrow_{x_i:\mathbb{P}}}$. Then, $\mathbb{M} \models \mathbb{P}$ iff $\mathbb{M}{\downarrow_i} \models \mathbb{P}$.*

Proof. Let v be a trace assignment. We show by induction on the number of quantifiers in α, that $\mathbb{M} \models_v (\alpha, \mathbb{P})$ iff $\mathbb{M}{\downarrow_i} \models_v (\alpha, \mathbb{P})$.

Base: When there are no quantifiers, then since the alphabet that refers to x_i is Σ_i, the claim holds.

Step: Since all other models are the same in $\mathbb{M}$ and $\mathbb{M}{\downarrow_i}$, it is enough to consider only the case for which $\alpha = \exists^i x_i.\alpha'$.

$\mathbb{M} \models_v (\exists^i x_i.\alpha', \mathbb{P})$ iff there exists a trace $\tau \in \mathcal{L}(\mathcal{M}_i)$ such that $\mathbb{M} \models_{v[x_i \to \tau]} (\alpha', \mathbb{P})$. Since the alphabet that refers to $\mathcal{M}_i$ in $\mathbb{P}$ is Σ_i, it holds that $\tau \in \Sigma_i^*$. The latter iff $\mathbb{M}{\downarrow_i} \models_{v[x_i \to \tau]} (\alpha', \mathbb{P})$. By the semantics of MNFH, this holds iff $\mathbb{M}{\downarrow_i} \models_v (\exists^i x_i.\alpha', \mathbb{P})$. ■

By applying Lemma 7.2.2 for all the existentially quantified models in $\mathbb{M}$, we can decrease the alphabet size of those NFAs. This allows the observation table in the L^* algorithm to be much smaller.

Note 7.2.3. For a universally quantified model $\mathcal{M}_i$, if there is a word $w \in \mathcal{L}(\mathcal{M}_i) \setminus {\Sigma_i}^*$, then when $v(x_i) = w$, the MNFH $\mathbb{P}$ cannot be satisfied, since there cannot be an accepting run for w in $\mathbb{P}$. Thus, in this case, it is immediate that $\mathbb{M} \not\models \mathbb{P}$.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

We have introduced multi-models and multi-properties – useful notions that generalize hyperproperties to handle multiple systems. We have formalized these notions for both finite-trace (terminating) and infinite-trace (reactive) systems, and presented compositional proof rules for model-checking multi-properties.

For infinite-trace systems, we have introduced MultiLTL, a generalization of HyperLTL, and have applied our proof rules in abstraction-refinement and CEGAR based algorithms. For finite-trace systems, we have introduced multi-NFH, which offer an automata-based specification formalism for regular multi-properties. Here, we have applied our proof rules in automata-learning algorithms. The algorithms for both approaches accelerate model-checking by computing small abstractions, that allow avoiding performing model-checking on the full multi-model.

We further enhanced these algorithms for the $\forall^*\exists^*$ fragments of multi-properties, by considering information from the multi-property itself. Thus, we eliminate spurious runs that obstruct the construction of the needed approximations, allowing faster convergence of the algorithm.

8.2 Future Work

Regarding Counterexamples. Counterexamples obtained by performing model-checking for multi-properties, are extractable only for the first k outermost $\forall$-quantifiers of the multi-property. This is due to the fact, that counterexamples for existential quantifiers are hard to define and obtain. This is also the case in the branching temporal logic CTL. Some works [20, 48, 21, 50] try to define a new notion of counterexamples for CTL or its fragments. Exploring similar ideas in the context of multi-properties might be the key for expanding the CEGAR framework to the full logic of MultiLTL.

Regarding Weakest Assumptions. Weakest assumptions, as described in Section 6.2, cannot be directly extended to other quantification conditions (See Appendix B). Exploring and defining new notions of weakest assumptions for other quantification conditions allows to extend the learning based algorithm for every multi-property described as an MNFH. Since hyperproperties and multi-properties are not closed under regular operations, finding the golden mean for this

definition, which is both regular and sound, may pose a challenge.

Appendix A

Some Additional Proof Rules

This appendix lists additional proof rules that were developed and investigated as part of the initial work on this dissertation. Those rules were the inspiration for the proof rules (PR), ($\overline{\text{PR}}$), ($\text{PR}_{\mathcal{A}}$) and ($\overline{\text{PR}_{\mathcal{A}}}$). When trying to extend those rules for more complex quantification conditions, the one-model semantics of HyperLTL was not enough.

A.1 Proof Rules for Hyperproperties

At first we explored the ideas of over- and under-approximation in the context of HyperLTL.

Definition A.1.1. $\exists^*$HyperLTL is the fragment of HyperLTL which contains all of the HyperLTL formulae with only existential quantifiers.

$\forall^*$HyperLTL is the fragment of HyperLTL which contains all of the HyperLTL formulae with only universal quantifiers.

