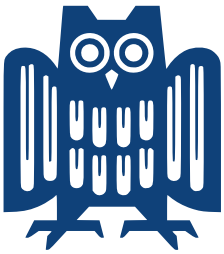

SAARLAND UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Bachelor's Thesis

Learning Hyperproperties

Author
Lennart Julius Haas

Supervisor
Prof. Bernd Finkbeiner, Ph.D.
Advisor
Hazem Torfah, M.Sc.

Reviewers
Prof. Bernd Finkbeiner, Ph.D.
Prof. Dr.-Ing. Holger Hermanns

12th September 2018

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum/Date) (Unterschrift/Signature)

Abstract

Hyperproperties formalize an important class of specifications including information-flow policies and symmetry requirements that are not expressible as traditional trace properties. They generalize trace properties to sets of sets of computation traces. In this thesis, we study an automata-theoretic approach for hyperproperties. We introduce a canonical automata representation for regular k-safety hyperproperties; the class of specifications for which violating systems contain a bad-prefix of size at most k and their k-folds form a regular language. And complement this approach by framework for learning our automata representations from unknown hyperproperties. In general, such hyperproperties can be represented by different automata using the k-fold self-composition of the alphabet. However this construction imposes an order on the traces in a set. Thus the representations are not unique. We define a notion of automata that are resistant under reordering of trace components; they are denoted permutation-complete.

We investigate the construction of permutation-complete automata from various representations, such as HyperLTL, nondeterministic safety automata, and deterministic bad-prefix automata. Our first construction is based on automata-transformations and our second one relies on a learning framework called L^*_{Hyper} that extends the L^*-framework for regular languages. In addition to the learner, we provide an implementation of the teacher deciding membership and equivalence queries for specifications given in HyperLTL. We conclude our work by presenting new decidability results concerning the learnability of hyper- and trace properties from queries and counterexamples. These include the undecidability of membership queries for safety languages and k-safety hyperproperties and the undecidability of the problem whether a HyperLTL formula expresses a k-safety hyperproperty.

Acknowledgements

First, I would like to express my deepest gratitude to my supervisor Prof. Bernd Finkbeiner for providing me with this challenging topic and especially, for the fruitful conversations we had together with Hazem. I also thank my advisor Hazem without whom this thesis would not have been possible. I benefited a lot from our intense conversations and his advice. Furthermore, I want to thank my second reviewer Prof. Dr.-Ing. Holger Hermanns.

I would like to give special thanks to Philip and Nick for proofreading this thesis. Besides, I am grateful to Jana, Carsten, Marvin and Maximilian for all the joyful moments we shared in the lab, the moral support I was given by them, and the valuable and interesting discussions we had.

Last, but not least, I would like to thank my family and my love Jana, for their ongoing and loving support during all situations of my life.

Contents

Chapter 1

Introduction

Hyperproperties define an important class of specifications which relate different executions of a system instead of exclusively reasoning about single traces. They extend the classic notion of trace properties: trace properties are sets of traces, whereas hyperproperties are sets of trace properties. Thus, a system satisfies a trace property if its set of traces is *contained* in the trace property; whereas it satisfies a hyperproperty if its set of traces is a trace property that is an *element* of the hyperproperty. Hyperproperties cover many important specifications such as information-flow policies, symmetry or error-resistant codes, which are not expressible in classic *trace properties* [10, 35, 11, 16]. One prominent example of hyperproperties is *Observational Determinism* (OD), defined by Zdancewic and Myers [36], which affects security-critical systems, i.e., systems in which two types of inputs and outputs—low- and high-security— are distinguished with respect to the permissions of the observer. OD requires executions with equal public initial input to agree on their entire public outputs; that way systems that satisfy OD behave deterministically with respect to the initial input to public observers. Note that a policy such as OD requires the presence of at least two distinct execution traces to be violated.

Following the increasing attention for hyperproperties, different logics, such as HyperLTL, HyperSTL or HyperCTL, have been introduced [11, 28]. In addition to these logics a variety of algorithms to solve problems, such as model-checking, satisfiability, and synthesis were introduce [11, 16, 15, 18]. Most of these approaches make use of automata-theoretic constructions, even though hyperproperties cannot be completely characterized by finite state automata. It was shown by Alur et al. that even fairly simple policies, like non-interference, are non-regular [2]. A classical workaround is to consider the k-fold parallel self-composition of a system S that is the composition of k-renamed versions of S. This approach allows one to relate up to k traces in a single trace; Clarkson and Schneider even showed that this approach is complete for the set of k-safety hyperproperties [10].

One purpose of this thesis is to investigate the *automata-theoretic* approach for hyperproperties. Therefore we introduce a canonical automata representation for *regular k-safety hyperproperties*; k-safety hyperproperties for which the set of bad-prefixes can be represented by a regular language. The automata-theoretic approach primarily emerged from the idea of representing specifications

for temporal logics by automata and developing algorithms for these representations [34]. As a consequence, it is possible to verify, synthesize, and monitor a wider range of properties in a uniform manner, regardless of the specification language used. In addition, we present a learning algorithm for these representations adapting Dana Angluin's learning framework L*[3].

Our representations are finite traces over the alphabet Σ^k where we denote k as the arity. Such representations are called *k-representations* and they can be interpreted as a set of up to k traces by projecting each position to the same component. In order to better understand the interpretation of k-representations, consider the finite trace t:

$$t = (\{a\}, \{\})(\{\}, \{a\})^5 \qquad\qquad T = \{\{a\}\{\}^5, \{\}\{a\}^5\}$$

over the alphabet $\mathcal{P}(\{a\})^2$ of length 6, which is a 2-representation of T.

Following this approach and the fact that violations of k-safety hyperproperties can be detected in k traces, we can represent k-safety hyperproperties by finite automata over Σ^k instead of Σ. Each trace in the resulting language can be interpreted as a trace property by the set of traces represented. Therefore, a regular k-safety hyperproperty can be represented as a language of k-representations. An ω-word safety automaton accepting such a language over the alphabet Σ^k is denoted a k-safety automaton. The following example should familiarize the reader with the notion of automata for regular k-safety hyperproperties:

Example: Consider the following hyperproperty expressed in HyperLTL:

$$\varphi := \forall\pi.\forall\pi'.\ \square(a_\pi \rightarrow a_{\pi'})$$

φ enforces that for all pairs of traces it holds: Whenever one trace satisfies a, then every trace has to satisfy a. First note that φ is a 2-safety hyperproperty since for any system violating φ it suffices to consider two traces in order to find a witness, namely a position i where a holds on one trace but not on the other trace. Furthermore φ is regular since it is expressed in HyperLTL. One example of a bad-prefix for φ is the set of finite traces $T := \{t_1, t_2, t_3\}$ with

$$t_1 := \{a\}\{\}\{a\} \qquad t_2 := \{a\}\{a\}\{a\} \qquad t_3 := \{a\}\{a\}\{\}$$

T is a bad-prefix because t_2 satisfies a in its second position and t_1 does not. However, because φ is 2-safety, there must exist a smaller bad-prefix consisting of only two traces:

$$T' := \{t_1, t_2\}$$

The bad-prefix can be further reduced in its length as the violation already occurs in the second position. Thus $T'' = \{\{a\}\{\}, \{a\}\{a\}\}$ is a bad-prefix; it is denoted a *minimal* bad-prefix for φ. Following the notion of a k-representation, s as well as s' are 2-representations of T'':

$$s = (\{a\}, \{\})(\{a\}, \{a\}) \qquad s' = (\{a\}, \{a\})(\{a\}, \{\})$$

Figure 1.1: A permutation-complete 2-bad-prefix automaton (left) and a non-permutation-complete 2-bad-prefix automaton (right) for $\forall\pi.\forall\pi'.\ \square(a_\pi \rightarrow a_{\pi'})$

The above example visualizes the fact that, unlike the set representation of bad-prefixes, their k-representations are not unique and consequently, the automata-representation of k-safety hyperproperties is not unique as well. For example, consider the two automata depicted in Figure 1.1, each accepting 2-representations of bad-prefixes of φ. We define an automaton to be a *k-bad-prefix automaton* for φ if it accepts a k-representation of a prefix of every set of traces violating φ but not necessarily all k-representations. Thus both automata in Figure 1.1 are 2-bad-prefix automata for φ. Consequently, we can classify k-bad-prefix automata according to whether they accept all representations. Such automata are called *permutation-completeness*.

Following the terminology for bad-prefix automata introduced by Kupferman and Vardi [24], we call a k-bad-prefix automata *tight* if it accepts representations for all bad-prefixes of minimal length and *fine* otherwise. Note that permutation-completeness of a k-bad-prefix automaton is not required for tightness and tightness does not imply permutation-completeness.

It is often crucial to determine the minimal k such that some hyperproperty is k-safety—the complexity of many algorithms dealing with k-safety hyperproperties highly depends on the order of k. We classify a bad-prefix *minimal* if it is minimal in the length as well as the number of traces. Correspondingly, a k-bad-prefix automaton is called *minimal* if its state space is of minimal size and the arity k is minimal.

This yields a canonical characterization of regular k-safety hyperproperties by minmial, tight, permutation-complete, deterministic k-bad-prefix automata. Thus, our formalism marks the first step towards a canonical representation of general hyperproperties.

Following the above notion of representations of k-safety hyperproperties, we introduce the first *learning algorithm for hyperproperties*, which learns minmial, tight, permutation-complete, deterministic k-bad-prefix automata for some unknown regular k-safety hyperproperty. The algorithm is based on Dana Angluin's L* algorithm for learning regular languages from queries and counterexamples [3]. Our algorithm extends previous work that applied L* to richer languages and different models of computation. Thus, it bridges the gap between the large set of applications of the L*-framework in the context of formal methods and the comparably recent class of hyperproperties.

Following the classic approach of learning algorithms given by Angluin [3], our learning framework consists of two components: a learner and a teacher. The learner can pose membership queries, i.e, 'Is some set T a bad-prefix?' and equivalence queries, i.e., 'Is some automaton $\mathcal{A}$ a k-bad-prefix automaton for the target hyperproperty?'. In the end, the learner outputs a minimal

k-BPA accepting all representations of all minimal k-bad-prefixes. In addition to the learner, we provide algorithms for answering membership and equivalence queries for hyperproperties expressed in HyperLTL.

We conclude our work by presenting a collection of decidability results concerning the learnability of hyperproperties from queries and counterexamples, which heavily affect the possible future extensions of our work. At first, we show that membership queries for safety languages are in general undecidable. Our result uses the undecidability of the halting problem and it propagates immediately to the undecidability of such queries for k-safety and safety hyperproperties. Furthermore, we show that, unlike for trace properties in LTL, it is undecidable whether a hyperproperty given in HyperLTL is safety. We make use of the undecidability of satisfiability of formula given in HyperLTL by Finkbeiner and Hahn [15]. Finally, we characterize a fragment of HyperLTL formulas for which we can decide whether the hyperproperty described is safety.

Related Work

Clarkson and Schneider introduced hyperproperties as sets of trace properties [10]. Their approach unified a variety of previously defined information-flow policies such as noninterference, introduced by Goguen and Meseguer [20], or observational determinism, defined by Zdancewic and Myers [36], and a variety of previous approaches to classify information-flow policies, such as Schneider's prior approach to classify the enforceable security policies [32]. Moreover, they introduce the class of k-safety hyperproperties, which is of special interest to our work.

Hyperproperties allow for studying the verification and synthesis of specifications in a unified manner. One logic that evolved out of this approach is Clarkson et al.'s HyperLTL [11]. HyperLTL is an extension of LTL that is able to express non-trace properties by explicitly quantifying different traces and relating them with basic LTL-operators. They provided an efficient model-checking algorithm for HyperLTL$_2$ (the fragment of HyperLTL containing all formulas with at most two quantifier alternations) based on the k-fold self-composition of the system under investigation, this method relates to our definition of k-bad-prefixes. The only prior approach, to the best of our knowledge, investigating an automata-theoretic approach to characterizing information-flow policies is by D'Souza et al. [13]. They establish language-based operations for Mantel's Basic Security Predicates (BSP), which are regularity preserving and extend this approach to verifying finite-state systems by considering their induced regular language under the application of BSP operations. That way, they reduce model-checking of information-flow policies expressed in Mantel's BSP to language inclusion of regular languages. Their approach is limited to information-flow policies expressed in BSP whereas our approach is independent of any logic and characterizes a large class of hyperproperties.

Besides model-checking, a collection of runtime-verification algorithms for hyperproperties have been introduced. Bonakdarpour and Agrawal, for example, developed a Petri net-based approach to monitor k-safety hyperproperties [1]. They use deterministic finite automata to

monitor single executions according to LTL sub-formulas and utilize Petri nets to memorize dependencies within different execution traces and sub-formulas. Whereas Finkbeiner et al. presented a rather automata-based approach [17]. Their approach monitors k-safety hyperproperties expressed in HyperLTL using finite automata reasoning about the k-fold of a system, i.e., they use non-permutation complete automata and check for all permutations of the system whether or not they satisfy φ. One major contribution of their work is the identification of different properties of the LTL part, which enable one to consider only a fraction of all permutations for specific HyperLTL formulas. Restricting themselves to deterministic programs, Pinisetty et al. obtained a monitoring algorithm for hypersafety properties by reducing the problem to trace properties [29].

Besides hyperproperties, our is heavily influenced by Angluin's L*. The idea of learning unknown concepts, such as languages, has been heavily studied in the context of two components communicating over queries; Sammut and Banerji introduced a query-based learning algorithm 'Marvin' posing queries to a teacher and learning concepts expressed in first-order logic [31]. Their work was shifted to the study of formal languages by Angluin in elaborating the different kinds of query types for learning different types of languages [5].

In the area of formal methods, one of the most influential algorithms rising from the area of concept learning by queries was Angluin's L* [3]. The algorithm learns a minimal deterministic finite automaton for some regular language while making use of two different types of queries, membership, i.e., 'is a string x in a language $\mathcal{L}$?', and equivalence queries, i.e., 'does an automaton $\mathcal{A}$ accept exactly the language $\mathcal{L}$?'. In particularly L*'s running time complexity is interesting: It is polynomially bounded in the size of the output, i.e., it is output-sensitive, unlike most standard algorithms. Thus L* greatly improves over algorithms using only one type of queries—it was shown by Angluin that only equivalence queries yield an exponential lower-bound and only membership queries obviously do not suffice for learning unknown regular languages [4].

Upon the development of L*, many extensions were introduced applying the power of learning to formal system verification. For example, Cobleigh et al. [12]. They make use of L* to verify composed systems against safety properties. Consider a composed system $S \mid\mid T$ consisting of two components S and T. They employ L* to learn an assumption P such that $\langle \emptyset \rangle\, S\, \langle P \rangle$ holds, i.e., under no assumptions ($\emptyset$) S satisfies P, and $\langle P \rangle\, T\, \langle \varphi \rangle$ holds where φ is a safety property, i.e., under assumption P on the environment of T it follows that φ is satisfied. Their approach does not rely on the explicit construction of the composed system and is therefore efficient in practice.

Besides applications of L*, many approaches aimed at the extension of L*'s expressive power to different languages other than regular ones as well as to different machine models more expressive then DFAs. Approaches to extend L* to richer machine models include the work of Drewes et al. [14]. they replaced finite automaton by symbolic automata and thus enabled the learning of infinite size alphabets while sticking to finite word languages.

Maler and Pnueli initialized the work on ω languages over a finite alphabet by learning infinitary languages represented by DBAs and co-DBAs [26]. They use ultimately-periodic ω-words ($uv^\omega, u, v \in \Sigma^*$) to identify ω-regular languages in the observation table and use a transition graph in order to represent the learned Büchi, co-Büchi automaton. Their work was extended

to a variety of approaches learning richer ω-regular languages using \$-languages to represent ω-words and families of deterministic finite acceptors to during the learning process [7, 25]. Angluin and Fissman recently initialized the work on learning tree-automata for infinite ω-tree languages [6]. They provide a reduction of their problem to learning ω-regular languages, which is possible since ω-tree languages are derived from ω-regular languages. Thus, improvements in the context of learning ω-regular languages directly translate into improvements on learning ω-tree languages.

Outline

The remainder of this thesis is structured as follows. We first recap the necessary background to understand our constructions and define the employed logic HyperLTL in Chapter 2. Afterwards we formalize the notion of permutation-complete k-bad-prefix automata and inspect their construction with respect to a HyperLTL formula or non-permutation-complete k-bad-prefix automata in Chapter 3. Based on the foregoing definition Chapter 4 then provides a learning framework L^*_{Hyper} devoted to minmial, tight, permutation-complete, deterministic k-bad-prefix automata We give an example of our framework put in use to learn HyperLTL formula and conclude this chapter by analysing termination, correctness and running time of L^*_{Hyper}. In order to complete the learning algorithm, we provide algorithms for answering the posed queries for regular k-safety hyperproperties expressed in HyperLTL in Chapter 5. In Chapter 6, we give some basic theoretic result about the boundaries of learning hyperproperties and k-safety hyperproperties. The entire work is concluded in Chapter 7 together with a collection of potential future work on this topic.