The following are sound and complete proof rules for the aforementioned fragments:

$$\frac{\mathcal{P} \in \exists^*\text{HyperLTL} \quad \mathcal{A} \models \mathcal{P} \quad \mathcal{A} \models \mathcal{M}}{\mathcal{M} \models \mathcal{P}} \tag{A.1}$$

$$\frac{\mathcal{P} \in \forall^*\text{HyperLTL} \quad \mathcal{A} \models \mathcal{P} \quad \mathcal{M} \models \mathcal{A}}{\mathcal{M} \models \mathcal{P}} \tag{A.2}$$

Lemma A.1.2. *(A.1),(A.2) are both sound and complete.*

Proof. Let $\mathcal{P}$ be $\exists^*$HyperLTL formula with n quantifiers and $\mathcal{M}$ be a Kripke structure, both over the set AP.

Completeness: If $\mathcal{M} \models \mathcal{P}$, by choosing $\mathcal{A} = \mathcal{M}$ we obtain $\mathcal{A} \models \mathcal{P}$ and $\mathcal{A} \models \mathcal{M}$.

Soundness: Assume that $\mathcal{A} \models \mathcal{M}$ and $\mathcal{A} \models \mathcal{P}$. Therefore, there exist traces $\pi_1, \ldots, \pi_n \in \mathcal{L}(\mathcal{A})$ such that $\langle \pi_1, \ldots, \pi_n \rangle \models \mathcal{P}$. Since $\mathcal{A} \models \mathcal{M}$, it holds that $\pi_i \in \mathcal{L}(\mathcal{M})$ for every $i \in [1, n]$. Thus, by the semantics of HyperLTL, $\mathcal{M} \models \mathcal{P}$.

Let $\mathcal{P}$ be $\forall^*$HyperLTL formula with n quantifiers and $\mathcal{M}$ be a Kripke structure, both over the set AP.

Completeness: If $\mathcal{M} \models \mathcal{P}$, by choosing $\mathcal{A} = \mathcal{M}$ we obtain $\mathcal{A} \models \mathcal{P}$ and $\mathcal{M} \models \mathcal{A}$.

Soundness: Assume that $\mathcal{M} \models \mathcal{A}$ and $\mathcal{A} \models \mathcal{P}$. Therefore, for every traces $\pi_1, \ldots, \pi_n \in \mathcal{L}(\mathcal{A})$, it holds that $\langle \pi_1, \ldots, \pi_n \rangle \models \mathcal{P}$. Let $\pi_1, \ldots, \pi_n \in \mathcal{L}(\mathcal{M})$. Since $\mathcal{M} \models \mathcal{A}$, it holds that $\pi_1, \ldots, \pi_n \in \mathcal{L}(\mathcal{A})$, meaning that $\langle \pi_1, \ldots, \pi_n \rangle \models \mathcal{P}$. Thus, by the semantics of HyperLTL, $\mathcal{M} \models \mathcal{P}$. ■

Note that, the proof of completeness does not help in finding the over- or under-approximation. The rule is complete under the assumption that such approximations exists.

Appendix B

Discussing Weakest Assumptions

We consider different possibilities for definitions of weakest assumptions under different quantification conditions.

B.1 Weakest Assumptions for $\text{MNFH}_{\exists\forall}$

We denote MNFH with quantification condition of the form $\exists_1^1 x \forall_2^2 y$ by $\text{MNFH}_{\exists\forall}$. Consider the following definition:

Definition B.1.1. Let $\mathbb{M}$ be an MNFA and let $\mathbb{P}$ be an $\text{MNFH}_{\exists\forall}$. The *weakest assumption for* $\mathbb{P}$ *w.r.t* $\mathcal{M}_1$ is as follows.

$$W^{\mathcal{M}_1:\mathbb{P}} = \bigcup_{\mathcal{A} \text{ s.t. } \langle \mathcal{M}_1, \mathcal{A} \rangle \models \mathbb{P}} \mathcal{L}(\mathcal{A})$$

That is, $W^{\mathcal{M}_2:\mathbb{P}}$ is the union of all languages that along with $\mathcal{M}_1$ satisfy $\mathbb{P}$.

We show that soundness is violated for this definition. I.e. that $\mathcal{L}(\mathcal{A}) \subseteq W^{\mathcal{M}_1:\mathbb{P}}$ iff $\langle \mathcal{M}_1, \mathcal{A} \rangle \models \mathbb{P}$ does not hold. Intuitively, this definition is not sound since different models might satisfy an $\text{MNFH}_{\exists\forall}$ using a different witness for the existential quantifier.

The following example shows that there exist an $\text{MNFH}_{\exists\forall}$ and NFAs $\mathcal{M}_1, \mathcal{C}$ such that $\langle \mathcal{M}_1, \mathcal{C} \rangle \not\models \mathbb{P}$, yet $\mathcal{C} \models W^{\mathcal{M}_1}$.

Example B.1.2. Consider the regular languages L_1, L_2, L_3 and $L_{\mathbb{P}}$ over $\Sigma = \{a, b\}$.

$$L_1 = \{a, aa\} \qquad L_2 = \{b\} \qquad L_3 = \{bb\} \qquad L_P = \{(a, b)\}^*$$

Let $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ be NFAs for the languages L_1, L_2 and L_3 respectively. Let $\mathbb{P}$ be a $\text{MNFH}_{\forall\exists}$ such that $\mathcal{L}(\mathsf{nfa}(\mathbb{P})) = L_P$.

Notice that:

- $\langle \mathcal{M}_1, \mathcal{M}_2 \rangle \models \mathbb{P}$, since there exists $w = a \in L_1$ such that for every word $w' \in L_2$, it holds that $\langle w, w' \rangle \models \mathbb{P}$.
- $\langle \mathcal{M}_1, \mathcal{M}_3 \rangle \models \mathbb{P}$, since there exists $w = aa \in L_1$ such that for every word $w' \in L_2$, $\langle w, w' \rangle \models \mathbb{P}$.

Thus, according to the definition of $W^{\mathcal{M}_1:\mathbb{P}}$, both $L_2 \subseteq W^{\mathcal{M}_1}$ and $L_3 \subseteq W^{\mathcal{M}_1}$. Let $\mathcal{C}$ be an NFA such that $\mathcal{L}(\mathcal{C}) = L_2 \cup L_3$. Note that it holds that $\langle \mathcal{M}_1, \mathcal{C} \rangle \not\models \mathbb{P}$ – since both $\langle a, bb \rangle \not\models \mathbb{P}$ and $\langle aa, b \rangle \not\models \mathbb{P}$. Additionally, no other words can be in the $W^{\mathcal{M}_1:\mathbb{P}}$, by the definitions of $\mathbb{P}$ and L_1.

One might wonder whether changing the definition of the weakest assumption to be $\hat{W}^{\mathcal{M}_1:\mathbb{P}} = \bigcap_{\mathcal{A} \text{ s.t. } \langle \mathcal{M}_1, \mathcal{A} \rangle \models \mathbb{P}} \mathcal{L}(\mathcal{A})$ works. However, by consider the same models in Example B.1.2, we get that $\hat{W}^{\mathcal{M}_1:\mathbb{P}} = \emptyset$. Yet, we have seen that there exist an NFA $\mathcal{A}$ such that $\langle \mathcal{M}_1, \mathcal{A} \rangle \models \mathbb{P}$, and its language is not empty.

Another possibility is to fix $\mathcal{M}_2$, and try to define a weakest assumption as an under-approximation of $\mathcal{M}_1$. This results in a search for a witness word $w \in \mathcal{L}(\mathcal{M}_1)$ for the satisfaction of $\mathbb{P}$, which is equivalent to performing model-checking on $\langle \mathcal{M}_1, \mathcal{M}_2 \rangle \models \mathbb{P}$.

B.2 Weakest Assumption for MNFH$_{\forall\mathbb{Q}^*}$

We denote MNFH with quantification condition of the form $\forall_1^1 x_1 \mathbb{Q}_2^2 x_2 \dots \mathbb{Q}_n^n x_n$ by MNFH$_{\forall\mathbb{Q}^*}$. We prove in a similar manner to Section 6.2, that by fixing all models except of the first, soundness still holds and that the weakest assumption is regular. Thus, we can extend Algorithm 6.2 for MNFH$_{\forall\mathbb{Q}^*}$, by considering an approximation of the first model only.

Definition B.2.1. Let $\mathbb{M} = \langle \mathcal{M}_1, \dots, \mathcal{M}_n \rangle$ be an MNFA and let $\mathbb{P}$ be an MNFH$_{\forall\mathbb{Q}^*}$. The *weakest assumption for* $\mathbb{P}$ *w.r.t.* $\mathbb{M}_{-1}$ is as follows.

$$W^{\mathbb{M}_{-1}:\mathbb{P}} = \bigcup_{\mathcal{A} \text{ s.t. } \langle \mathcal{A}, \mathcal{M}_2, \dots, \mathcal{M}_n \rangle \models \mathbb{P}} \mathcal{L}(\mathcal{A})$$

That is, $W^{\mathbb{M}_{-1}:\mathbb{P}}$ is the union of all languages that along with $\mathcal{M}_2, \dots, \mathcal{M}_n$ satisfy $\mathbb{P}$.

Lemma B.2.2. *Let $\mathcal{A}$ be an NFA, $\mathbb{M}$ be multi-NFA, and $\mathbb{P}$ be an MNFH$_{\forall\mathbb{Q}^*}$ with quantification condition $\alpha = \forall_1^1 x_1 \mathbb{Q}_2^2 x_2 \dots \mathbb{Q}_n^n x_n$. Then $\mathcal{L}(\mathcal{A}) \subseteq W^{\mathbb{M}_{-1}:\mathbb{P}}$ iff $\langle \mathcal{A}, \mathcal{M}_2, \dots, \mathcal{M}_n \rangle \models \mathbb{P}$.*

Proof. If $\langle \mathcal{A}, \mathcal{M}_2, \dots, \mathcal{M}_n \rangle \models \mathbb{P}$ then the claim holds by the definition of $W^{\mathbb{M}_{-1}:\mathbb{P}}$.

For the other direction, denote $\alpha' = \mathbb{Q}_2^2 x_2 \dots \mathbb{Q}_n^n x_n$, and $\mathbb{P}'$ be as $\mathbb{P}$ with the quantification condition α' and $v : X \to \hat{\Sigma}^*$ be a trace assignment.