Chapter 2

Background

In the following chapter we outline the basic definitions and notations used to understand the thesis. We start in Section 2.1 by recapitulating a selection of automata together with the well-known classification of regular languages due to Myhill-Nerode with respect to continuation languages. In Section 2.2 we establish the basic notion of trace and safety properties together with corresponding automata constructions due to Kupferman and Vardi [24]. We provide the definition of hyperproperties paired with some recent results in Section 2.3, accompanied by the syntax and semantics of HyperLTL, a temporal logic for expressing hyperproperties in Section 2.4. The last section's purpose is to introduce Dana Angluin's L* algorithm, which forms the foundations of L$^*_{Hyper}$.

We start with some basic notations. Let AP denote a finite set of *atomic propositions*. We define the finite alphabet $\Sigma = \mathcal{P}(\mathsf{AP})$ where $\mathcal{P}(X)$ is the power-set of X and we denote an element of Σ a *label*. An infinite sequence over Σ is called an (*infinite*) *trace* and we write the set of all traces as $\Sigma^\omega = \{\alpha_0\alpha_1 \cdots \mid \forall i \in \mathbb{N}.\ \alpha_i \in \Sigma\}$. The set of all *finite* traces is denoted by $\Sigma^* = \bigcup_{n\in\mathbb{N}}\{\alpha_0 \dots \alpha_n \mid \forall i \leq n.\ \alpha_i \in \Sigma\}$. For finite or infinite sequences $\sigma = \alpha_0\alpha_1 \dots$ and $i \leq j \leq |\sigma|$, $\sigma[i,j] = \alpha_i \dots \alpha_j$ denotes the finite substring from position i to j and for $\sigma \in \Sigma^\omega$: $\sigma[i,\infty] = \alpha_i\alpha_{i+1} \dots$ denotes the infinite suffix of σ starting at position i. Furthermore $\sigma[i]$ is a shorthand for $\sigma[i,i]$. For σ, a finite trace, and τ, a finite or infinite trace, we denote their *concatenation* by $\sigma \cdot \tau$. We say that σ is a *prefix* of τ denoted by $\sigma \leq \tau$ if $|\sigma| \leq |\tau| \wedge \forall\, 1 \leq i \leq |\sigma|.\ \sigma[i] = \tau[i]$. The prefix relation is extended to sets of traces in the following way: For $U, V \subseteq \Sigma^* \cup \Sigma^\omega$, we define $U \leq V$ if $\forall \sigma \in U.\ \exists \tau \in V.\ \sigma \leq \tau$.

2.1 Automata

Our underlying model for languages are various kinds of automata. A *nondeterministic finite automaton* (NFA) is defined as a tuple $\mathcal{A} = (Q, \Sigma, q_0, F, \Delta)$, where Q denotes a finite set of states, Σ a finite alphabet, q_0 a designated initial state, $F \subseteq Q$ the set of accepting states and $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$ the transition relation that maps a state $q \in Q$ and a letter $a \in \Sigma$ to the set of successor states $\Delta(q, a) \subseteq Q$.

A *run* in $\mathcal{A}$ on a finite word $w = w_0 \dots w_n \in \Sigma^*$ is defined as a mapping $r : \{0, \dots, n+1\} \to Q$

from a position to the state of $\mathcal{A}$ currently reached with $r(0) = q_0$ and $r(i+1) \in \Delta(r(i), w_i)$ for all $0 \leq i \leq n$. A run r is *accepting* if $r(n+1) \in F$. The set of all finite traces accepted by an automaton $\mathcal{A}$ is called its language, denoted by $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$. $\mathcal{A}$ is called *minimal* if all automata with less states than $\mathcal{A}$ accept a different language.

A special case of finite automata are deterministic finite automata (DFA) for which the set of successor states contains at most 1 element for every state q and label a, i.e., $|\Delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. Its transition relation may equivalently be seen as a partial mapping $\delta : Q \times \Sigma \nrightarrow Q$. For a deterministic finite automata $\mathcal{A}$ there exists a minimal DFA $\mathcal{A}'$ accepting $\mathcal{L}(\mathcal{A})$, which is unique up to renaming of states.

With respect to languages over infinite traces we define a nondeterministic *Büchi automaton* (NBA) $\mathcal{B} = (Q, \Sigma, q_0, F, \Delta)$. A run of $\mathcal{B}$ on an infinite trace $w = w_1 w_2 \cdots \in \Sigma^\omega$ is a mapping $r : \mathbb{N} \to Q$ with $r(i+1) \in \delta(r(i), w_i)$ for all $i \in \mathbb{N}$. A run r is accepting if infinitely many accepting states are visited in r, i.e., there exists infinitely many $i \in \mathbb{N}$ such that $r(i) \in F$. Similar to finite word automata we can distinguish deterministic and nondeterministic Büchi automata.

For some of our later proofs we require a basic understanding about the size of a minimal DFAs accepting some regular language $\mathcal{L} \subseteq \Sigma^*$. We define the set of finite traces which can extend a finite trace $t \in \Sigma^*$ to become a member of $\mathcal{L}$ as t's continuation language denoted by $F_\mathcal{L}(t) = \{u \in \Sigma^* \mid t \cdot u \in \mathcal{L}\}$ and denote the continuation languages of $\mathcal{L}$ by $\mathcal{F}_\mathcal{L} = \{F_\mathcal{L}(x) \mid x \in \Sigma^*\}$. Using the Myhill-Nerode Theorem it follows [22]:

Theorem 2.1 [22]
A language $\mathcal{L}$ is regular if and only if $\mathcal{F}_\mathcal{L}$ is of finite size and $|\mathcal{F}_\mathcal{L}|$ denotes the minimal size of a DFA $\mathcal{A}$ accepting $\mathcal{L}$.

2.2 Trace Properties

Trace properties define specifications that are dependent on single executions traces of a system. They can be modeled as infinite word languages.

Definition 2.2 (Trace Property)
A trace property P is defined as a set of traces, i.e, $P \subseteq \Sigma^\omega$.

A set of traces $T \subseteq \Sigma^\omega$ satisfies a trace property P if $T \subseteq P$. This relation can be shifted to systems a S by defining $\mathsf{traces}(S)$ to be the set of infinite executions $T \subseteq \Sigma^\omega$ of S; S satisfies P if $\mathsf{traces}(S) \subseteq P$. Moreover we refer to the set of all trace properties by $\mathsf{Prop} = \mathcal{P}(\Sigma^\omega)$.

A classic example for a trace property is *progress* enforcing a system to always execute an action in the future. It can be formalized as follows:

$$\mathsf{Prog} = \{t \in \Sigma^\omega \mid \forall i \in \mathbb{N}. \exists j \in \mathbb{N}.\ j \geq i \wedge \mathsf{action}(t[j])\}$$

where action(s) identifies whether an action is executed in state s.

A subclass of trace properties which received special interest are the so-called *safety properties*. A violation of such a property is always required to be finitely detectable and irreversible.

Definition 2.3 (Safety Property)
A trace property $\mathsf{S} \in \mathsf{Prop}$ is called a *safety property* if the following holds:

$$\forall \sigma \in \Sigma^{\omega}.\ \sigma \not\models \mathsf{S} \Rightarrow [\exists w \leq \sigma . \forall \tau \in \Sigma^{\omega}.\ w \cdot \tau \not\models \mathsf{S}]$$

Such a prefix w is called a *bad-prefix* for S.

Many properties of general interest are safety properties. We elaborate on two examples:

- *Mutual exclusion* (ME) is a safety property defined with respect to concurrent systems $\mathcal{S}$ (systems with multiple agents and shared memory). Such a system $\mathcal{S}$ violates ME if two or more agents access the same memory at a time. A violation can be detected on a finite trace and can not be resolved in infinite time, i.e., ME is a safety language and can be formalized as follows:

  $$\mathsf{ME} = \{t \in \Sigma^{\omega} \mid \forall n \in \mathbb{N}. \exists i \in \mathbb{N}.\ \mathsf{accShare}(t[n], i) \Rightarrow \forall j \in \mathbb{N}.\ (i \neq j \Rightarrow \neg \mathsf{accShare}(t[n], j))\}$$

  where $\mathsf{accShare}(s, i)$ denotes process i accessing shared memory in state s.

- *No-Read-after-Write* (NRAW) requires a system that wrote data to some shared memory not to read from shared memory afterward. NRAW is a safety property as every time a system writes and then reads this has to happen within a finite time period and afterward the trace can not be extended in a way to satisfy NRAW again. It can be formalized as follows:

  $$\mathsf{NRAW} = \{t \in \Sigma^{\omega} \mid \forall n \in \mathbb{N}. \forall i \in \mathbb{N}.\ \mathsf{write}(t[n], i) \Rightarrow \forall m \geq n.\ \neg \mathsf{read}(t[m], i)\}$$

  where $\mathsf{write}_(s, i)$ denotes agent i writing in state s and $\mathsf{read}(s, i)$ denotes agent i reading in state s.

Given an ω-regular safety language $\mathcal{L}$, we denote a nondeterministic Büchi automaton $\mathcal{S} = (Q, \Sigma, q_0, F, \Delta)$ that accepts $\mathcal{L}$ with $Q = F$ a *safety* automaton. In the following chapters, we will omit the set of accepting states F in the definition of safety automata.

We denote the set of all bad-prefixes for a safety property S by $\mathsf{BadPref}(\mathbf{S})$. For a safety automaton $\mathcal{S}$, there exists a dual finite automaton $\mathcal{A}$ accepting all bad-prefixes of the safety language $\mathcal{L}(\mathcal{S})$ ($\mathsf{BadPref}(\mathcal{L}(\mathcal{S}))$). It this one can be constructed in time $2^{\Theta(|\mathcal{S}|)}$ [24]. We denote the set of all bad-prefixes for a language $\mathcal{L}$ by $\mathsf{BadPref}(\mathcal{L})$ and say $X \subseteq \mathsf{BadPref}(\mathcal{L})$ is a *trap* if for every $w \notin \mathcal{L}$ there exists a prefix of w in X and denote the set of all traps by $\mathsf{Trap}(\mathcal{L})$. A bad-prefix automaton $\mathcal{A}$ is called *tight* if $\mathcal{L}(\mathcal{A}) \equiv \mathsf{BadPref}(\mathcal{L})$ and *fine* if there exists some $X \in \mathsf{Trap}(\mathcal{L})$ and $\mathcal{L}(\mathcal{A}) \equiv X$. Intuitively, a tight and a fine bad-prefix automata can be distinguished as

follows: A tight one accepts every bad-prefix whereas a fine one only accepts finite extensions of bad-prefixes.

2.3 Hyperproperties

Hyperproperties extend trace properties by relating different executions and are defined as sets of trace properties.

Definition 2.4 [10] (Hyperproperties)
A hyperproperty $\mathbf{H} \subseteq \mathsf{Prop}$ is a set of trace properties, i.e., a set of sets of traces.

A set of traces $T \subseteq \Sigma^\omega$ satisfies a hyperproperty $\mathbf{H}$ if $T \in \mathbf{H}$; we denote this by $T \models \mathbf{H}$. Note that for every trace property there exists a hyperproperty satisfied by the same sets of traces.

Example: A classic example of k-safety hyperproperties is observational determinism introduced in Chapter 1. It can be defined as follows [10]:

$$\{T \subseteq \Sigma^\omega \mid \forall t, t' \in T.\ t[0] \equiv_{\mathsf{low},\ \mathsf{in}} t'[0] \Rightarrow t \equiv_{\mathsf{low},\ \mathsf{out}} t'\}$$

where $\equiv_{\mathsf{low},\ \mathsf{in}}$ and $\equiv_{\mathsf{low},\ \mathsf{out}}$ denote that the two traces in relation are equal with respect to their low input or output variables.

The notion of safety can be shifted to hyperproperties is the following way:

Definition 2.5 [10] (Hypersafety)
A hyperproperty $\mathbf{S}$ is called a *hypersafety property* if

$$\forall T \in \mathcal{P}(\Sigma^\omega).\ (T \notin \mathbf{S} \Rightarrow \exists T' \in \mathcal{P}(\Sigma^*).\ (T \geq T' \wedge \forall \widetilde{T} \in \mathcal{P}(\Sigma^\omega).\ (\widetilde{T} \geq T' \Rightarrow \widetilde{T} \notin \mathbf{S})))$$

Hypersafety captures bad interaction between traces, i.e., if $T \not\models \mathbf{S}$, then some prefix T' of finite traces must exists which can be used as a witness for the violation and no infinite extension $\widetilde{T} \geq T'$ can satisfy the hypersafety property.

Similarly to safety properties, *bad-prefixes* can be defined for hyperproperties, i.e., a bad-prefix now is a finite set of finite traces responsible for the violation.

Definition 2.6 [10] (Bad-Prefix)
For a safety hyperproperty $\mathbf{S}$, we denote by $\mathsf{BadPref}(\mathbf{S})$ the set of bad-prefixes of $\mathbf{S}$:

$$\mathsf{BadPref}(\mathbf{S}) := \{T \subseteq \mathcal{P}(\Sigma^*) \mid \forall T' \subseteq \mathcal{P}(\Sigma^\omega).\ (T' \geq T \Rightarrow T' \notin \mathbf{S})\}$$

By $\mathsf{BadPref}_k$ we denote the bad-prefixes consisting of at most k traces:

$$\mathsf{BadPref}_k(\mathbf{S}) := \{T \in \mathsf{BadPref}(\mathbf{S}) \mid |T| \leq k\}$$

If there exists some $k \in \mathbb{N}$ such that for every set of traces T violating $\mathbf{S}$ there exists a bad-prefix of T in $\mathsf{BadPref}_k(\mathbf{S})$, then we call $\mathbf{S}$ a *k-safety hyperproperty* [10].

Example: The example formalized above, observational determinism, is a 2-safety hyperproperty because every set of traces violating OD must contain two traces t and t' whose low inputs variables are equal in the first position but they differ with respect to their low output variables in some position.

A basic property of bad-prefixes for trace properties is *extension-closedness*, i.e., if $t \in \Sigma^*$ is a bad-prefix, then every extension t' of t is a bad-prefix as well. This notion can be applied to bad-prefixes of hypersafety properties as well, if $T \subseteq \Sigma^*$ is a bad-prefix, then every extension $T' \geq T$ is a bad-prefix. Further hypersafety properties are *subset-closed*:

Lemma 2.7 [10] (Subset-Closedness)
Every hypersafety property $\mathbf{S}$ is *subset-closed*. That means:

$$\forall T, T' \in \mathcal{P}(\Sigma^\omega).\ T \subseteq T' \Rightarrow T' \models \mathbf{S} \Rightarrow T \models \mathbf{S}$$

Therefore, every hypersafety property $\mathbf{S}$, besides the trivial property false, must be satisfied by the empty set, i.e., $\emptyset \models \mathbf{S}$.

2.4 HyperLTL

HyperLTL is a common temporal logic for expressing hyperproperties [11]. The syntax of HyperLTL is given by the following grammar:

$$\begin{aligned} \psi &::= \exists \pi.\ \psi \mid \forall \pi.\ \psi \mid \varphi \\ \varphi &::= a_\pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi\ \mathcal{U}\ \varphi \end{aligned}$$

where $\mathcal{V} = \{\pi_1, \pi_2, \dots\}$ is an infinite supply of trace variables, $a \in \mathsf{AP}$ is an atomic proposition and $\pi \in \mathcal{V}$ is a trace variable. Note that atomic propositions are indexed by trace variables. The

quantification over traces makes it possible to express properties like "on all traces ψ must hold", which is expressed by $\forall\pi.\ \psi$. Moreover one can express that "there exists a trace such that ψ is satisfied", which is denoted by $\exists\pi.\ \psi$. The derived operators $\Diamond$, $\Box$, and $\mathcal{R}$ are defined as for LTL. We abbreviate the formula $\bigwedge_{x\in X}(x_\pi \leftrightarrow x_{\pi'})$, expressing that the traces π and π' are equal with respect to a set $X \subseteq AP$ of atomic propositions in the first position, by $\pi \equiv_X \pi'$. Furthermore a trace variable π is called *free* in a HyperLTL formula if there is no quantification over π and a HyperLTL formula φ is called *closed* if there exists no free trace variable in φ.

A HyperLTL formula φ defines a *hyperproperty* in such a way that a set T of traces satisfies φ if it satisfies the hyperproperty described. Formally, the semantics of HyperLTL formulas is given with respect to a set of traces T and a *trace assignment* $\Pi : \mathcal{V} \to T$, i.e., a partial function mapping trace variables to actual traces. $\Pi[\pi \mapsto t]$ denotes that π is mapped to t and every other trace mapped according to Π. $\Pi[i, \infty]$ is a shorthand the trace assignment that is equal to $\Pi(\pi)[i, \infty]$ for all π.

$\Pi \models_T \exists\pi.\psi$	iff	there exists $t \in T\ :\ \Pi[\pi \mapsto t] \models_T \psi$
$\Pi \models_T \forall\pi.\psi$	iff	for all $t \in T\ :\ \Pi[\pi \mapsto t] \models_T \psi$
$\Pi \models_T a_\pi$	iff	$a \in \Pi(\pi)[0]$
$\Pi \models_T \neg\psi$	iff	$\Pi \not\models_T \psi$
$\Pi \models_T \psi_1 \vee \psi_2$	iff	$\Pi \models_T \psi_1$ or $\Pi \models_T \psi_2$
$\Pi \models_T \bigcirc\psi$	iff	$\Pi[1, \infty] \models_T \psi$
$\Pi \models_T \psi_1\, \mathcal{U}\, \psi_2$	iff	there exists $i \geq 0 : \Pi[i, \infty] \models_T \psi_2$
		and for all $0 \leq j < i : \Pi[j, \infty] \models_T \psi_1$

We say that a set of traces T *satisfies* a HyperLTL formula φ if $\emptyset \models_T \varphi$, where $\emptyset$ is the empty trace assignment. The language of φ is denoted by $\mathcal{L}(\varphi)$. W.l.o.g., we assume an enumeration of the trace quantifiers when using HyperLTL formulas, i.e., $\forall\pi_1 \ldots \forall\pi_k$.