Assume that $\mathcal{L}(\mathcal{A}) \subseteq W^{\mathbb{M}_{-1}:\mathbb{P}}$ and let $w \in \mathcal{L}(\mathcal{A})$. It holds that $\langle \mathcal{M}_2, \dots, \mathcal{M}_n \rangle \models_v \mathbb{P}'$ since there exists an automaton $\mathcal{B}$ such that $\langle \mathcal{B}, \mathcal{M}_2, \dots, \mathcal{M}_n \rangle \models_v \mathbb{P}'$ and $w \in \mathcal{L}(\mathcal{B})$. Since this holds for every $w \in \mathcal{L}(\mathcal{A})$, by the semantics of MNFH, $\langle \mathcal{A}, \mathcal{M}_2, \dots, \mathcal{M}_n \rangle \models_v \mathbb{P}$. ■

B.2.1 Regularity of the Weakest Assumption

The following lemma is a generalization of Lemma 6.2.4.

Lemma B.2.3. $w = \langle w_1, \dots, w_n \rangle \in \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_n)$ *iff* $w_i \in \mathcal{L}(\mathcal{A}_i)$ *for every* $i \in [1, n]$.

Proof. By induction on the number of productions. ■

Lemma B.2.4. *Let $\mathbb{P}$ be an MNFH$_{\forall\mathbb{Q}^*}$ and $\mathbb{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_n \rangle$ be a multi-model. Denote by $\mathcal{M}'_i$ the automaton for the language $\mathcal{L}(\mathcal{M}_i) \cdot \{\#\}^*$. Then for every $w \in \Sigma^*$ it holds that*

$$w \in W^{\mathbb{M}_{-1}:\mathbb{P}} \text{ iff } w \in \mathcal{L}((\mathsf{nfa}\,(\mathbb{P}) \cap (\mathcal{A}_{\Sigma^* \cdot \{\#\}^*} \times \mathcal{M}'_2 \times \cdots \times \mathcal{M}'_n)) \downarrow_{x_1})$$

Proof. Denote $\mathcal{B} = (\mathcal{A}_{\Sigma^* \cdot \{\#\}^*} \times \mathcal{M}'_2 \times \cdots \times \mathcal{M}'_n)$ and $\bar{w}' = \langle w_2, \ldots, w_n \rangle$.

$$\begin{aligned}
& w \in \mathcal{L}((\mathsf{nfa}\,(\mathbb{P}) \cap \mathcal{B}) \downarrow_{x_1}) && \iff \text{(Lemma 6.2.6)} \\
& \exists \bar{w}' \in (\Sigma^*)^{n-1} : \langle w, \bar{w}' \rangle \in \mathcal{L}((\mathsf{nfa}\,(\mathbb{P}) \cap \mathcal{B})) && \iff \text{(intersection)} \\
& \exists \bar{w}' \in (\Sigma^*)^{n-1} : \langle w, \bar{w}' \rangle \in \mathcal{L}(\mathsf{nfa}\,(\mathbb{P})) \wedge \langle w, \bar{w}' \rangle \in \mathcal{L}(\mathcal{B}) && \iff \text{(Lemma B.2.3)} \\
& \exists \bar{w}' \in (\Sigma^*)^{n-1} : \langle w, w' \rangle \in \mathcal{L}(\mathsf{nfa}\,(\mathbb{P})) \wedge \\
& \qquad\qquad w \in \Sigma^* \wedge w_i \in \mathcal{L}(\mathcal{M}_i) \text{ for } i \in [2, n] && \iff \\
& \exists w_i \in \mathcal{L}(\mathcal{M}_i) \text{ for } i \in [2, n] : \langle w, \bar{w}' \rangle \in \mathcal{L}(\mathsf{nfa}\,(\mathbb{P})) && \iff \text{(Definition 5.1.1)} \\
& \exists w_i \in \mathcal{L}(\mathcal{M}_i) \text{ for } i \in [2, n] : \langle w, w_2, \ldots, w_n \rangle \models \mathbb{P} && \iff \text{(Lemma B.2.2)} \\
& \{w\} \subseteq W^{\mathbb{M}_{-1}:\mathbb{P}} && \iff w \in W^{\mathbb{M}_{-1}:\mathbb{P}}
\end{aligned}$$

We can therefore deduce that $W^{\mathbb{M}_{-1}:\mathbb{P}}$ is regular, meaning that we can use an algorithm similar to Algorithm 6.2 for model-checking general MNFH, by fixing all models except one.

Bibliography

[1] Erika Ábrahám and Borzoo Bonakdarpour. Hyperpctl: A temporal logic for probabilistic hyperproperties. In Annabelle McIver and András Horváth, editors, *Quantitative Evaluation of Systems - 15th International Conference, QEST 2018, Beijing, China, September 4-7, 2018, Proceedings*, volume 11024 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2018.

[2] Shreya Agrawal and Borzoo Bonakdarpour. Runtime verification of k-safety hyperproperties in hyperltl. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 239–252. IEEE Computer Society, 2016.