Example: Having defined HyperLTL, we are able to formalize OD in HyperLTL [11]:

$$\forall\pi.\forall\pi'.\pi \equiv_{\mathsf{low,\ in}} \pi' \rightarrow \Box(\pi \equiv_{\mathsf{low,\ out}} \pi')$$

HyperLTL-SAT is the problem of deciding whether there exists a non-empty set of traces T such that $\emptyset \models_T \varphi$. Following a result by Finkbeiner and Hahn HyperLTL-SAT is undecidable, in general:

Theorem 2.8 [15]
Let φ be a HyperLTL formula of the form $\forall^*\exists^*.\ \psi$, where ψ is an LTL formula. HyperLTL-SAT of φ is undecidable.

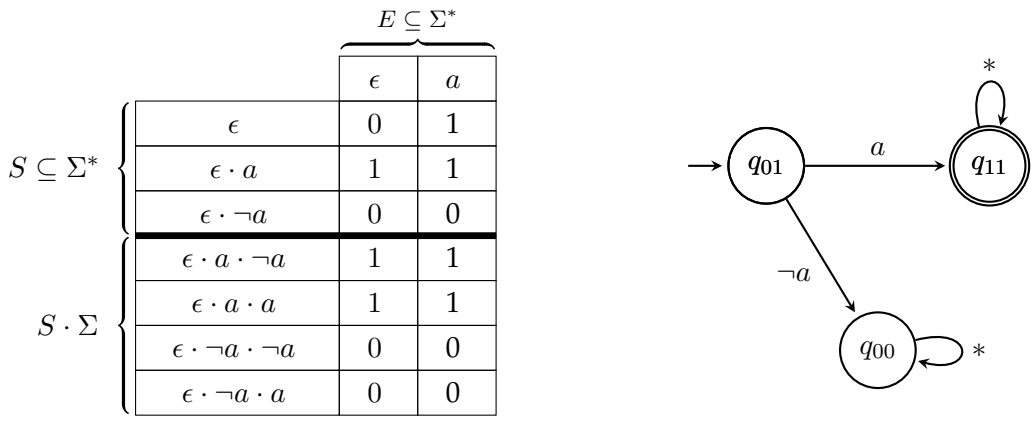

Figure 2.1: On the left, a closed and consistent observation table over the alphabet $\Sigma = \{a, \neg a\}$ with $S = \{\epsilon, a, \neg a\}$ and $E = \{\epsilon, a\}$. On the right, the corresponding automaton

2.5 Dana Angluin's L* Algorithm

Dana Angluin's L* algorithm learns a minimal deterministic finite automaton for an unknown regular language $\mathcal{L}$ [1]. Its running time is polynomially bounded in the size of the output. Thus it is output-sensitive. The algorithm itself is based on different rounds of communication between two components, a teacher and a learner. The learner poses questions to the teacher, who knows the target language $\mathcal{L}$ and answers these questions according to $\mathcal{L}$. Communication is restricted to two types of queries, membership queries and equivalence queries. In order to store the information about $\mathcal{L}$ that is gathered by the learner, he makes use of an observation table $\mathcal{O}$ defined as follows:

Definition 2.9 (Observation Table)
An observation table $O = (S, E, \Delta)$ is defined as follows:

- $S \subseteq \Sigma^*$ a prefix-closed set of finite sequences,
- $E \subseteq \Sigma^*$ a suffix-closed set of finite sequences,
- $\Delta : (S \cup S \cdot \Sigma) \times E \to \{0, 1\}$ the acceptance relation.

An example of such an observation table $\mathcal{O}$ over the alphabet $\Sigma = \{a, \neg a\}$ is given in Figure 2.1 on the left. Observation tables are usually depicted as a table with rows in $S \cup S \cdot \Sigma$, columns in E, and the entries denoted by Δ. An observation table is interpreted as follows: For a sequence $s \in (S \cup S \cdot \Sigma)$ and $e \in E$, the mapping $\Delta(s, e) = 1$ implies that the sequence $s \cdot e$ is a member of the target language. To determine the entries of Δ, the teacher poses *membership queries* to the teacher, they consist of a finite sequence s and the teacher returns whether $s \in \mathcal{L}$. For $s \in S \cdot \Sigma$ we

[1] We will not go into details about the algorithm and why it is correct but rather recap all definitions and lemma we later need for L^*_{Hyper}. For a detailed explanation of L*, we refer the reader to Angluin's original paper **??**

denote by $\mathsf{row}(s)$ a finite function mapping E to $\{0,1\}$ defined by $\mathsf{row}(s)(e) = \Delta(s \cdot e)$. Thus two sequences $s, t \in S$ agree on row ($\mathsf{row}(s) = \mathsf{row}(t)$) if and only if they have the same continuation language with respect to E: $F_{\mathcal{L}}(\mathsf{row}(s)) \cap E \equiv F_{\mathcal{L}}(\mathsf{row}(t)) \cap E$. To be able to obtain a deterministic automaton from an observation table $\mathcal{O} = (S, E, \Delta)$ it must be closed and consistent:

- $\mathcal{O}$ is called *closed* if for all $t \in S \cdot \Sigma$ there exists $s \in S$ such that $\mathsf{row}(t) = \mathsf{row}(s)$. Given a non-closed observation table, adding a $t \in S \cdot \Sigma$ that is pairwise different from each element in S to S resolves this conflict.
- $\mathcal{O}$ is called *consistent* if for all $t, t' \in S$ that agree in row ($\mathsf{row}(t) = \mathsf{row}(t')$) it follows for all $e \in \Sigma$ that $\mathsf{row}(t \cdot e) = \mathsf{row}(t' \cdot e)$. Given an inconsistent observation table, note that adding $t \cdot e$ or $t' \cdot e$ resolves the conflict caused by t and t'.

For a closed and consistent observation table $\mathcal{O}$ we can construct an DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ that is consistent with the observation table $\mathcal{O}$, especially Δ as follows:

$$
\begin{aligned}
Q &= \{\mathsf{row}(s) \mid s \in S\} \\
q_0 &= \mathsf{row}(\epsilon) \\
F &= \{\mathsf{row}(s) \mid s \in S \text{ and } \Delta(s) = 1\} \\
\delta(\mathsf{row}(s), a) &= \mathsf{row}(s \cdot a)
\end{aligned}
$$

Closedness guarantees that every transition is defined, i.e., for each state $q \in Q$ and label $a \in \Sigma$ $\delta(q, a) \in Q$. Consistency guarantees that $\mathcal{A}$ is deterministic.

Dana Angluin showed the following theorem regarding the size of $\mathsf{row}(S)$ and a minimal automaton $\mathcal{A}$ consistent with $\mathcal{O}$.

Lemma 2.10 [3]
Let $\mathcal{O} = (S, E, \Delta)$ be an observation table. Let n denote the number of different values of $\mathsf{row}(s)$ for s in S. Any automaton consistent with $\mathcal{O}$ must have at least n states.

After we defined the automaton-interpretation of $\mathcal{O}$, we can intuitively explain the definition of S and E: S is denoted the set of *accessing sequences* since for each state $q \in Q$ there exists a sequence $s \in S$ such that a run on $\mathcal{A}$ leads to q, i.e, every state $q \in Q$ can be *accessed* via a sequence $s \in S$. E is called the set of *separating sequences*, since given two accessing sequences $s, t \in S$ such that s and t run on $\mathcal{A}$ lead to different states q, q'. Then there exists some *separating* sequence $e \in E$ such that $\Delta(s, e) \neq \Delta(t, e)$.

Given a closed and consistent observation table the learner cannot gather any further information himself observing the state of $\mathcal{O}$. Thus an equivalence query is posed to the teacher. Such a query consists of a conjectured automaton $\mathcal{A}$, consistent with the current observation table $\mathcal{O}$. Afterward, the teacher either accepts and terminates or answers the query with a counterexample—embodied by a mistakenly accepted/rejected trace—to why the provided automaton does not represent the target language $\mathcal{L}$, this counterexamples is added to the set of accessing sequences and the algorithm repeats.

The learner algorithm is given in Algorithm 2.1. It was shown that for learning a regular language the algorithm terminates and outputs a minimal deterministic finite automaton accepting the unknown language.

Theorem 2.11 [3]
Given any minimally adequate teacher presenting an unknown regular language $\mathcal{L}$ the learner L^* eventually terminates and outputs a DFA $\mathcal{A}$ isomorphic to the minimum DFA $\mathcal{A}'$ accepting $\mathcal{L}$. Moreover, if n is the number of states of the minimum DFA accepting $\mathcal{L}$ and m is an upper bound on the length of any counterexample provided by the Teacher, then the total running time of L^* is bounded by a polynomial in m and n.

Algorithm 2.1 L^*_{Hyper}

Input: Observation table $\mathcal{O} = (S, E, \Delta)$, teacher $\mathcal{T}$, alphabet Σ
Output: Minimal DFA $\mathcal{A}$ satisfying the teacher's EQ query.

while true **do**
  $\mathcal{O} = \textsc{ClosedAndConsistent}(\mathcal{O}, \mathcal{T}, \Sigma)$
  construct conjecture automaton $\mathcal{A}$
  $w = EQ(\mathcal{A})$
  if w **then**
    add w and all prefixes $w' \leq w$ to S and extend Δ to $(S \cup S \cdot \Sigma) \cdot E$
  else
    return $\mathcal{A}$

Algorithm 2.2 ClosedAndConsistent

Input: Observation table $\mathcal{O} = (S, E, \Delta)$, teacher $\mathcal{T}$, alphabet Σ
Output: Closed and consistent observation table $\mathcal{O}$.

while (S, E, Δ) is not closed or not consistent **do**
  if (S, E, Δ) is not consistent **then**
    find s_1 and $s_2 \in S$, $a \in \Sigma$ and $e \in E$
    such that $\mathsf{row}(s_1) = \mathsf{row}(s_2)$ and $\Delta(s_1 \cdot a \cdot e) \neq \Delta(s_2 \cdot a \cdot e)$,
    and add $a \cdot e$ to E and extend $\mathcal{T}$ to $(S \cup S \cdot \Sigma) \cdot E$
  if (S, E, Δ) is not closed **then**
    find $s_1 \in S$ and $a \in \Sigma$
    such that $\mathsf{row}(s_1 \cdot a) \neq \mathsf{row}(s)$ for all $s \in S$,
    and add $s_1 \cdot a$ to S and extend Δ to $(S \cup S \cdot \Sigma) \cdot E$
return $\mathcal{O}$

Chapter 3

Automata for Hyperproperties

In this chapter we give a canonical representation for regular k-safety hyperproperties. We introduce k-safety and k-bad-prefix automata as a representation of regular k-safety hyperproperties and present algorithms for constructing these automata.

3.1 Regular k-safety Hyperproperties

To represent hyperproperties by ω-word automata, we first model sets of traces by single traces over the self-composition of the alphabet. For now, let $T = \{t_1, \dots, t_k\} \subseteq \Sigma^n$ for some $k, n \in \mathbb{N}$ be a set of finite traces of equal length and let $v = v_1 \dots v_n \in (\Sigma^k)^n$. We intend to construct a trace $v \in (\Sigma^k)^n$ for a given T such that v contains all the information stored in T, i.e., at every position $i < n$, we have $v_i[j] = t_j[i]$ for all $1 \leq j \leq k$. Given such a trace $v = v_1 \dots v_n \in (\Sigma^k)^n$, we define a mapping $\mathsf{unzip} : (\Sigma^k)^* \to \mathcal{P}(\Sigma^*)$ from traces over Σ^k to sets of traces with

$$\mathsf{unzip}(v) = \{t \in \Sigma^n \mid \exists\, 1 \leq j \leq k \text{ and } \forall 0 \leq i \leq n-1.\ t[i] = v_i[j]\}$$

The definition of unzip can be extended to ω-traces in a straight forward manner.

Definition 3.1 (k-Representation)
For some $k \in \mathbb{N}$, let $\sigma \in (\Sigma^k)^* \cup (\Sigma^k)^\omega$ and $T \subseteq \Sigma^* \cup \Sigma^\omega$. We call σ a *k-representation* of T if $\mathsf{unzip}(\sigma) \equiv T$ and we denote k the *arity* of this representation.

The notion of a k-representation is lifted to sets of sets of traces as follows: Let $S \subseteq (\Sigma^k)^* \cup (\Sigma^k)^\omega$ be a set of k-representations and let $\mathcal{T} \subseteq \mathcal{P}(\Sigma^*) \cup \mathcal{P}(\Sigma^\omega)$ be a set of sets of traces. S is called a k-representation of $\mathcal{T}$ if for every $T \in \mathcal{T}$: T has a k-representation in S and for all $s \in S$: $\mathsf{unzip}(s) \in \mathcal{T}$. Note that k-representations of sets of traces or sets of sets of traces are not unique. For the purpose of a better understanding we introduce some notions with respect to representations. First ZIP is the set of all representations:

$$\mathsf{ZIP}(T) = \left\{v \in (\Sigma^k)^n \mid T \equiv \mathsf{unzip}(v)\right\}$$

We use $\mathsf{zip}(T)$ as a shorthand for an arbitrary element in $\mathsf{ZIP}(T)$. Note that unzip denotes the inverse of zip, one can easily verify that for any T: $\mathsf{unzip}(\mathsf{zip}(T)) \equiv T$. However, for $v \in \Sigma^*$, $\mathsf{zip}(\mathsf{unzip}(t))$ does not equal v in general. Since zip might chose an order of the traces that is different from the initial order.

For the purpose of later chapters, where it becomes crucial to compare different representations. Let $v = v_1 \dots v_n \in (\Sigma^{k'})^n$ and let $\varsigma : \{1, \dots, k\} \to \{1, \dots, k'\}$ be a function: Define $v^\varsigma = (v_1[\varsigma(1)], \dots, v_1[\varsigma(k)]) \dots (v_n[\varsigma(1)], \dots, v_n[\varsigma(k)])$, the rotation of v under ς. Further, we extend the definition of zip with respect to some function ς as follows: For a set of traces $T = \{t_1, \dots, t_k\} \subseteq \Sigma^n$ and a function $\varsigma : \{1, \dots, l\} \to \{1, \dots, k\}$ for some $l, k \in \mathbb{N}$, we denote the zipped version of T under the mapping ς by $\mathsf{zip}_\varsigma(T) = t_{\varsigma,1} \dots t_{\varsigma,n} \in (\Sigma^l)^n$ where $v_{\varsigma,i} = (t_{\varsigma(1)}[i], \dots, t_{\varsigma(l)}[i])$ for all $1 \leq i \leq n$. The last approach can also be extended to automata over an alphabet Σ^k: Let $\mathcal{A} = (Q, \Sigma^k, q_0, \Delta, F)$ be a finite automaton and $\varsigma : \{1, \dots, k\} \to \{1, \dots, k'\}$ we define $\mathcal{A}^\varsigma = (Q, \Sigma^{k'}, q_0, \Delta_\varsigma, F)$ with $\Delta_\varsigma(q, a) = \{q' \in Q \mid q' \in \Delta(q, a^\varsigma)\}$. $\mathcal{A}^\varsigma$ accepts a trace v if and only if $\mathcal{A}$ accepts a trace w and $w^\varsigma = v$. Hence the language is rotated according to ς.

Note that, for $k' > k$ and for every k-representation v of some T there exists a (not necessarily unique) k'-representation v' of T that can be constructed by repeating the last label in every position of v for $k' - k$ times; this construction is denoted by $\mathsf{extend}_{k'}$:

Definition 3.2 (Extending k-representations)
Let $k' > k \in \mathbb{N}$ and $v = v_0 \dots v_n \in (\Sigma^k)^*$. We define the *extension* from k to k' traces by

$$\mathsf{extend}_{k'}(v) = v'_0 \dots v'_n \in (\Sigma^{k'})^*$$

such that for all $0 \leq i \leq n$ and $1 \leq j \leq k$: $v'_i[j] := v_i[j]$ and for $k < j \leq k'$: $v'_i[j] := v_i[k]$.

One can easily verify that for $k > 0$ and $v \in (\Sigma^k)^*$: $\mathsf{unzip}(v) = \mathsf{unzip}(\mathsf{extend}_{k'}(v))$.

In general, if a sequence σ is a k-representation of a set of traces T, any permutation of σ is a k-representation of T. Here we define a permutation of σ by exchanging the position in every k-tuple in every position in the trace according to some permutation $\varsigma \in S_k$, where S_k denotes the symmetric group in k elements. For example, the trace $v = (\{a\}, \{\})(\{\}, \{a\})$ is a 2-representation of $T = \{\{a\}\{\}, \{\}\{a\}\}$. Thus $v' = (\{\}, \{a\})(\{a\}, \{\})$ is a 2-representation of T, as well.

We characterize a k-safety hyperproperty $\mathbf{S}$ according to the representations of its set of bad-prefixes of size at most k. We denote the set of all k-representations of $\mathsf{BadPref}_k(\mathbf{S})$ by $\mathsf{BadPref}_{\vec{k}}(\mathbf{S})$. We can now define regular k-safety hyperproperties:

Definition 3.3 (Regular k-Safety Hyperproperties)
Let $\mathbf{S}$ be a k-safety hyperproperty. If $\mathsf{BadPref}_{\vec{k}}(\mathbf{S})$ is a regular language, we call $\mathbf{S}$ a *regular k-safety hyperproperty*.

The example discussed in Chapter 2—observational determinism—turns out to be regular k-safety.

As a final note on k-representations and hyperproperties, let us compare k-representations and trace assignments: Given a k-representation $v = v_0 \dots v_n \in (\Sigma^k)^{n+1}$ we can construct a trace assignment Π as follows $\Pi[\pi_i] = v_0[i] \dots v_n[i]$ for $i \in \{1, \dots, k\}$. Similarly, given a trace assignment Π defined on the trace variables $\pi_1 \dots, \pi_k$ with $\Pi[\pi_i] = t_i \in \Sigma^{n+1}$, we can construct $\mathsf{zip}(\Pi) = (v_1[0], \dots, v_k[0]) \dots (v_1[n], \dots, v_k[n])$. Therefore we can construct an isomorphism between k-representations and trace assignments on k trace variables. In the remainder of this thesis we use $v \models_\emptyset \psi(\pi_1, \dots, \pi_k)$ as a shorthand for $\Pi \models_\emptyset \psi(\pi_1, \dots, \pi_k)$, where Π is the trace assignment isomorphic to v.

3.2 Constructing Automata for Regular k-Safety Hyperproperties

For a regular k-safety hyperproperties $\mathbf{S}$, we can construct a finite-word automaton that recognizes $\mathbf{S}$ by accepting k-representations of every bad-prefix containing at most k traces. We extend the definition of a trap to the notion of hyperproperties in the straight-forward manner: For a safety hyperproperty $\mathbf{H}$, a suffix-closed set of sets $\mathbf{X} \subseteq \mathsf{BadPref}(\mathbf{H})$ is called a trap ($\mathbf{X} \in \mathsf{Trap}(\mathbf{H})$, note: $\mathbf{X}$ is not necessarily subset-closed) if for all $T \subseteq \mathsf{Prop}$ with $T \notin \mathbf{H}$ there exists a subset of finite prefixes $T' \in \mathbf{X}$ of T, i.e., $T' \leq T$. $\mathsf{Trap}_k(\mathbf{H})$ denotes the set of all traps of size at most k and $\mathsf{Trap}_{\vec{k}}(\mathbf{H})$ denotes the set of all k-representations of $\mathsf{Trap}_k(\mathbf{H})$. Formally, a k-bad-prefix automaton is defined as follows:

Definition 3.4 (k-Bad-Prefix Automata (k-BPA))
Let $\mathbf{S}$ be a regular k-safety hyperproperty. A finite bad-prefix automaton $\mathcal{A}$ over the alphabet Σ^k is called a k-bad-prefix automaton for $\mathbf{S}$ if

- $\exists \mathbf{X} \in \mathsf{Trap}_k(\mathbf{S}). \forall T \in \mathbf{X}. \exists v \in \mathcal{L}(\mathcal{A}).\ \mathsf{unzip}(v) = T$
- $\forall v \in \mathcal{L}(\mathcal{A}).\ \mathsf{unzip}(v) \in \mathsf{BadPref}(\mathbf{S})$

Note that a k-bad-prefix automaton is not required to accept all k-representations in $\mathsf{BadPref}_{\vec{k}}(\mathbf{S})$. We denote the dual automaton of a k-bad-prefix automaton a k-safety automaton. It is defined according to the semantics of a k-bad-prefix automaton: A safety automaton $\mathcal{S}$ over the alphabet Σ^k that accepts all k-representations of every set of size at most k satisfying $\mathbf{S}$ and in case a set of traces does not satisfy $\mathbf{S}$, then one k-representation of a subset of it is not accepted. Formally, this means $\mathcal{S}$ has to satisfy:

- $\forall T \notin \mathbf{S}. \exists v \in (\Sigma^k)^\omega.\ \mathsf{unzip}(v) \subseteq T \wedge v \notin \mathcal{L}(\mathcal{S})$
- $\forall T \in \mathbf{S}. \forall v \in (\Sigma^k)^\omega.\ \mathsf{unzip}(v) \subseteq T \Rightarrow v \in \mathcal{L}(\mathcal{S})$

Note that k-bad-prefix automata are allowed to accept too little representations of bad-prefixes. Thus, the dual k-safety automata may accept to many representations of set of traces even some that do not satisfy the regular safety property.

Figure 3.1: A permutation-complete 2-bad-prefix automaton (left) and a non-permutation-complete 2-bad-prefix automaton (right) for $\forall\pi.\forall\pi'.\ \square(a_\pi \rightarrow a_{\pi'})$

Following the tradition of bad-prefix automata [24], we distinguish between two types of k-bad-prefix automata:

- *Tight* bad-prefix automata, which accept at least one k-representation of every element in $\mathsf{BadPref}_k(\mathbf{S})$.
- *Fine* bad-prefix automata, which accept at least one k-representation of every element in some trap $\mathbf{X} \in \mathsf{Trap}_k(\mathbf{S})$.

As it was pointed out before, we can provide different k-representations for $\mathsf{BadPref}(\mathbf{S})$ and in turn provide different k-bad-prefix automata that each recognize $\mathsf{BadPref}(\mathbf{S})$ depending on the chosen representation. Thus in order to discriminate the automata according to their respective accepted representations we call $\mathcal{A}$ *permutation-complete* if $\mathcal{L}(\mathcal{A}) \in \mathsf{Trap}_{\vec{k}}(\mathbf{S})$. In order to better understand k-bad-prefix automata, let us consider the following example before further investigating their construction:

Example: Consider the 2-safety hyperproperty given by the following HyperLTL formula:

$$\varphi = \forall\pi.\forall\pi'.\ \square(a_\pi \rightarrow a_{\pi'})$$

The set of bad-prefixes of φ is given by $\mathsf{BadPref}(\varphi)$. B, B', and $\mathsf{BadPref}_{\vec{2}}(\varphi)$ are 2-representations of $\mathsf{BadPref}(\varphi)$. Figure 3.1 depicts two 2-BPAs for φ, the one on the left accepts $\mathsf{BadPref}_{\vec{2}}(\mathbf{S})$ and the one on the right accepts B'.

$$\begin{aligned}
\mathsf{BadPref}(\varphi) &= \{T \subseteq \Sigma^* \mid \exists t, t' \in T.\ \exists i.\ t[i] \models a \wedge t'[i] \not\models a\} \\
\mathsf{B} &= \{\sigma = (\alpha_0, \alpha_0') \dots (\alpha_m, \alpha_m') \in (\Sigma^2)^* \mid \exists i.\ a \in \alpha_i \wedge a \notin \alpha_i'\} \\
\mathsf{B}' &= \{\sigma = (\alpha_0, \alpha_0') \dots (\alpha_m, \alpha_m') \in (\Sigma^2)^* \mid \exists i.\ a \in \alpha_i' \wedge a \notin \alpha_i\} \\
\mathsf{BadPref}_{\vec{2}}(\varphi) &= \{\sigma = (\alpha_0, \alpha_0') \dots (\alpha_m, \alpha_m') \in (\Sigma^2)^* \mid \exists i.\ a \in \alpha_i \leftrightarrow a \notin \alpha_i'\}
\end{aligned}$$

For the remainder of this thesis, the *length* of a bad-prefix T is defined as the length of its longest element and the *size* of a bad-prefix T is defined as the number of traces in T.

When considering the detection of violations of a system against some specifications, it is, in some cases, of high priority to react as early as possible. Thus tight automata for safety

properties play an important role in the context of monitoring algorithms. In the context of hyperproperties, an equivalent notion are *tight* and *permutation-complete* k-bad-prefix automata. With tight automata violations can be detected quickly at the earliest instance. A permutation-complete automaton is resistant to the ordering in which traces are observed and is able to detect a violation no matter in what representation the traces of a system are fed into the automaton.

Before we dive into further details regarding the applications of k-bad-prefix automata, we treat the construction of such k-bad-prefix automata in the next two theorems. We show how to build a deterministic, permutation-complete and tight k-bad-prefix automaton for a k-safety hyperproperty given as a nondeterministic k-safety automaton or a HyperLTL formula in the $\forall$-fragment. Vardi and Kupfermann showed that tight bad-prefix automata for safety-properties can be built starting from a nondeterministic safety automaton or an LTL formula [24]. The size of the automaton is exponential in the size of the nondeterministic safety automaton and doubly-exponential in the size of the LTL formula. Their results can be adopted to HyperLTL while restricting to the following fragment of HyperLTL:

Definition 3.5 (Safe HyperLTL)
A HyperLTL formula φ is called safe if φ is of the form $\varphi = \forall \pi_1 \dots \forall \pi_n.\ \psi$ and ψ is a safe with respect to the semantics of LTL.

It is easy to verify that a safe HyperLTL formula always expresses a k-safety hyperproperty. Restricting to safe HyperLTL formulas, we obtain the following lemma regarding the construction of k-BPAs:

Lemma 3.6
Given a safe HyperLTL formula $\varphi = \forall \pi_1 \dots \forall \pi_k.\ \psi$ we can construct a nondeterministic k-safety automaton for φ of size $2^{O(|\psi|)}$.

Proof According to Kupferman and Vardi's construction, we can construct a nondeterministic safety automaton $\mathcal{S}_\psi$ such that $\mathcal{L}(\mathcal{S}_\psi) = L(\psi)$ with $|\mathcal{S}_\psi| = 2^{O(|\psi|)}$ [24].

Given a nondeterministic safety automaton $\mathcal{S}_\psi = (Q, \Sigma_\Pi, \delta_\psi, q_0)$, where Σ_Π is the set Σ indexed by trace propositions $\pi_1, \dots, \pi_n$, proceed as follows: Construct a safety automaton $\mathcal{S}$ that resolves the trace quantification by assigning each quantifier one component in every position of a trace. Let $\mathcal{S} = (Q, \Sigma^k, \delta, q_0)$ with the transition relation δ formally defined as follows: $\delta(q, (a_1, \dots, a_k)) = Q'$ if and only if $\delta_\psi(q, a_{1,\pi_1} \wedge \dots \wedge a_{k,\pi_k}) = Q'$, where a_{π_i} is equal to the atomic propositions described by a, but indexed by the trace proposition π_i.

In order to achieve correctness, we have to show that $\mathcal{S}$ is a k-safety automaton for φ. According to Kupferman and Vardi's construction, $\mathcal{S}$ is a safety automaton on the alphabet Σ^k. Next we make use of the following lemma given by Clarkson et al. regarding their construction of Büchi

automata, which can be easily adapted to our construction above [11]:

$$\Pi \models_\emptyset \psi \text{ if and only if } \mathsf{zip}(\Pi) \in \mathcal{L}(\mathcal{S}) \tag{3.1}$$

We show that $\mathcal{S}$ satisfies the two requirements of a k-safety automaton for φ:

- $\forall T \notin \mathcal{L}(\varphi).\exists v \in (\Sigma^k)^\omega.\ \mathsf{unzip}(v) \subseteq T \wedge v \notin \mathcal{L}(\mathcal{S})$
  Follows immediately from Equation (3.1) using subset-closedness of hypersafety properties.
- $\forall T \in \mathcal{L}(\varphi).\forall v \in (\Sigma^k)^\omega.\ \mathsf{unzip}(v) \subseteq T \Rightarrow v \in \mathcal{L}(\mathcal{S})$:
  Assume that there exists some $T \in \mathcal{L}(\varphi)$ such that for some $v \in (\Sigma^k)^\omega$ with $\mathsf{unzip}(v) \subseteq T$ it holds that $v \notin \mathcal{L}(\mathcal{S})$. Then there exists a trace assignment Π of T such that $\mathsf{unzip}(\Pi) \equiv \mathsf{unzip}(v)$ and $\mathsf{zip}(\Pi) \notin \mathcal{L}(\mathcal{S})$. Thus according to (3.1), $\Pi \not\models_\emptyset \psi$ and since φ is a safe formula, we obtain $T \notin \mathcal{L}(\varphi)$. This contradicts our assumption.

■

Lemma 3.7
Let **S** be a regular k-safety hyperproperty given by a nondeterministic k-safety automaton of size n, or a safe HyperLTL formula of size m. We can construct a deterministic k-bad-prefix automaton for **S** of size $2^{\Theta(n)}$ or $2^{2^{O(m)}}$, respectively.

Proof

- Nondeterministic k-safety automaton $\mathcal{S}$:
  Note that a k-safety automaton $\mathcal{S}$ is a safety automaton. Thus, we can apply the result by Kupferman and Vardi to transform $\mathcal{S}$ into a dual determinstic bad-prefix automaton $\mathcal{A}$ of size $2^{\Theta(n)}$ [24]. They make use of a subset-construction and hence implicitly determinize and complement the automaton at once. It remains to show that $\mathcal{A}$ is a k-bad-prefix automaton for **S**.
  - $\exists \mathbf{X} \in \mathsf{Trap}_k(\mathbf{S}).\forall T \in \mathbf{X}.\exists v \in \mathcal{L}(\mathcal{A}).\ \mathsf{unzip}(v) = T$:
    Note that $\mathcal{L}(\mathcal{S})$ is a safety property thus

    $$\forall \sigma \in \Sigma^\omega.\ [\sigma \notin \mathcal{L}(\mathcal{S}) \Rightarrow [\exists w \leq \sigma.\forall \tau \in \Sigma^\omega.\ w \cdot \tau \not\models \mathsf{S}]]$$

    and

    $$\mathcal{L}(\mathcal{A}) = \mathsf{Pref}(\mathcal{L}(\mathcal{S}))$$

    where $\mathsf{Pref}(\mathcal{L}(\mathcal{S}))$ contains exactly all prefixes $x \in (\Sigma^k)^*$ such that for all $y \in (\Sigma^k)^\omega$ we have $x \cdot y \notin \mathcal{L}(\mathcal{S})$. We show that $\mathsf{unzip}(\mathsf{Pref}(\mathbf{S}))$ is a member of $\mathsf{Trap}_k(\mathcal{L}(\mathcal{S}))$ and thus the language of $\mathcal{A}$ satisfies the above condition.

    Remind yourself of the definition of $\mathsf{Trap}(\mathbf{S})$: We have to show that for all $T \subseteq \mathsf{Prop}$ with $T \notin \mathbf{S}$ there exists $T' \in \mathsf{unzip}(\mathsf{Pref}(\mathbf{S}))$ such that $T' \leq T$.

Let $\mathbf{T} \subseteq \mathsf{Prop}$ be chosen arbitrarily with $T \notin \mathbf{H}$. Then there exists some $v \in (\Sigma^k)^\omega \backslash \mathcal{L}(\mathcal{S})$ with $\mathsf{unzip}(v) \subseteq T$ following the definition of k-safety automata. Since $\mathcal{L}(\mathcal{S})$ is a safety language, there also exists some $v' \in (\Sigma^k)^*$ with $v' \leq v$ such that no infinite trace extending v' is a member of $\mathcal{L}(\mathcal{S})$. Hence, $\mathsf{unzip}(v') \in \mathsf{BadPref}_k(\mathbf{S})$ (definition of k-safety automata) and because T was chosen arbitrarily, it follows that $\mathsf{unzip}(\mathsf{Pref}(\mathcal{L}(\mathcal{S}))) \in \mathsf{Trap}_k(\mathbf{S})$.

– $\forall v \in \mathcal{L}(\mathcal{A}).\ \mathsf{unzip}(v) \in \mathsf{BadPref}(\mathbf{S})$:
Prove by contraposition: Assume there exists a $v \in \mathcal{L}(\mathcal{A})$ with $\mathsf{unzip}(v) \notin \mathsf{BadPref}(\mathbf{S})$. Then, for all $\sigma \geq v$ it follows that $\sigma \notin \mathcal{L}(\mathcal{S})$ since $\mathcal{A}$ is the dual bad-prefix automaton of $\mathcal{S}$, which implies that $\mathsf{unzip}(\sigma) \notin \mathbf{S}$. Thus, $\mathsf{unzip}(v)$ is a bad-prefix of $\mathbf{S}$, which contradicts our assumption.

- LTL formula φ:
Lemma 3.6 and the above result.

■

The next step is the construction of a permutation-complete deterministic k-bad-prefix automata for k-safety hyperproperties. Given a non-deterministic k-safety automaton $\mathcal{S}$, the straightforward approach involves taking the intersection $\mathcal{S}$ with respect to all functions $\varsigma : \{1, \dots, k\} \to \{1, \dots, k\}$. This approach yields a doubly exponential blow-up in $|\mathcal{S}|$.

Thus, we apply a more involved construction: First, we construct the dual k-bad-prefix automaton $\mathcal{A}$ of $\mathcal{S}$ and then we compose the different versions of $\mathcal{A}$ with respect to all ς without yielding a nondeterministic automaton. This composition can be obtained since the nondeterministic choice is only in the initial state and all automata are bad-prefix automata, i.e., as soon as they reach an accepting state they never leave it.