[3] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[4] Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. 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 3308–3314. AAAI Press, 2015.

[5] Dana Angluin and Dana Fisman. Learning regular omega languages. *Theoretical Computer Science*, 650:57 – 72, 2016. Algorithmic Learning Theory.

[6] Borja Balle and Mehryar Mohri. Learning weighted automata. In Andreas Maletti, editor, *Algebraic Informatics - 6th International Conference, CAI 2015, Stuttgart, Germany, September 1-4, 2015. Proceedings*, volume 9270 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2015.

[7] Gilles Barthe, Juan Manuel Crespo, and César Kunz. Relational verification using product programs. In Michael J. Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, volume 6664 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2011.

[8] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, pages 100–114. IEEE Computer Society, 2004.

[9] Béatrice Bérard, Stefan Haar, and Loïc Hélouët. Hyper partial order logic. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations*

of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India, volume 122 of *LIPIcs*, pages 20:1–20:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[10] Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput.*, 25(6):1268–1280, 1996.

[11] Sebastian Berndt, Maciej Liskiewicz, Matthias Lutter, and Rüdiger Reischuk. Learning residual alternating automata. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1749–1755. AAAI Press, 2017.

[12] Armin Biere. Bounded model checking. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 457–481. IOS Press, 2009.

[13] Brandon Bohrer and André Platzer. A hybrid, dynamic logic for hybrid-dynamic information flow. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 115–124. ACM, 2018.

[14] Borzoo Bonakdarpour and Bernd Finkbeiner. The complexity of monitoring hyperproperties. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 162–174. IEEE Computer Society, 2018.

[15] Borzoo Bonakdarpour and Bernd Finkbeiner. Program repair for hyperproperties. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis*, pages 423–441, Cham, 2019. Springer International Publishing.

[16] Borzoo Bonakdarpour and Sarai Sheinvald. Finite-word hyperlanguages. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications - 15th International Conference, LATA 2021, Milan, Italy, March 1-5, 2021, Proceedings*, volume 12638 of *Lecture Notes in Computer Science*, pages 173–186. Springer, 2021.

[17] Yu-Fang Chen, Azadeh Farzan, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Learning minimal separating dfa's for compositional verification. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 31–45, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[18] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proceedings of the 12th International Conference on Computer Aided Verification*, CAV '00, pages 154–169, London, UK, 2000. Springer-Verlag.

[19] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[20] Edmund M. Clarke, Somesh Jha, Yuan Lu, and Helmut Veith. Tree-like counterexamples in model checking. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 19–29. IEEE Computer Society, 2002.

[21] Edmund M. Clarke and Helmut Veith. Counterexamples revisited: Principles, algorithms, applications. In Nachum Dershowitz, editor, *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2003.

[22] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 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 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014.

[23] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010.

[24] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2619 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2003.

[25] Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019.

[26] Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. 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 121–139. Springer, 2019.

[27] Dennis Dams and Orna Grumberg. Abstraction and abstraction refinement. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 385–419. Springer, 2018.

[28] Rayna Dimitrova, Bernd Finkbeiner, Máté Kovács, Markus N. Rabe, and Helmut Seidl. Model checking information flow in reactive systems. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, volume 7148 of *Lecture Notes in Computer Science*, pages 169–185. Springer, 2012.

[29] Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Extending automated compositional verification to the full class of omega-regular languages. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 2–17, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[30] Azadeh Farzan and Anthony Vandikas. Automated hypersafety verification. 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 200–218. Springer, 2019.

[31] Bernd Finkbeiner, Lennart Haas, and Hazem Torfah. Canonical representations of k-safety hyperproperties. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 17–31. IEEE, 2019.

[32] Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing ømega-regular hyperproperties. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 40–63. Springer, 2020.

[33] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesis from hyperproperties. *Acta Informatica*, 57(1-2):137–163, 2020.

[34] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. In Shuvendu K. Lahiri and Giles Reger, editors, *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*, volume 10548 of *Lecture Notes in Computer Science*, pages 190–207. Springer, 2017.

[35] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. 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 30–48. Springer, 2015.

[36] Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[37] Patrice Godefroid, Michael Huth, and Radha Jagadeesan. Abstraction-based model checking using modal transition systems. In Kim Guldstrand Larsen and Mogens Nielsen, editors, *CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2001.

[38] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021.

[39] Christopher Hahn. Algorithms for monitoring hyperproperties. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification - 19th International Conference, RV 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11757 of *Lecture Notes in Computer Science*, pages 70–90. Springer, 2019.

[40] Tzu-Han Hsu, Cesar Sanchez, and Borzoo Bonakdarpour. Bounded model checking for hyperproperties. In Jan Friso Groote and Kim G. Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Luxembourg, Luxembourg, 27 March-1 April 2021. Proceedings, Part II*, Lecture Notes in Computer Science. Springer, 2021.

[41] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.

[42] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984.

[43] Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michał Szynwelski. Learning nominal automata. *SIGPLAN Not.*, 52(1):613–625, January 2017.

[44] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, and Taylor T. Johnson. Hyperproperties of real-valued signals. In Jean-Pierre Talpin, Patricia Derler, and Klaus Schneider, editors, *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2017, Vienna, Austria, September 29 - October 02, 2017*, pages 104–113. ACM, 2017.

[45] Corina S Păsăreanu, Dimitra Giannakopoulou, Mihaela Gheorghiu Bobaru, Jamieson M Cobleigh, and Howard Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods Syst. Des.*, 32(3):175–205, 2008.

[46] Riccardo Pucella and Fred B. Schneider. Independence from obfuscation: A semantic framework for diversity. *J. Comput. Secur.*, 18(5):701–749, 2010.

[47] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993.

[48] Natarajan Shankar and Maria Sorea. Counterexample-driven model checking. Technical report, SRI International, Menlo Park, CA 94025, 2003.

[49] Ron Shemer, Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. Property directed self composition. 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 161–179. Springer, 2019.

[50] Sharon Shoham and Orna Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. *ACM Trans. Comput. Log.*, 9(1):1, 2007.

[51] Marcelo Sousa and Isil Dillig. Cartesian hoare logic for verifying k-safety properties. In Chandra Krintz and Emery Berger, editors, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 57–69. ACM, 2016.

[52] Alex Spelten, Wolfgang Thomas, and Sarah Winter. Trees over infinite structures and path logics with synchronization. In Fang Yu and Chao Wang, editors, *Proceedings 13th International Workshop on Verification of Infinite-State Systems, INFINITY 2011, Taipei, Taiwan, 10th October 2011*, volume 73 of *EPTCS*, pages 20–34, 2011.

[53] Tachio Terauchi and Alexander Aiken. Secure information flow as a safety problem. In Chris Hankin and Igor Siveroni, editors, *Static Analysis, 12th International Symposium, SAS 2005, London, UK, September 7-9, 2005, Proceedings*, volume 3672 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2005.

[54] Weikun Yang, Yakir Vizel, Pramod Subramanyan, Aarti Gupta, and Sharad Malik. Lazy self-composition for security verification. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 136–156. Springer, 2018.

לתלמיד, המבוססת על רב-המערכת הנתונה. אחרת, אנו מבצעים בדיקת מודל על קבוצת הקירובים ביחס לתכונה. מאחר שהאוטומטים שהתלמיד בונה קטנים באופן יחסי, בדיקת המודל על המועמדים מהירה בהרבה ביחס לבדיקת המודל על רב-המערכת המקורית.

ב-[45], אלגוריתם הלמידה מנסה ללמוד את ההנחה-החלשה-ביותר W, שהיא השפה הרגולרית שמכילה את כל הריצות שמספקות את התכונה תחת הגבלות מסויימות. הבנייה של W מתבססת על קבלת דוגמאות נגדיות הנובעות מבדיקת המודל. אנו מגדירים את ההנחה-החלשה-ביותר לפרגמנט מוגבל של MNFH, עבורו ניתן לקבל דוגמה נגדית מבדיקת המודל, ואף מראים שההנחה רגולרית. לכן, אנו משתמשים בה כיעד לאלגוריתם הלמידה המשופר. שני שיפורים אלה – הפקת הדוגמאות הנגדיות מבדיקת המודל, ולמידת הנחה-החלשה-ביותר במקום את המבנה עצמו, מאפשרים התכנסות מהירה יותר של אלגוריתם בדיקת המודל לפרגמנט הזה.

של מבני קריפקה. הלוגיקה בה אנו מתמקדים נקראת MultiLTL, והיא הרחבה של הלוגיקה HyperLTL [22]. הלוגיקה MultiLTL מאפשרת כימות בעל-אינדקס, $\forall^i$ ו-$\exists^i$, שמתייחס לרכיב ה-i ברב-המערכת.

אנו מראים כי ישנה רדוקציה דו-כיוונית בין בעיית בדיקת המודל עבור HyperLTL ובעיית בדיקת המודל עבור MultiLTL. אנו מדגישים כי למרות ששתי בעיות בדיקת המודל הללו שקולות, הפורמליזם החדש חזק יותר, בכך שהוא מאפשר לבצע אפיון ואימות ישיר של רב-מערכות, באמצעות התייחסות מפורשת לרכיבים השונים.

אנו מתעלים יתרון זה באמצעות ניסוח של שני כללי הוכחה מודולריים, המבוססים על על-קירוב ותת-קירוב עבור כל רכיב בנפרד. כללי הוכחה אלה מאפשרים להוכיח סיפוק של על-תכונה או של שלילתה על-ידי רב-המערכת. אנו מציגים שתי שיטות לחישוב הקירובים שנמצאים בשימוש בכללי ההוכחה.

השיטה הראשונה מבוססת על אבסטרקציה ועידון. הקירובים מחושבים בהדרגה, החל מקירוב התחלתי גס, ומעודנים באמצעות דוגמאות נגדיות. שיטה זו ממומשת באמצעות שני אלגוריתמים. בשניהם, כאשר בדיקת המודל ביחס לעל-המערכת מצליח, אנו מסיקים כי רב-המערכת המקורית מספקת את התכונה. אחרת, דוגמה נגדית מוחזרת.