Lemma 3.8
Given a nondeterministic k-safety automaton of size n for a regular k-safety hyperproperty $\mathbf{S}$, we can construct a tight, permutation-complete, deterministic k-bad-prefix automaton for $\mathbf{S}$ of size $2^{2^{O(k \log(k))}}$ and $2^{\Theta(n)}$.

Proof Let $\mathcal{S}$ be a nondeterministic k-safety automaton for $\mathbf{S}$ and let $\varsigma_1, \dots, \varsigma_{k^k} : \{1, \dots, k\} \to \{1, \dots, k\}$ be pairwise different functions. According to Lemma 3.7, we can construct a dual deterministic k-bad-prefix automaton $\mathcal{A} = (Q, \Sigma^k, q_0, \delta, F)$ for $\mathcal{S}$ of size $2^{|\mathcal{S}|}$.

Note that $\mathcal{A}$ is not necessarily a permutation-complete automaton but for all k-representations t of a bad-prefix, there exists an i such that $t_{\varsigma_i} \in \mathcal{L}(\mathcal{S})$. Thus in order to achieve permutation-completeness, we can construct the union of all k-bad-prefix automata $\mathcal{A}_\varsigma$. Using the following construction this yields a deterministic k-bad-prefix automaton for $\mathbf{S}$. W.l.o.g. we assume that $\mathcal{A}$ is complete and define $\mathcal{A}_\otimes = (Q_\otimes, \Sigma^k, q_{\otimes,0}, \delta_\otimes, F_\otimes)$, with:

- $Q_\otimes = \bigtimes_{i \in \{1,\dots,k^k\}} Q$

- $q_{\otimes,0} = (q_0, \dots, q_0)$
- $\delta_\otimes((q_1, \dots, q_{k^k}), a) = (q'_1, \dots, q'_{k^k})$, where $q_i \xrightarrow{\sigma_i(a)} q'_i$ for all i
- $F_\otimes = \{(q_1, \dots, q_{k^k}) \in Q^{k^k} \mid \exists i.\ q_i \in F)\}$

First, it is easy to verify that $A_\otimes$ is a deterministic automaton. Further, the size of $\mathcal{A}_\otimes$ is in $2^{2^{O(k \log k)}}$ ($|Q_\otimes| = |Q|^{k^k} = |Q|^{2^{k \log(k)}}$) and linear in $|Q|$, hence in $2^{O(|\mathcal{S}|)}$. To obtain correctness of our construction we now have to show that $v \in \mathcal{L}(\mathcal{A}_\otimes)$ if and only if $\mathsf{unzip}(v) \in \mathsf{BadPref}(\mathbf{S})$.

($\Rightarrow$) Let $v \in \mathcal{L}(\mathcal{A}_\otimes)$, i.e., there exists some $\varsigma : \{1, \dots, k^k\} \to \{1, \dots, k^k\}$ such that $v_\varsigma \in \mathcal{L}(\mathcal{A})$. Hence, no infinite sequence extending $v_\varsigma \in \mathcal{L}(\mathcal{S})$ and v_ς represents a bad-prefix for $\mathbf{S}$. Using superset-closedness of bad-prefixes it follows that $\mathsf{unzip}(v) \in \mathsf{BadPref}(\mathbf{S})$.

($\Leftarrow$) Let $v \in (\Sigma^k)^*$ and let $\mathsf{unzip}(v) \in \mathsf{BadPref}(\mathbf{S})$. By definition of k-safety automata, there exists $v' \in (\Sigma^k)^*$ such that $\mathsf{unzip}(v') \subseteq \mathsf{unzip}(v)$, i.e., $v' = v_\varsigma$ for some ς, and all infinite extensions of v' are not in the language of $\mathcal{S}$. Thus $v' \in \mathcal{L}(\mathcal{A})$ and since $t_\varsigma = v'$ it follows that there exists some $\varsigma : \{1, \dots, k^k\} \to \{1, \dots, k^k\}$ such that $v_\varsigma \in \mathcal{L}(\mathcal{A})$. Finally, following the above construction: $v \in \mathcal{L}(\mathcal{A}_\otimes)$.

The deterministic automaton constructed from Lemma 3.7 is tight for ψ but it is not necessarily tight for $\mathsf{BadPref}_{\vec{k}}(\varphi)$. However, since the $\mathcal{A}_\otimes$ is deterministic, permutation-complete, and fine for φ, it can be turned tight without increasing its state space: Since $\mathcal{A}$ is fine for φ, every trace representing a set of traces violating φ has an accepted prefix and all their suffixes are again accepted. Further, $\mathcal{A}$ is deterministic. Thus if a word has a non-accepting run it is not accepted. Hence a state q in $\mathcal{A}$ represents bad-prefixes, i.e., is supposed to be accepted if and only if there exist no reachable non-accepting loop from q. We can adjust $\mathcal{A}_\otimes$ such that it becomes tight for φ: Add every state from which no non-accepting loop can be reached to the set of accepting states. This involves a nested-DFS, looking for non-accepting loops, for every state and does not increase the size of $\mathcal{A}$.

■

Proposition 3.9
Let $\varphi = \forall \pi_1 \dots \forall \pi_k.\ \psi$ be a k-safety HyperLTL formula, we can construct a deterministic permutation-complete k-bad-prefix automaton for φ of size $2^{2^{\mathcal{O}(k \log(k))}}$ and $2^{2^{O(|\psi|)}}$.

Proof We can construct a nondeterministic k-safety automaton $\mathcal{S}$ for φ of size in $2^{O(|\psi|)}$ using the construction in Lemma 3.6. Afterward, by applying Lemma 3.8 to $\mathcal{S}$, the claimed bound follows.

■

3.3 Equivalence of k-Bad-Prefix Automata

From Lemma 2.7, we infer that the language of bad-prefixes for a safety hyperproperty is superset-closed. Thus every k-safety hyperproperty is also a k'-safety hyperproperty for all $k' \geq k$, which implies that **S** can be represented by k-bad-prefix automata of different arities k'. We next provide an algorithm to decide equivalence of k-bad-prefix automata with different respective arity.

Given a k-bad-prefix automaton $\mathcal{A}_k$ for a k-safety hyperproperty **S**, we lift the definition of unzip to k-bad-prefix automata as follows: $\mathsf{unzip}(\mathcal{A}_k) = \{T \mid T = \mathsf{unzip}(\sigma) \text{ where } \sigma \in \mathcal{L}(\mathcal{A}_k)\}$.

Definition 3.10 (Equivalence of k-Bad-Prefix Automata)
Given two k-bad-prefix automaton $\mathcal{A}_k$ and a k'-bad-prefix automaton $\mathcal{A}_{k'}$ over alphabets Σ^k and $\Sigma^{k'}$, where $k \leq k'$. We say that $\mathcal{A}_k$ and $\mathcal{A}_{k'}$ are equivalent, denoted by $\mathcal{A}_k \equiv \mathcal{A}_{k'}$ if and only if:

1. $\forall T \in \mathsf{unzip}(\mathcal{A}_k).\forall \widetilde{T} \subseteq \Sigma^\omega.\ T \leq \widetilde{T} \Rightarrow (\exists T' \in \mathsf{unzip}(\mathcal{A}_{k'}).\ T' \leq \widetilde{T})$
2. $\forall T' \in \mathsf{unzip}(\mathcal{A}_{k'}).\forall \widetilde{T} \subseteq \Sigma^\omega.\ T' \leq \widetilde{T} \Rightarrow (\exists T \in \mathsf{unzip}(\mathcal{A}_k).\ T \leq \widetilde{T})$

Intuitively, $\mathcal{A}_k$ and $\mathcal{A}_{k'}$ are equivalent if for every set of traces T which extends a k-representation accepted by $\mathcal{A}_k$, some k'-representation of a prefix of T is accepted by $\mathcal{A}_{k'}$. And for set of traces T that extends a bad-prefix represented in $\mathcal{L}(\mathcal{A}_{k'})$ a prefix of a subset of T is represented in $\mathcal{L}(\mathcal{A}_k)$. Note that this definition is independent of tight and fine as well as permutation-complete and non-permutation-complete k-bad-prefix automata.

An algorithm for checking equivalence of k-bad-prefix automata is presented in the next theorem.

Theorem 3.11
Let $\mathcal{A}_k$ be a deterministic k-bad-prefix automata and let $\mathcal{A}_{k'}$ be a deterministic k'-bad-prefix automaton with $k \leq k'$. Checking whether $\mathcal{A}_k \equiv \mathcal{A}_{k'}$ can be done in time polynomially bounded in $|\mathcal{A}_k|$,$|\mathcal{A}_{k'}|$ and in space exponentially bounded in k'.

Proof Let $\mathcal{A}_k = (Q_k, q_{0,k}, \Sigma^k, \delta_k, F_k)$ and $\mathcal{A}_{k'} = (Q_{k'}, q_{0,k'}, \Sigma^{k'} \delta_{k'}, F_{k'})$ be k-bad-prefix automata. Let $\mathcal{S}_k = (Q_k^{\mathcal{S}}, q_{0,k}^{\mathcal{S}}, \Sigma^k, \delta_k^{\mathcal{S}})$ and $\mathcal{S}_{k'} = (Q_{k'}^{\mathcal{S}}, q_{0,k'}^{\mathcal{S}}, \Sigma^{k'}, \delta_{k'}^{\mathcal{S}})$ be the corresponding dual k-safety automata of $\mathcal{A}_k$ and $\mathcal{A}_{k'}$, respectively.

1. We look for a counterexample to Definition 3.10 (1), i.e., a representation $v \in \mathcal{L}(\mathcal{A}_k)$ such that there exists a $T \subseteq \Sigma^\omega$ with $\mathsf{unzip}(v) \leq T$, for which no k-representation is accepted by $\mathcal{A}_{k'}$. Note that $\mathcal{A}_{k'}$ is not necessarily permutation-complete. Thus in $\mathcal{A}_{k'}$'s dual safety automaton $\mathcal{S}_{k'}$ a word that is accepted might be a k'-representation of a set of traces that is not a member of the hypersafety property $\mathsf{unzip}(\mathcal{S}_{k'})$. Because some other k'-representation of it is accepted. Therefore, finding a counterexample is equal to finding some $v' \in (\Sigma^{k'})^\omega$ for which every k'-representation is accepted by $\mathcal{A}_{k'}$ and some k'-representation of a prefix of $\mathsf{unzip}(v')$ is a k'-bad-prefix according to the language of $\mathcal{A}_k$.

Keeping this in mind, we construct the following safety automaton $\mathcal{S}^{\varsigma}_{k'} = (Q^{\varsigma}_{k'}, q^{\varsigma}_{0,k'}, \Sigma^{k'}, \delta^{\varsigma}_{k'})$ with:

- $Q^{\varsigma}_{k'} = \{q^{\varsigma}_{0,k'}\} \cup Q^{\mathcal{S}}_{k',1} \times \cdots \times Q^{\mathcal{S}}_{k',k'^{k'}}$ where each $Q^{\mathcal{S}}_{k',i}$ is the i-indexed version of $Q^{\mathcal{S}}_{k'}$
- Let $\varsigma_1, \ldots, \varsigma_{k'^{k'}} : \{1, \ldots, k'\} \to \{1, \ldots, k'\}$ be pairwise different functions. The transition-relation $\delta^{\varsigma}_{k'}$ allows the following transitions:
  - $q^{\varsigma}_{0,k'} \xrightarrow{a} (q_1, \ldots, q_{k'^{k'}})$ if and only if $q^{\mathcal{S}}_{0,k'} \xrightarrow{a^{\varsigma}} q'$, where $a \in \Sigma^{k'}$ and q'_i is the i-indexed copy of q'
  - $(q_1, \ldots, q_{k'^{k'}}) \xrightarrow{a} (q'_1 \ldots, q'_{k'^{k'}})$ if and only if for all i: $q \xrightarrow{a^{\varsigma}} q'$, where $a \in \Sigma^{k'}$ and q_i, q'_i are the i-indexed copies of q, q'

The automaton $\mathcal{S}^{\varsigma}_{k'}$ accepts a sequence σ if and only if all representations of $\mathsf{unzip}(\sigma)$ and all subsets of $\mathsf{unzip}(\sigma)$ are accepted by $\mathcal{S}_{k'}$. According to the definition of subset-closedness, a sequence is accepted if it is not a k'-bad-prefix of the hypersafety property recognized by $\mathcal{S}_{k'}$.

We next expand the automaton $\mathcal{A}_k$ to an automaton $\tilde{\mathcal{A}}_k = (Q_k, q_{0,k}, \Sigma^{k'}, \delta'_k, F_k)$ over the alphabet $\Sigma^{k'}$ such that:

$$\delta'_k(q, (a_1, \ldots, a_k, \ldots, a'_k)) = q' \quad \Leftrightarrow \quad t_i = a_k \text{ for all } k < i \leq k' \text{ and } \delta_k(q, (a_1, \ldots, a_k)) = q'$$

We have $\mathsf{extend}_{k'}(\mathcal{L}(\mathcal{A}_k)) \equiv \mathcal{L}(\tilde{\mathcal{A}}_k)$ and it is easy to see that $\mathsf{unzip}(\mathcal{L}(\mathcal{A}_k)) \equiv \mathsf{unzip}(\mathcal{L}(A'_k))$.

Lastly, we build the product automaton of $\tilde{\mathcal{A}}_k$ and $\mathcal{S}^{\varsigma}_{k'}$. In case there is an infinite run in $\mathcal{S}^{\varsigma}_{k'}$ with a prefix accepted by $\tilde{\mathcal{A}}_k$, we found a counterexample and the two automata do not recognize the same k-safety hyperproperty.

The size of the product automaton is $|Q_k| \cdot |Q_{k'}|^{k'^{k'}}$. Hence finding an accepting lasso in the automaton can be done in time polynomial in $|\mathcal{A}_k|$ and $|\mathcal{A}_{k'}|$ and in space exponential in k'.

2. Next, we have to verify that for every set T for which a k'-representation v of a prefix of T is accepted by $\mathcal{A}_{k'}$ there exits a representation $' \in \mathcal{L}(\mathcal{A}_k)$ that detects this violation, i.e., $\mathsf{unzip}(v') \leq T$.

   Therefore, we construct an automaton $\mathcal{S}^{\varsigma}_{k'} = (Q^{\varsigma}, q^{\varsigma}_0, \Sigma^{k'}, \delta^{\varsigma})$ as follows:

   - $Q_{\varsigma} = \{q^{\varsigma}_0\} \cup Q^{\mathcal{S}}_{k,1} \times \cdots \times Q^{\mathcal{S}}_{k,k'^{k}}$, where each $Q^{\mathcal{S}}_{k,i}$ is a copy of $Q^{\mathcal{S}}_{k}$ and each state q is labeled with an additional index i.
   - Let $\varsigma_1, \ldots, \varsigma_{k'^{k}} : \{1, \ldots, k\} \to \{1, \ldots, k'\}$ be pairwise different functions. The function δ^{ς} allows the following transitions:

- $q_0^\varsigma \xrightarrow{a} (q_1, \dots, q_{k'k})$ iff $q_{0,k}^{\mathcal{S}} \xrightarrow{a^\varsigma} q$ where $a \in \Sigma^{k'}$ and q_i is the i indexed copy of q.
- $(q_1, \dots, q_{k'k}) \xrightarrow{a} (q'_1, \dots, q'_{k'k})$ iff for each i: $q \xrightarrow{a^\varsigma} q'$ where $a \in \Sigma^{k'}$ and q_i, q'_i are the i-indexed copies of q, q'

The automaton $\mathcal{S}_{k'}^\varsigma$ accepts a trace $\omega \in (\Sigma^{k'})^\omega$ if and only if all its projections to k-traces are accepted by $\mathcal{S}_k$.

We can now build a product automaton of $\mathcal{S}_{k'}^\varsigma$ and $\mathcal{A}_{k'}$ and check if the intersection is empty. The size of the product automaton is $|Q'_k| \cdot |Q_k|^{k'k}$. Finding a lasso in the accepting automaton can be done in time polynomial in $|\mathcal{A}_k|$, $|\mathcal{A}_{k'}|$, and k' and in space exponential in k.

■

Corollary 3.12
Checking the equivalence of two nondeterministic k-bad-prefix automata $\mathcal{A}_k$ and $\mathcal{A}_{k'}$ of arity k, k' for $k \leq k'$ can be done in time exponential in $|\mathcal{A}_k|$, $|\mathcal{A}_{k'}|$ and space exponential in k'.

Proof According to Lemma 3.7, we can construct deterministic k-bad-prefix automata for $\mathcal{A}_k$ and $\mathcal{A}_{k'}$ of size $2^{O(|\mathcal{A}_k|)}$ and $2^{O(|\mathcal{A}_{k'}|)}$, respectively. Afterward, we can check their equivalence in time polynomial in $2^{O(|\mathcal{A}_k|)}$ and $2^{O(|\mathcal{A}_{k'}|)}$ and space exponential in k according to Theorem 3.11.

■

Corollary 3.13
Checking whether two safe HyperLTL formulas φ_k and $\varphi_{k'}$ for $k \leq k'$ are equivalent can be done in space exponential in k' and time doubly exponential in $|\varphi_k|$ and $|\varphi_{k'}|$.

Proof According to Lemma 3.6, we can construct nondeterministic k-safety automata $\mathcal{A}_k$ for φ_k and $\mathcal{A}_{k'}$ for $\varphi_{k'}$ of size $2^{O(|\varphi_k|)}$ and $2^{O(|\varphi_{k'}|)}$, respectively. Afterward, we can check their equivalence in time exponential in $|\mathcal{A}_k|$ and $|\mathcal{A}_k|$ and space exponential in k according to Corollary 3.12.

■

3.4 Minimal k-Bad-Prefix Automata

According to the construction above, we can construct a permutation-complete and tight k-bad-prefix automata for a HyperLTL formula that is of size doubly-exponential in both the number of quantifiers and the size of the HyperLTL formula. For many hyperproperties, these automata can be minimized to substantially smaller ones. Two factors result in a blow-up in the size of the automaton: The first one being the automata construction used for HyperLTL uses the traditional techniques for constructing automata for LTL [19, 33]. These tend to construct automata that are

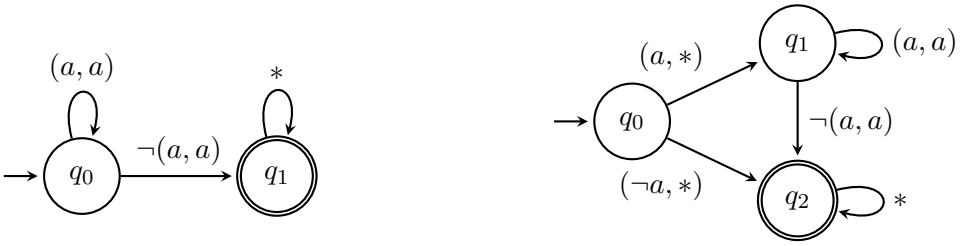

Figure 3.2: A minimal permutation-complete 2-BPA (left) and a minimal non-permutation-complete 2-BPA (right) for $\forall\pi\forall\pi'.\ a_\pi \wedge \square(a_\pi \rightarrow \bigcirc(a_{\pi'} \wedge a_\pi))$

not permutation-complete [17, 16]. The second factor being that every k-safety hyperproperty is k'-safety for $k' \geq k$ as well.

We start by discussing the first factor: Surprisingly the automata constructed in Lemma 3.7 can be extraordinarily larger than their permutation-complete counterparts. Consider, for example, the 2-hyperproperty $\forall\pi.\forall\pi'.\ a_\pi \wedge \square(a_\pi \rightarrow \bigcirc(a_{\pi'} \wedge a_\pi))$. Figure 3.2 shows a permutation-complete and a larger non-permutation-complete 2-BPA recognizing φ with respect to the regular languages that each automaton accepts they are of minimal size. Yet, the non-permutation-complete automaton requires a larger state space.

The following theorem shows that there are families of automata for which minimal non-permutation-complete automata can be arbitrarily larger than permutation-complete ones. Especially the construction for HyperLTL formulas introduced by Clarkson et al. in [11] can easily lead to such a blow-up as it uses the semantics of LTL on the self-composition of the system and therefore makes no direct requirements on whether a should hold at the first position of π'.

Lemma 3.14
There is a family of equivalent k-safety HyperLTL formulas φ_h with a deterministic permutation-complete k-BPA of constant size in $|\varphi_h|$ and a non-permutation-complete k-BPA for φ which is of minimal size regarding the accepted regular language is of size at least $2h$.

Proof Consider the family of HyperLTL formulas φ_h over $\mathsf{AP} = \{a\}$

$$\varphi_h := \forall\pi.\forall\pi'.\ \underbrace{a_\pi \wedge \square(a_\pi \rightarrow \bigcirc a_{\pi'}) \wedge \square(a_{\pi'} \rightarrow \bigcirc a_\pi) \wedge (\neg \bigcirc a_\pi) \rightarrow \bigcirc^{2h-1} a_\pi}_{\psi_h}$$

φ_h requires that a holds on every trace at every position—this is indeed a 1-safety property and equivalent to $\forall\pi.\ \square a_\pi$. The permutation-complete deterministic 1-BPA for φ_h of minimal size is depicted in Figure 3.3. It is easy to verify that its size does not depend on h.

Next we consider the following 2-representation of $\mathsf{BadPref}_2(\varphi_h)$:

$$\mathsf{BadPref}_{\vec{2}}(\varphi_h) = \{v \in (\mathcal{P}(\mathsf{AP})^2)^* \mid v_0[0] \not\models a \vee \exists i \in \mathbb{N}.\ ((v_i[0] \models a \wedge v_{i+1}[1] \not\models a) \vee (v_i[1] \models a \wedge v_{i+1}[0] \not\models a) \vee (v_1[0] \not\models a \wedge v_{2h-1} \not\models a))\}$$

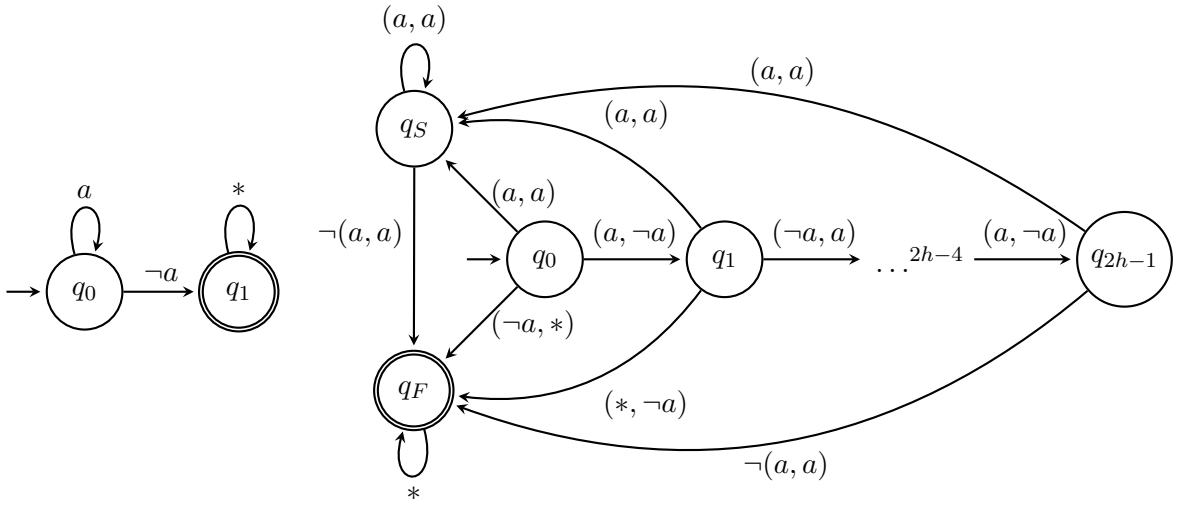

Figure 3.3: A permutation-complete and a non-permutation-complete 2-BPA recognizing the bad-prefixes of $\varphi_h = \forall\pi.\forall\pi'.\ a_\pi \wedge \square(a_\pi \rightarrow \bigcirc a_{\pi'}) \wedge \square(a_{\pi'} \rightarrow \bigcirc a_\pi) \wedge (\neg \bigcirc a_\pi) \rightarrow \bigcirc^{2h-1} a_\pi$.

This is the language induced by the LTL part ψ_h of φ_h. A 2-bad-prefix automaton of minimal size accepting $\mathsf{BadPref}_{\vec{2}}(\varphi_h)$ is depicted on the right-hand side of Figure 3.3. In order to prove the claimed lower bound regarding the size of a deterministic automaton accepting $\mathsf{BadPref}_{\vec{2}}(\varphi_h)$, we show that every DFA accepting $\mathsf{BadPref}_{\vec{2}}(\varphi_h)$ requires at least $2h$ states. Therefore, we use the existence of $2h$ many sequences in $(\mathcal{P}(\mathsf{AP})^2)^{2h}$ that each have pairwise different continuation languages with respect to $\mathsf{BadPref}_{\vec{2}}(\varphi_h)$.

Consider $\mathcal{S} = \{x \mid x < [(\{a\},\{\})(\{\},\{a\})]^h\}$ containing $2h$ sequences. First we distinguish the sequences of even and odd length. Let $x, y \in \mathcal{S}$ with $|x| \bmod 2 = 0, |y| \bmod 2 = 1, x \cdot (a,\neg a) \notin \mathsf{BadPref}_{\vec{2}}(\varphi_h)$ but $y \cdot (a,\neg a) \in \mathsf{BadPref}_{\vec{2}}(\varphi_h)$. Thus their continuation languages differ. We only consider the even sequences, for now, the odd ones can be analyzed in a similar manner.

Let $x, y \in \mathcal{S}$ and $|x| \bmod 2 \equiv |y| \bmod 2 \equiv 0$ and $x \neq y$. W.l.o.g., $|x| \leq |y|$. Thus after $2h$ steps a has to hold globally in both positions. Therefore x and y can be distinguished by the sequences of length at most $2h$. Consider the following continuation $z = (((\{a\},\{\})(\{\},\{a\}))^{h-|y|/2}\ y \cdot z \in \mathsf{BadPref}_{\vec{2}}(\varphi_h)$ and since x is shorter than y, it follows $x \cdot z \notin \mathsf{BadPref}_{\vec{2}}(\varphi_h)$. Consequently the continuation languages of x and y, $F_{\mathsf{BadPref}_{\vec{2}}(\varphi_h)}(x), F_{\mathsf{BadPref}_{\vec{2}}(\varphi_h)}(y)$ differ for all pairwise distinct $x, y \in \mathcal{S}$. Thus the minimal DFA accepting $\mathsf{BadPref}_{\vec{2}}(\varphi_h)$ has at least $2h$ different states according to Myhill-Nerode's theorem (Theorem 2.1).

■

One might argue that phenomena like Lemma 3.14 can happen if the non-permutation-complete language is chosen badly. However, the main point of this theorem is to emphasize the disadvantages of the standard algorithm for constructing automata for HyperLTL formulas (Lemma 3.7). Because this construction yields the above family of automata.

We now investigate the second factor for blow-ups, arising from k-safety hyperproperties being k'-safe for all $k' \geq k$. An automaton of arity k' has a potentially larger alphabet $\Sigma^{k'}$ and it

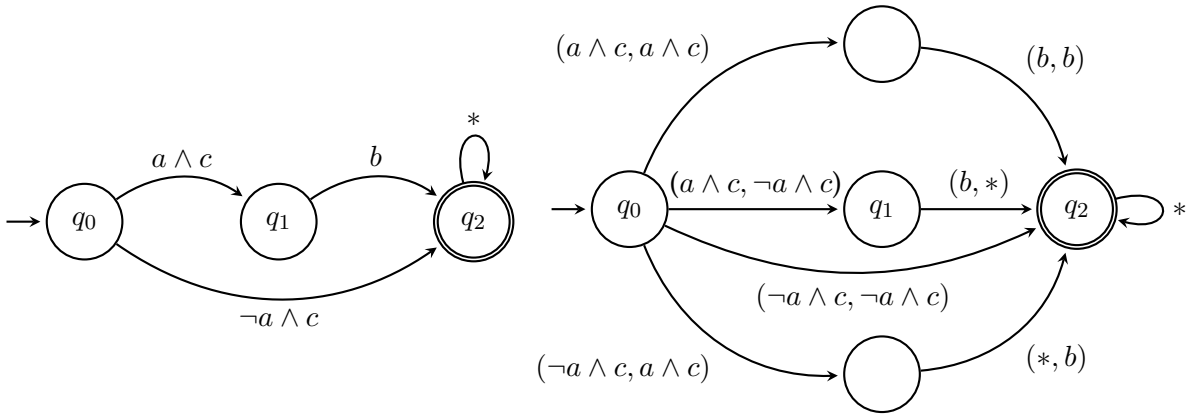

Figure 3.4: A permutation-complete 1-safety automaton (left) for $\varphi_1 = \forall\pi.\ (a_\pi \rightarrow \bigcirc b_\pi) \wedge c_\pi$ and a permutation-complete 2-safety automaton (right) for $\varphi_2 = \forall\pi.\forall\pi'.\ (a_\pi \rightarrow \bigcirc b_\pi) \wedge c_{\pi'}$, both of minimal size with respect to their accepted regular language.

will be shown later that the minimal permutation-complete k'-bad-prefix automaton is at least as large as the corresponding minimal k-bad-prefix automaton.

Consider, for example, the two equivalent hyperproperties $\varphi_1 = \forall\pi.\ (a_\pi \rightarrow \bigcirc b_\pi) \wedge c_\pi$ and $\varphi_2 = \forall\pi.\forall\pi'.\ (a_\pi \rightarrow \bigcirc b_\pi) \wedge c_{\pi'}$. The first one is obviously 1-safe whereas the second one appears to be 2-safe on first sight. Figure 3.4 shows a permutation-complete k-safety automaton for φ_1 and φ_2 of arity 1 and 2, respectively. The permutation-complete automaton with minimal size and arity 2 is exponentially larger, in the arity, than the minimal size, permutation-complete automaton of arity 1. We define the minimal size of k-BPAs as follows:

Definition 3.15 (Minimal Bad-prefix Automaton)
A k-bad-prefix automaton $\mathcal{A}$ is called *minimal* if k is minimal with respect to $\equiv$ and the state space of $\mathcal{A}$ is of minimal size.

Our last result in this section is the the construction of a minmial, tight, permutation-complete, deterministic k-bad-prefix automaton from a k-safety HyperLTL formula.

Proposition 3.16
Let $\varphi = \forall\pi_1 \ldots \forall\pi_k.\ \psi$ be a safe HyperLTL formula. We can construct a minmial, tight, permutation-complete, deterministic k-bad-prefix automaton for φ of size $2^{2^{\mathcal{O}(k \log(k))}}$ and $2^{2^{\mathcal{O}(|\psi|)}}$.

Proof Let $\mathcal{A}_k = (Q, \Sigma^k, q_0, F, \delta_k)$ be a deterministic permutation-complete k-bad-prefix automaton for φ constructed using Proposition 3.9. The size of $\mathcal{A}_k$ is in $2^{2^{\mathcal{O}(k \log(k))}}$ and $2^{2^{\mathcal{O}(|\psi|)}}$.

In order to obtain a minimal k-BPA, we proceed as follows: Let $\mathcal{A}_{k-1} = (Q, \Sigma^{k-1}, q_0, F, \delta_{k-1})$ be a $(k-1)$-bad-prefix automaton with the transition relation $\delta_{k-1}(q, a) := \delta_k(q, \mathsf{extend}_k(a))$. Observe that if $\mathcal{A}_k$ is deterministic and permutation-complete, then $\mathcal{A}_{k-1}$ is again deterministic

and permutation-complete. Being deterministic follows immediately from the transition relation and we prove permutation-completeness by contraposition:

Assume that $\mathcal{A}_{k-1}$ is not a permutation-complete $(k-1)$-bad-prefix automaton for φ but φ is a $(k-1)$-safety hyperproperty. W.l.o.g., there exists a sequence $v \in (\Sigma^{k-1})^*$ such that $\mathsf{unzip}(v) \in \mathsf{BadPref}(\varphi)$ and $v \notin \mathcal{L}(\mathcal{A}_{k-1})$. Then, by construction $\mathsf{extend}_k(v) \notin \mathcal{L}(\mathcal{A}_k)$ and it follows that $\mathcal{A}_k$ was not a permutation-complete k-bad-prefix automaton for φ. This is a Contradiction.

Further, $\mathcal{A}_{k-1}$ is a permutation-complete $(k-1)$-bad-prefix automaton for φ if and only if φ is a $(k-1)$-safety hyperproperty. Assume that $\mathcal{A}_{k-1}$ is not a $(k-1)$-bad-prefix automaton but φ is $(k-1)$-safety, then there is a representation v of $k-1$ traces which is not accepted by $\mathcal{A}_{k-1}$ but $v \in \mathsf{BadPref}_{\overrightarrow{k}}(\varphi)$. Hence, $\mathsf{extend}_k(t) \notin \mathcal{L}(\mathcal{A}_{k-1})$ and thus, $\mathcal{A}_k$ is not a permutation-complete k-bad-prefix automaton for φ. The opposite direction is obvious. At this point we can recursively construct $\mathcal{A}_{k-1}$ and use the equivalence algorithm presented in Theorem 3.11 to compare $\mathcal{A}_k$ with $\mathcal{A}_{k-1}$. As soon as $\mathcal{A}_{k-1}$ is not equivalent to $\mathcal{A}_k$, we determined the minimal $k \in \mathbb{N}$ such that φ is k-safe.

As soon as the minimal k is reached, we have a tight, permutation-complete, deterministic k-bad-prefix automaton $\mathcal{A}_k$ fine for φ. Finally a standard poly-time state space minimization algorithms for deterministic finite automata can be applied, for example, Hopcroft's algorithm [21]. The result is a minmial, tight, permutation-complete, deterministic bad-prefix automaton for φ.

The runtime complexity of all the intermediate steps is dominated by the initial construction of the deterministic permutation-complete k-bad-prefix automaton for φ and thus in time doubly-exponential in $|\varphi|$ and k.