האלגוריתם הראשון משתמש בדוגמאות נגדיות שנובעות מרב-המערכת בלבד. לכל רכיב, אנו מוצאים התנהגות שיש להסיר מעל-הקירוב של הרכיב או להוסיפה לתת-הקירוב של הרכיב, ומבצעים עידון בהתאמה. האלגוריתם השני מותאם לפרגמנט מוגבל של הלוגיקה MultiLTL, בו הכימות בנוי מרצף כמתי $\forall$ ולאחריהם רצף כמתי $\exists$. פרגמנט זה הינו שימושי במיוחד למידול של תכונות אבטחה באמצעות על-תכונות, כגון אי-הפרעה (noninterference). באלגוריתם זה, הדוגמאות הנגדיות נובעות ישירות מכישלון של בדיקת המודל, ולכן מתייחסות גם למערכת וגם לתכונה הנבדקת. חשוב להדגיש, כי מאחר שהמבנים האבסטרקטיים לרוב קטנים משמעותית מהרכיבים המקוריים, גם בדיקת המודל עבורם מהירה יותר.

הלוגיקה MultiLTL והייצוג של רב-מערכות באמצעות מבני קריפקה, מתאימים לתיאור של מערכות שאינן עוצרות. כאשר רוצים להתאים את הייצוג לתכניות בהן העצירה מובטחת, יש לבחור בייצוג שונה. לפיכך, עבור מקרה זה, אנו מתמקדים בתיאור של רב-מערכות ורב-תכונות עבור מסלולי-ריצה סופיים. בהקשר הזה, אנו משתמשים באוטומטים סופיים אי-דטרמיניסטיים (NFA) לתיאור כל רכיב, ובוקטור שלהם לייצוג של רב-מערכת. עבור המפרטים, אנו משתמשים בהיפר-אוטומטים סופיים (NFH [16]). ניתן לחשוב על ייצוג זה כמקבילה הרגולרית של HyperLTL. בדומה ל-HyperLTL ניתן בקלות להתאים את NFH לתיאור רב-תכונות, באמצעות המודל של רב-אוטומט סופי (MNFH).

אנו מראים שבדומה למקבילה עבור ריצות אינסופיות, ישנה רדוקציה דו-כיוונית בין בעיית בדיקת המודל עבור NFH ובין בעיית בדיקת המודל עבור MNFH. לאחר מכן, אנו ממשיכים בתיאור של שיטה מודולרית לביצוע בדיקת מודל עבור MNFH. בדומה למקרה עבור ריצות אינסופיות, השיטה מנסה למצוא קירובים מתאימים לרב-המבנה. אולם הפעם, החיפוש מבוסס על למידת אוטומטים.

בבדיקת מודל מבוססת למידה [45] מחפשים מועמד אפשרי לקירוב באמצעות אלגוריתם למידת האוטומטים L^* [3]. באלגוריתם זה, התלמיד מנסה לבנות אוטומט סופי דטמיניסטי עבור שפה רגולרית לא ידועה $\mathcal{L}$ באמצעות שאילתות שייכות ("האם המילה w נמצאת בשפה $\mathcal{L}$?") ושאילתות שקילות ("האם אוטומט $\mathcal{A}$ הוא אוטומט עבור השפה $\mathcal{L}$"). עבור שאילתות אלה, התלמיד מקבל תשובה מהמורה, אשר יודע מהי שפת היעד. לפיכך, התלמיד ממשיך לבנות ולהציע אוטומט-מועמד, עד אשר המורה משיב בחיוב על שאילתת השקילות.

באלגוריתם שלנו, התלמיד בונה קבוצה של אוטומטים-מועמדים בכל איטרציה, מועמד לכל רכיב ברב-המערכת. הרעיון המרכזי הוא בהתייחסות למועמדים הללו בתור קירובים למבנה. כאשר שאילתת שקילות נשלחת, אנחנו (בתור המורה) בודקים האם המועמד שנשלח מתאים כקירוב לרכיב. אם לא, אנו מחזירים דוגמה נגדית

תקציר

לוגיקות טמפורליות, כדוגמת LTL, נמצאות בשימוש נרחב לאפיון התנהגות של תכניות. תכונת LTL מאפיינת קבוצה של מסלולי-ריצה, שכל אחד מהם מספק את התכונה הנדרשת. אולם, תכונות-מסלול אלה אינן בעלות כוח ביטוי מספק לתיאור מפרטים הנוגעים להמצאות פרצות אבטחה או העדרן.

המושג של על-תכונות, הכללה של תכונות-מסלול, מספק פורמליזם אחיד לתיאור תכונות המתייחסות לקבוצות של מסלולי-ריצה. על-תכונות מתאימות במיוחד לתיאור מפרטים של תכונות אבטחה. לדוגמה, התכונה של זרימת-מידע-מאובטחת יכולה להיות מאופיינת באמצעות חלוקה של משתני התכנית למשתנים פומביים, שחשופים לצופה מהסביבה, ומשתנים סודיים שלא אמורים להיות ניתנים לצפייה מהסביבה. זרימת-מידע-מאובטחת נשמרת במערכת אם לכל שני מסלולי-ריצה, אם הם מסכימים על כל ערכי המשתנים הפומביים בקלט, אז גם ערכי הפלטים הפומביים שלהם שווים, ללא קשר לערכים של המשתנים הסודיים בתכנית. תכונה זו לא ניתנת לתיאור באמצעות מסלול יחיד.