■

The provided construction is expensive and requires an explicit construction of the automaton of maximal arity k to minimize it afterward. In order overcome this drawback, we will provide a learning-based construction in Chapter 4.

3.5 A Canonical Representation of Regular k-Safety Hyperproperties

In this section, we complete our search for a canonical representation of regular k-safety hyperproperties and prove that minmial, tight, permutation-complete, deterministic k-bad-prefix automata meet the requirements.

Lemma 3.17
Two hypersafety properties $\mathbf{S}$ and $\mathbf{S}'$ are equal if and only if $\mathsf{BadPref}(\mathbf{S}) \equiv \mathsf{BadPref}(\mathbf{S}')$.

Proof

(⇒) Let $T \in \mathsf{BadPref}(\mathbf{S})$. Thus for all $T' \geq T$ it follows $T' \not\models \mathbf{S}$ and therefore $T' \not\models \mathbf{S}'$, by assumption. Hence, $T \in \mathsf{BadPref}(\mathbf{S}')$. The same proof holds when exchanging $\mathbf{S}$ and $\mathbf{S}'$ yielding $\mathsf{BadPref}(\mathbf{S}) \equiv \mathsf{BadPref}(\mathbf{S}')$.

($\Leftarrow$) Let $T \notin \mathbf{S}$. Thus there exists some $T' \in \mathsf{BadPref}(\mathbf{S})$. According to our assumption it follows $T' \in \mathsf{BadPref}(\mathbf{S}')$ and thus $T \notin \mathbf{S}'$. The same proof holds when exchanging $\mathbf{S}$ and $\mathbf{S}'$ yielding $\mathbf{S} \equiv \mathbf{S}'$.

■

Theorem 3.18
Minimal, tight, permutation-complete, deterministic k-bad-prefix automata are a canonical representation for regular k-safety hyperproperties.

Proof Let $\mathbf{S}$ and $\mathbf{S}'$ be two regular k-safety hyperproperties. We show that they are equal if and only if their minmial, tight, permutation-complete, deterministic k-bad-prefix automata coincide.

($\Rightarrow$) Let $\mathbf{S} \equiv \mathbf{S}'$ and let k' be the minimal arity k such that $\mathbf{S}$ and $\mathbf{S}'$ are k-safe. Using Lemma 3.17, their bad-prefixes coincide and thus it follows that $\mathsf{BadPref}_{k'}(\mathbf{S}) \equiv \mathsf{BadPref}_{k'}(\mathbf{S}')$. Hence, a k'-bad-prefix automaton accepting $\mathsf{BadPref}_{k'}(\mathbf{S})$ is a k'-BPA for $\mathbf{S}'$ as well and the the claim follows.

($\Leftarrow$) Let $\mathcal{A}$ be a minmial, tight, permutation-complete, deterministic k-bad-prefix automaton for $\mathbf{S}$ and $\mathbf{S}'$. It follows that both hyperproperties have the same minimal k such that they are k-safety and $\mathsf{BadPref}_{\vec{k}}(\mathbf{S}) \equiv L(\mathcal{A}) \equiv \mathsf{BadPref}_{\vec{k}}(\mathbf{S}')$. Bad-prefixes are superset-closed and every set of traces violating $\mathbf{S}$ or $\mathbf{S}'$ must have a subset which is in the corresponding set of bad-prefixes of size at most k. Thus $\mathsf{BadPref}(\mathbf{S}) \equiv \mathsf{BadPref}(\mathbf{S}')$ and using Lemma 3.17 $\mathbf{S} \equiv \mathbf{S}'$ follows.

It remains to show that minmial, tight, permutation-complete, deterministic k-bad-prefix automaton are unique for a k-safety hyperproperty $\mathbf{S}$. Clearly, the minimal arity k is unique. The language of all bad-prefixes of size k is also unique for $\mathbf{S}$. Thus the set of all k-representations is unique and this language is regular by assumption. Further it is well-known that minimal deterministic automata are a unique representation for regular languages. Hence, the claimed uniqueness follows.

■

Chapter 4

Learning k-Safety Hyperproperties

In the foregoing chapter, we introduced an algorithm for constructing minimal permutation-complete k-BPA. This construction heavily suffers from the explicit construction of all intermediate automata. In order to handle this problem, we present a framework for learning minmial, tight, permutation-complete, deterministic k-BPAs for an unknown k-safety hyperproperty **S** in the following chapter. The algorithm extends Dana Angluin's L* for learning regular languages from queries and counterexamples. For the entire section, we denote the target k-safety property by **S**.

Like L*, our framework mainly consists of two components: a learner and a teacher. We establish an implementation of the learner and analyze its running time complexity. On the other hand, Chapter 5 deals with an implementation of the teacher. The two components, as well as their communication, are depicted in Figure 4.1.

In our setting the minimal arity k such that **S** is k-safety is initially unknown[1]. Thus, the learner starts by learning an automaton over the alphabet Σ, i.e., $k = 1$ and increases the arity whenever necessary. He can pose *membership queries*, which are defined as follows: Given a set $T = \{t_1, \ldots, t_k\} \subseteq \Sigma^n$ of finite traces, the teacher answers whether T is a bad-prefix of **S**.

Regarding the learner's observation table we stick to the definition introduced for L*, i.e., an

[1] For an easier understanding of the algorithm we stick to the following notion: The current arity of the learner is denoted k_L, the arity known to the teacher is denoted k_T, and the minimal arity we aim to achieve is denoted k_G.

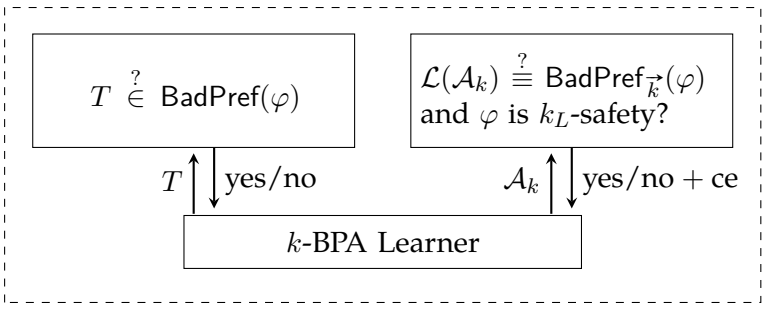

Figure 4.1: The interaction between learner and teacher in the modified L* for learning minimal k-BPA's

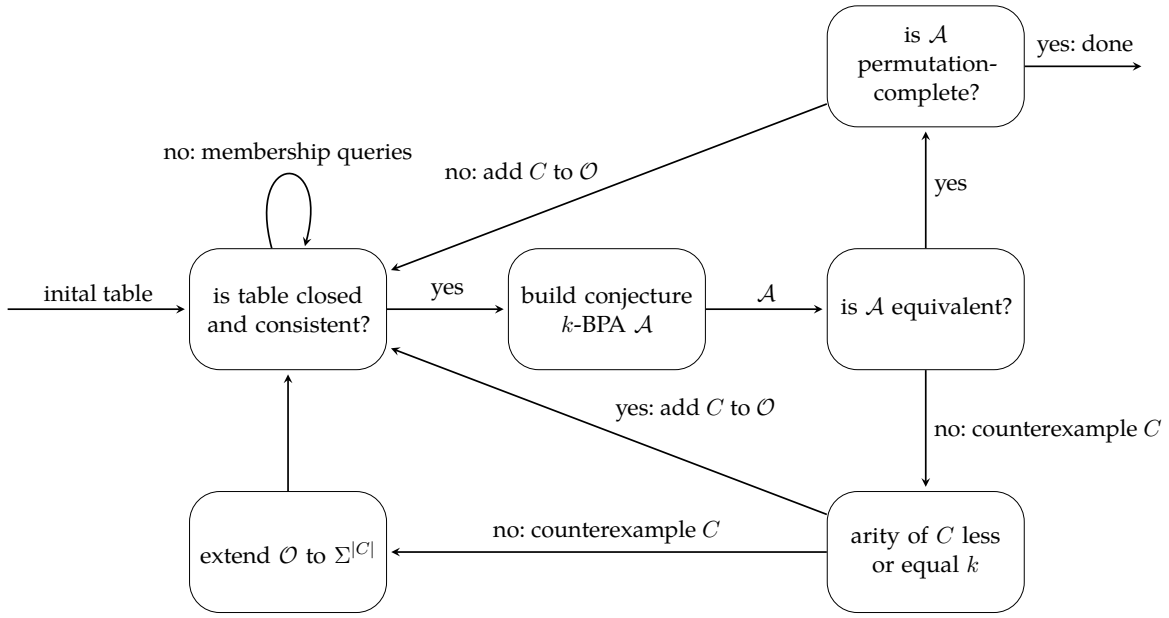

Figure 4.2: The learner's workflow in L^*_{Hyper}.

observation table $\mathcal{O} = (S, E, \Delta)$. $S \subseteq (\Sigma^k)^*$ is a non-empty finite prefix-closed set of *accessing* sequences, $E \subseteq (\Sigma^k)^*$ is a non-empty finite suffix-closed set of *separating* sequences, and $\Delta : (S \cup S \cdot \Sigma) \cdot E \to \{0, 1\}$ a mapping defined as $\Delta(s \cdot e) = 1$ if and only if $s \cdot e$ is a k-representation of a bad-prefix for $\mathbf{S}$ ($s \cdot e \in \mathsf{BadPref}_{\overrightarrow{k}}(\mathbf{S})$).

Provided a closed and consistent observation table $\mathcal{O}$, we can construct an automaton $\mathcal{A}$ over the alphabet Σ^k in time polynomial in $|\mathcal{O}|$, as described in Section 2.5. Afterward the constructed automaton can be used in *equivalence query* to the teacher. Such a query is answered by an indication of whether or not $\mathcal{A}_k$ is a k-BPA for $\mathbf{S}$. In the later case $\mathcal{A}_k$ a counterexample $T \subseteq \Sigma^n$ is provided. Such a counterexample T always fits into one of the following two categories:

- T is a bad-prefix of $\mathbf{S}$ and no k-representation is accepted by $\mathcal{A}_k$
- T is not a bad-prefix of $\mathbf{S}$, but some k-representation of T is accepted by $\mathcal{A}_k$

In order to guarantee that the outcome of our learning framework is minimal in k and in the size the teacher must be *minimally adequate* that is the counterexamples provided are of minimal length and every counterexample is of minimal size.

4.1 The Algorithm

The learner's workflow during the algorithm is illustrated in Figure 4.2 and an explicit algorithm is presented in Algorithm 4.1. Over the course of the next few pages, we give a detailed explanation of L^*_{Hyper}. Let $\mathbf{S}$ be a regular k-safety hyperproperty. Regarding the communication with a designated teacher $\mathcal{T}$, we denote membership queries corresponding to a set of traces T by $MQ_{\mathcal{T}}(T)$ and equivalence queries for an automaton $\mathcal{A}$ by $EQ_{\mathcal{T}}(\mathcal{A})$.

Initially, the learner assumes $\mathbf{S}$ to be a 1-safety hyperproperty and thus L^*_{Hyper} is called with

Algorithm 4.1 L^*_{Hyper}

Input: Observation table $\mathcal{O} = (S, E, \Delta)$, arity k, teacher $\mathcal{T}$, alphabet Σ
Output: Minimal k-BPA $\mathcal{A}$ satisfying the teacher's EQ query.

1: **while** True **do**
2: $\mathcal{O} = \textsc{ClosedAndConsistent}(\mathcal{O}, \mathcal{T}, \Sigma^k)$
3: construct conjecture automaton $\mathcal{A}$
4: $W = EQ(\mathcal{A})$
5: **if** $W \neq \emptyset$ **then**
6: **if** $|W| > k$ **then**
7: $\mathcal{O} := \textsc{Extend}(\mathcal{O}, \Sigma^k, k, |W|)$
8: **let** $\sigma \in (\Sigma^{|W|})^*$ such that $\mathsf{unzip}(\sigma) = W$
9: add σ and all prefixes $\sigma' \leq \sigma$ to S and extend Δ to $(S \cup S \cdot \Sigma^k) \cdot E$
10: **return** $\mathrm{L}^*_{Hyper}(\mathcal{O}, |W|, \mathcal{T}, \Sigma)$
11: **else**
12: **if** $MQ(W)$ **then**
13: **let** $\sigma \in (\Sigma^k)^*$ with $\mathsf{unzip}(\sigma) = W$
14: **else**
15: **let** $\sigma \in (\Sigma^k)^*$ with $\mathsf{unzip}(\sigma) = W$ and $\sigma \in \mathcal{L}(\mathcal{A})$
16: add σ and all prefixes $\sigma' \leq \sigma$ to S and extend Δ to $(S \cup S \cdot \Sigma^k) \cdot E$
17: **else**
18: $X = \textsc{IsComplete}(\mathcal{A}, k)$
19: **if** $X \neq \emptyset$ **then**
20: **let** $\sigma \in (\Sigma^k)^*$ such that $\mathsf{unzip}(\sigma) = W$ and $\sigma \notin \mathcal{L}(\mathcal{A})$
21: add σ and all prefixes $\sigma' \leq \sigma$ to S and extend Δ to $(S \cup S \cdot \Sigma^k) \cdot E$
22: **else**
23: **return** $\mathcal{A}$

input arity $k = 1$, alphabet Σ, and observation table $\mathcal{O} = (\{\epsilon\}, \{\epsilon\}, \{((\epsilon, \epsilon), MQ(\epsilon))\})$ containing only information about the empty word. The algorithm proceeds as follows: In every iteration of the main loop the observation table $\mathcal{O}$ is transformed until it becomes closed and consistent (Algorithm 4.2, line 2). As soon as a closed and consistent observation table is established, a deterministic automaton $\mathcal{A}_L$, consistent with $\mathcal{O}$, is constructed, as it was the case in the L^* algorithm (Algorithm 4.2, line 3).

Afterward, an equivalence query involving $\mathcal{A}_L$ is posed to the teacher $\mathcal{T}$. In case this query produces a counterexample $W \subseteq \Sigma^n$ for some $n \in \mathbb{N}$, the following two cases must be distinguished:

1. (line $6 - 10$) W contains more than k_L traces: Then W is not expressible in k_L traces according to the definition of a minimally adequate teacher. Hence the learner has to increase the arity to $|W|$ in order to express $\mathsf{BadPref}(\mathbf{S})$. At this point it is important to keep the information gathered so far. Therefore the learner shifts his current information from arity k_L to $|W|$ by calling Algorithm 4.4. It extends every element $x \in S \cup E$ to a trace in $\Sigma^{|W|}$, by repeating the last position in every trace. Proceeding this way, the represented sets of traces remain the same and the information gathered so far is preserved. Afterward

Algorithm 4.2 ClosedAndConsistent

Input: Observation table $\mathcal{O} = (S, E, \Delta)$, teacher $\mathcal{T}$, alphabet Σ^k
Output: Closed and consistent observation table $\mathcal{O}$.

1: **while** (S, E, Δ) is not closed or not consistent **do**
2: **if** (S, E, Δ) is not consistent **then**
3: find s_1 and $s_2 \in S$, $a \in \Sigma^k$ and $e \in E$ such that $\mathsf{row}(s_1) = \mathsf{row}(s_2)$ and $\Delta(s_1 \cdot a \cdot e) \neq \Delta(s_2 \cdot a \cdot e)$, and add $a \cdot e$ to E and extend $\mathcal{T}$ to $(S \cup S \cdot \Sigma^k) \cdot E$
4: **if** (S, E, Δ) is not closed **then**
5: find $s_1 \in S$ and $a \in \Sigma^k$ such that $\mathsf{row}(s_1 \cdot a) \neq \mathsf{row}(s)$ for all $s \in S$, and add $s_1 \cdot a$ to S and extend Δ to $(S \cup S \cdot \Sigma^k) \cdot E$
6: **return** $\mathcal{O}$

Algorithm 4.3 Extend

Input: Observation table $\mathcal{O} = (S, E, \Delta)$, alphabet Σ^k, integers k, k' with $k' > k$
Output: An observation table over the alphabet $\Sigma^{k'}$ based on $\mathcal{O}$

1: $\mathcal{O}' = (S', E', \Delta') = (\{\epsilon\}, \{\epsilon\}, \{((\epsilon, \epsilon), \Delta(\epsilon, \epsilon))\})$
2: **for** $s \in S$ **do**
3: add $\mathsf{extend}_{k'}(s)$ to S'
4: **for** $e \in E$ **do**
5: add $\mathsf{extend}_{k'}(e)$ to E'
6: **for** $s \in S, e \in E$ **do**
7: $\Delta'(\mathsf{extend}_{k'}(s), \mathsf{extend}_{k'}(e)) := \Delta(s, e)$
8: extend Δ' to $(S \cup S \cdot \Sigma^{k'}) \cdot E$
9: **return** $\mathcal{O}'$

the given counterexample W must be added to the observation table. It suffices to add an arbitrary representation of W to the set of accessing sequences, since no representation of a subset of W is accepted.

2. (line $11 - 16$) W contains less than or equal k traces: Then, there exists a set of traces W for which at least one representation is not treated as intended in $\mathcal{A}_L$. If $W \in \mathsf{BadPref}(\mathbf{S})$, then by definition of the equivalence queries no representation of W is accepted. Thus adding any representation of W resolves the issue. Otherwise, at least one representation is accepted by accident. Thus, it is necessary to find this k-representation of W and add it to the set of accessing sequences.