אף-על-פי שעל-תכונות שימושיות ביותר, הן עדיין מוגבלות – הן יכולות להתייחס למערכת אחת בכללותה בלבד. מערכות לרוב מורכבות ממספר רכיבים, ועולה הצורך לקשר בין ריצות בין רכיב אחד למשנהו. דוגמה בולטת לכך היא תכונת המגוון (*diversity*) [46]. תכונת המגוון מכלילה את דרישות האבטחה בכך שהיא מתייחסת למספר רכיבים של אותה המערכת. כל אחד מהרכיבים נדרש לממש את אותה הפונקציונליות, אך להיות שונה בפרטי המימוש מרעיו. כפי שמוזכר ב- [46], תכונת אבטחה זו, יכולה עקרונית, להיות מיוצגת באמצעות על-תכונה על מערכת יחידה שמהווה הרכבה של כל הרכיבים במערכת המקורית. ברם, ייצוג זו אינו טבעי ואינו יעיל.

אנו פותרים קושי זה על-ידי הצגה של פורמליזם המאפשר להתייחס באופן מפורש למערכת כקבוצת רכיבים הנקראת רב-מודל, בצירוף עם שפת מפרט MultiLTL, שיכולה לבטא רב-תכונות. שפת מפרט זו מאפשרת להתייחס מפורשות למסלולי-ריצה מרכיבים שונים ברב-מודל. פורמליזם זה מאפשר לנו לבטא באופן ישיר וטבעי תכונות כדוגמת תכונת המגוון, ללא צורך בבנייה מסועפת.

יתר-על-כן, הלוגיקה MultiLTL נהנית מיתרון המודולריות. לפיכך, אנו מציעים כללי הוכחה נאותים ושלמים, המבוססים על שימוש באבסטרקציה לכל רכיב בנפרד, מסוג על-קירוב או תת-קירוב. כך אנו משיגים יתרון נוסף.

לאחר מכן, אנו מציעים שתי שיטות לחישוב קירובים אלו. הראשונה, מבוססת על אבסטרקציה ועידון, בה אבסטרקציה התחלתית מעודנת באופן איטרטיבי באמצעות דוגמאות נגדיות, עד שקירוב מתאים נמצא. השניה, מותאמת למערכות עם מסלולי-ריצה סופיים, מוצאת קירובים באמצעות אלגוריתם הלמידה L^*. שתי שיטות אלה מסוגלות ליצור קירובים קטנים משמעותית מהמודלים המקוריים, ולכן יכולות לשמש להאצת אלגוריתמי בדיקת המודל של על-תכונות ורב-תכונות.

כעת, נתאר את העבודה בפירוט נוסף. אנו משתמשים בייצוג של מערכות באמצעות רב-מערכות, שהן סדרה

המחקר בוצע בהנחייתן של פרופסור ארנה גרימברג ודוקטור שרי שינוולד, בפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמר מאת המחבר ושותפותיו למחקר בכנס במהלך תקופת מחקר המגיסטר של המחבר:

Ohad Goudsmid, Orna Grumberg, and Sarai Sheinvald. Compositional model checking for multi-properties. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings*, volume 12597 of *Lecture Notes in Computer Science*, pages 55–80. Springer, 2021.

תודות

ראשית, ברצוני להודות למנחה שלי, פרופ׳ ארנה גרימברג, על היותה מנחה מדהימה, חברה ומקור השראה במהלך לימודיי. תודה לך על הפגישות השבועיות, על העידוד והסיוע הבלתי פוסקים לשיפור עבודתי, על הצגתי לעולם המדהים של המחקר ועל כך שגרמת לחוויה זו להיות מהנה. היה לי כבוד גדול לעבוד איתך וללמוד ממך.

כמו-כן, ברצוני להודות למנחה השותפה שלי, ד״ר שרי שינוולד, על תרומתה הכבירה לעבודה. תודה לך על מחוייבותך ועל יכולתך להפוך כל פגישה למועילה בעזרת רעיונות חדשים, הומור וחדות מחשבה. את מנחה מעולה ואדם מדהים, ואני מאד שמח שיצא לי לעבוד איתך במהלך המחקר הזה.

לבסוף, ברצוני להודות למשפחתי, על אהבתם ותמיכתם הבלתי נדלית. תודה לכם על כך שהאמנתם בי, דירבנתם אותי לסקרנות ולשאיפה גבוה.

אני מודה לטכניון ולמלגת מלווין ר. ברלין למחקר אבטחת סייבר על התמיכה הכספית הנדיבה בהשתלמותי.

בדיקת מודל מודולרית של רב-תכונות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

אוהד חאודסמיד

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
אדר התשפ"א חיפה פברואר 2021

בדיקת מודל מודולרית של רב-תכונות

אוהד חאודסמיד