In case the equivalence query was successful, $\mathcal{A}_L$ is not necessary a permutation-complete k-BPA for **S** since the definition equivalence queries only guarantees that one representation of every bad prefix is accepted. Assuming one is only interested in a non-permutation-complete k-BPA for **S**, the algorithm can terminate at this point. The learner must check permutation-completeness itself, which is handled by calling Algorithm 4.4. In a permutation-complete k-bad-prefix automaton

Algorithm 4.4 IsComplete

Input: Deterministic automaton $\mathcal{A}$ of arity k
Output: True iff $\mathcal{A}$ is permutation complete and a counterexample otherwise

1: **for** $\varsigma : \{1, \dots, k\} \to \{1, \dots, k\}$ **do**
2: **if** $\mathcal{L}(\mathcal{A}^\varsigma) \not\subseteq \mathcal{L}(\mathcal{A})$ **then**
3: **return** an element of minimal length from $\mathcal{L}(\mathcal{A}^\varsigma) \setminus \mathcal{L}(\mathcal{A})$
4: **return** Complete

$\mathcal{A}$ a k-representation t represents a bad-prefix if and only if some representations of a subset of $\mathsf{unzip}(t)$ is accepted by $\mathcal{A}$. Thus, permutation-completeness can be verified by permuting and projecting the language of $\mathcal{A}_L$ according to all function $\varsigma : \{1, \dots, k\} \to \{1, \dots, k\}$ and checking the subset-relation between $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{A}^\varsigma)$. In case $\mathcal{A}_L$ is a permutation-complete k-BPA, all the resulting sequences are included in the language of $\mathcal{A}_L$ since they only represent representations of subsets of $\mathsf{unzip}(\mathcal{L}(\mathcal{A}_L))$. If $\mathcal{A}_L$ is not permutation-complete, a counterexample $t \in (\Sigma^{k_L})^*$ will be found and the observation table can be extended by it. Otherwise, $\mathcal{A}_L$ is permutation-complete and the algorithm terminates.

This procedure is repeated until a permutation-complete k-BPA is learned. The resulting automaton is minimal with respect to Definition 3.15.

4.2 Example of L^*_{Hyper}

Before continuing with the analysis of L^*_{Hyper}, we provide an example run of the algorithm: Consider the HyperLTL formula $\varphi = \forall\pi.\forall\pi'.\ a_\pi \wedge \square(a_\pi \leftrightarrow a_{\pi'})$ over the alphabet $\mathsf{AP} = \{\{a\}, \{\}\}$. Note that φ describes a 2-safety hyperproperty. We introduce a and $\neg a$ as a shorthand for $\{a\}, \{\}$, respectively. The learner starts with an empty observation table $\mathcal{O}$ over the alphabet $\mathsf{AP}^1 = \mathsf{AP}$. Thus, the initial observation table $\mathcal{O}$, which is depicted in Figure 4.3, only contains the empty word ϵ and its extensions by the alphabet $(\epsilon \cdot \{a\}, \epsilon \cdot \{\})$. The observation table is not closed but

	ϵ
ϵ	0
a	0
$\neg a$	1

Figure 4.3: Initial observation table for learning $\forall\pi, \pi' \in \mathsf{Traces}(S).\ a_\pi \wedge \square(a_\pi \leftrightarrow a_{\pi'})$. It is consistent, but not closed.

consistent since there exists a $x \in S \cdot \Sigma$ such that for all $y \in S$ $\mathsf{row}(x) \neq \mathsf{row}(y)$ ($x := \epsilon \cdot a$). Thus, the first step is to close the observation table by adding $\epsilon \cdot a$ to the accessing sequences S and asking the corresponding membership queries. This yields a closed and consistent observation table. The corresponding deterministic automaton $\mathcal{A}_1$ which accepts the language $a^* \neg a (a \mid \neg a)^*$ can be constructed (Figure 4.4). Assume the teacher provides the following set of finite sequences

$\{a \cdot \neg a\}$ as a counterexample.

	ϵ
ϵ	0
$\epsilon \cdot \neg a$	1
$\epsilon \cdot a$	0
$\neg a \cdot a$	1
$\neg a \cdot \neg a$	1

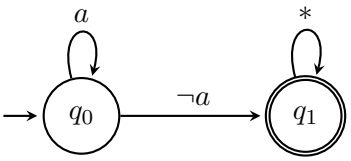

Figure 4.4: 2^{nd} observation table (on the left) for learning $\forall \pi, \pi' \in \mathsf{Traces}(S).\ a_\pi \wedge \square(a_\pi \leftrightarrow a_{\pi'})$. It is closed and consistent. The corresponding DFA (on the right).

Thus, we add $a \cdot \neg a$—a representation of $\{a \cdot \neg a\}$—and its prefix a to the set of accessing sequences (Figure 4.5). Thereafter we obtain a closed but not consistent observation table since the two sequences ϵ and $\epsilon \cdot a$ represent the same state, i.e., $\mathsf{row}(\epsilon) = \mathsf{row}(\epsilon \cdot a)$ and their Σ continuation languages differ, namely $\mathsf{row}(\epsilon \cdot \neg a) = 1 \neq 0 = \mathsf{row}(\epsilon \cdot a \cdot \neg a)$.

	ϵ
ϵ	0
$\epsilon \cdot \neg a$	1
$\epsilon \cdot a$	0
$a \cdot \neg a$	0
$\neg a \cdot a$	1
$\neg a \cdot \neg a$	1
$a \cdot a$	0
$a \cdot \neg a \cdot a$	0
$a \cdot \neg a \cdot \neg a$	0

Figure 4.5: 3^{rd} observation table for learning $\forall \pi, \pi' \in \mathsf{Traces}(S).\ a_\pi \wedge \square(a_\pi \leftrightarrow a_{\pi'})$. It is closed but not consistent.

Thus, we add the critical sequence $\neg a$ to the set of separating sequences E and finally receive a closed and consistent table (Figure 4.5). We can conjecture a new deterministic automaton $\mathcal{A}_2$ depicted in Figure 4.6. Querying $\mathcal{A}_2$ to the teacher, we receive a counterexample of size 2 since the $\mathcal{A}_2$ correctly accepts all counterexamples reasoning about one single trace. Assume we are given: $\{\{a \cdot \neg a\}\{\neg a \cdot a\}\}$

	ϵ	$\neg a$
ϵ	0	1
$\epsilon \cdot \neg a$	1	1
$\epsilon \cdot a$	0	0
$a \cdot \neg a$	0	0
$\neg a \cdot a$	1	1
$\neg a \cdot \neg a$	1	1
$a \cdot a$	0	0
$a \cdot \neg a \cdot a$	0	0
$a \cdot \neg a \cdot \neg a$	0	0

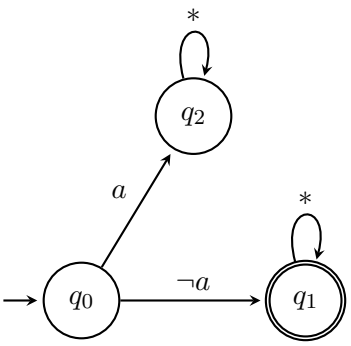

Figure 4.6: 4^{th} observation table (on the left) for learning $\forall \pi, \pi' \in \mathsf{Traces}(S).\ a_\pi \wedge \square(a_\pi \leftrightarrow a_{\pi'})$. The corresponding DFA (on the right).

Thus, the observation table $\mathcal{O}$ must be extended to Σ^2. Therefore Algorithm 4.2 is called with the current observation table and arity 2 yielding a new observation table, which is depicted in Figure 4.7.

	ϵ	$(\neg a, \neg a)$
ϵ	0	1
$\epsilon \cdot (\neg a, \neg a)$	1	1
$\epsilon \cdot (a, a)$	0	0
$(a, a) \cdot (\neg a, \neg a)$	0	0
$(a, \neg a)$	1	1
$(\neg a, a)$	1	1
$(\neg a, \neg a) \cdot (*)$	1	1
$(a, a) \cdot (a, a)$	0	0
$(a, a) \cdot (\neg a, a)$	1	1
$(a, a) \cdot (\neg a, \neg a) \cdot (a, a)$	0	0
$(a, a) \cdot (\neg a, \neg a) \cdot (\neg a, a)$	1	1
$(a, a) \cdot (\neg a, \neg a) \cdot (a, \neg a)$	1	0
$(a, a) \cdot (\neg a, \neg a) \cdot (\neg a, \neg a)$	0	0

Figure 4.7: 5^{th} observation table for learning $\forall \pi, \pi' \in \mathsf{Traces}(S).\ a_\pi \wedge \square(a_\pi \leftrightarrow a_{\pi'})$. The first observation table of arity 2. We employ the notation $x \cdot (*)$ in case x is accepted to denote that all extensions of x are accepted.

Having the extended observation table, we must add a representation of the counterexample $C = \{a \cdot \neg a, a \cdot a\}$ to the set of accessing sequences. C is a bad-prefix, which can be verified using a membership query. Thus no 2-representation of C is accepted by $\mathcal{A}_2$ and choosing the 2-representation $(a, a) \cdot (\neg a, a)$ and adding it to the set of accessing sequences resolves the current

counterexample. Afterward we yield a new observation table that is closed and consistent. It is depicted in Figure 4.8. Posing an equivalence check involving the most recent automaton

	ϵ	$(\neg a, \neg a)$
ϵ	0	1
$\epsilon \cdot (\neg a, \neg a)$	1	1
$\epsilon \cdot (a, a)$	0	0
$(a, a) \cdot (\neg a, \neg a)$	0	0
$(a, a) \cdot (\neg a, a)$	1	1
$(a, \neg a)$	1	1
$(\neg a, a)$	1	1
$(\neg a, \neg a) \cdot (*)$	1	1
$(a, a) \cdot (a, a)$	0	0
$(a, a) \cdot (\neg a, a)$	1	1
$(a, a) \cdot (\neg a, \neg a) \cdot (a, a)$	0	0
$(a, a) \cdot (\neg a, \neg a) \cdot (\neg a, a)$	1	1
$(a, a) \cdot (\neg a, \neg a) \cdot (a, \neg a)$	1	1
$(a, a) \cdot (\neg a, \neg a) \cdot (\neg a, \neg a)$	0	0
$(a, a) \cdot (\neg a, a) \cdot (*)$	1	1

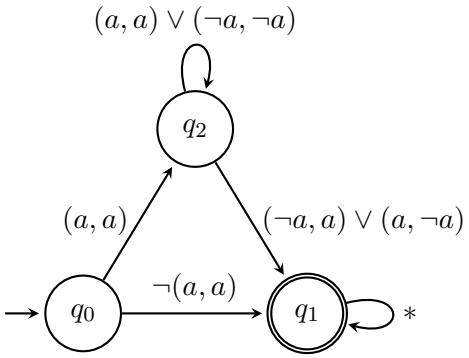

Figure 4.8: Final observation table (on the left) for learning $\varphi = \forall \pi, \pi' \in \mathsf{Traces}(S)$. $a_\pi \wedge \square(a_\pi \leftrightarrow a_{\pi'})$. A 2-BPA for φ (on the right). We employ the notation $x \cdot (*)$ in case x is accepted to denote that all extensions of x are accepted.

$\mathcal{A}_3$ does not yield a counterexample. Thus we are left to check whether $\mathcal{A}_3$ is permutation-complete. This is indeed the case and the algorithm will terminate. Hence, $\mathcal{A}_3$ is a minmial, tight, permutation-complete, deterministic 2-BPA for φ.

4.3 Correctness and Termination

Next, we prove the correctness and termination of L^*_{Hyper}. During a single execution, various automata of different arities are constructed. Hence we first bound the size of all these automata by the size of the outputed automaton We can achieve this bound by showing that the set of continuation languages for $\mathsf{BadPref}_{\vec{k'}}(\mathbf{S})$ is of larger cardinality than $\mathsf{BadPref}_{\vec{k}}(\mathbf{S})$ for $k \leq k'$.

Lemma 4.1
Let $\mathbf{S}$ be a regular k-safety hyperproperty, let $\mathcal{A}_k$ be the minimal deterministic automaton accepting $\mathsf{BadPref}_{\vec{k}}(\mathbf{S})$, and let $\mathcal{A}_{k'}$ be the minimal, deterministic automaton accepting $\mathsf{BadPref}_{\vec{k'}}(\mathbf{S})$ for $k' \leq k$. Then $|\mathcal{A}_{k'}| \leq |\mathcal{A}_k|$.

Proof By assumption, we know that $\mathsf{BadPref}_{\vec{k}}(\mathbf{S})$ is a regular language, i.e., $|\{\mathcal{F}_{\mathsf{BadPref}_{\vec{k}}(\mathbf{S})}(x) \mid x \in (\Sigma^k)^*\}|$ is finite according to Myhill-Nerode's theorem. Let $x, x' \in (\Sigma^{k'})^*$ be two arbitrary sequences with different continuation languages, i.e., $F_{\mathsf{BadPref}_{\vec{k'}}(\mathbf{S})}(x) \neq F_{\mathsf{BadPref}_{\vec{k'}}(\mathbf{S})}(x')$.

W.l.o.g., there exists a sequence $y \in (\Sigma^{k'})^*$ such that

$$x \cdot y \in \mathsf{BadPref}_{\vec{k'}}(\mathbf{S}) \text{ and } x' \cdot y \notin \mathsf{BadPref}_{\vec{k'}}(\mathbf{S})$$

We define $\bar{x}$, $\bar{x}'$, and $\bar{y}$ as the extension of x, x', and y to arity k ($\bar{x} = \mathsf{extend}_k(x)$, $\bar{x}' = \mathsf{extend}_k(x')$ and $\bar{y} = \mathsf{extend}_k(y)$). Since y was chosen to distinguish x and x' and $\mathsf{unzip}(z \cdot y) = \mathsf{unzip}(\bar{z} \cdot \bar{y})$ for $z \in \{x, y\}$, it follows that $\bar{x} \cdot \bar{y} \in \mathsf{BadPref}_{\vec{k}}(\mathbf{S})$ and $\bar{x}' \cdot \bar{y} \notin \mathsf{BadPref}_{\vec{k}}(\mathbf{S})$. Thus, $F_{\mathsf{BadPref}_{\vec{k}}(\mathbf{S})}(\bar{x}) \neq F_{\mathsf{BadPref}_{\vec{k}}(\mathbf{S})}(\bar{x}')$ and therefore

$$\left|\{F_{\mathsf{BadPref}_{\vec{k}}(\mathbf{S})}(x) \mid x \in (\Sigma^k)^*\}\right| \geq \left|\{F_{\mathsf{BadPref}_{\vec{k'}}(\mathbf{S})}(x) \mid x \in (\Sigma^{k'})^*\}\right| \tag{4.1}$$

because $x, x' \in \mathsf{BadPref}_{\vec{k'}}(\mathbf{S})$ were chosen arbitrarily. Further, it follows from 4.1 that regularity of $\mathsf{BadPref}_{\vec{k}}(\mathbf{S})$ implies that $\mathsf{BadPref}_{\vec{k'}}(\mathbf{S})$ is a regular language.

Let $\mathcal{A}_k$ be the minimal deterministic automaton accepting $\mathsf{BadPref}_{\vec{k}}(\mathbf{S})$ and let $A_{k'}$ be the minimal deterministic automaton accepting $\mathsf{BadPref}_{\vec{k'}}(\mathbf{S})$. The size of the minimal automaton accepting a regular language $\mathcal{L}$ over Σ is $|\{\mathcal{F}_{\mathcal{L}}(x) \mid x \in \Sigma^*\}|$ as a result of Myhill-Nerode's theorem [22]. So we conclude that $|\mathcal{A}_{k'}| \leq |\mathcal{A}_k|$.

■

Next, we prove that the learner poses at most n equivalence queries where n is the size of the minimal, permutation-complete k-BPA recognizing $\mathbf{S}$. First, the resolution for a non-closed or non-consistent observation table increases the size of $\mathsf{row}(S)$ by at least one. And every time a counterexample is added to the observation table it is not consistent afterward. Therefore there can be at most n equivalence queries posed. Otherwise the minimal automaton consistent with $\mathcal{O}$ is of size larger than n according to Lemma 2.10.

Before we conclude termination, we need to verify that Algorithm 4.3 correctly extends the observation table, i.e., the size of $\mathsf{row}(S)$ remains invariant under extension. We write $\mathsf{row}_{\mathcal{O}}(S)$ in order to reference the observation table we are currently interested in.

Lemma 4.2
Let $\mathcal{O} = (S, E, \Delta)$ be a closed and consistent observation table and $\mathcal{O}' = (S', E', \Delta')$ the observation table returned by calling Algorithm 4.3. It holds that $|\mathsf{row}_{\mathcal{O}}(S)| = |\mathsf{row}_{\mathcal{O}}(S')|$.

Proof Let k_L be the current arity for the learner and k' the new arity ($k' > k_L$). The first 5 lines of Algorithm 4.3 replace each sequence in S and E by their corresponding extension of arity k' Hence, $|S| = |S'|$ and $|E| = |E'|$. Next, the value of $\Delta'(\mathsf{extend}_{k'}(s \cdot e))$ is assigned to $\Delta(s \cdot e)$. The

assignment is sound because $\mathsf{unzip}(\mathsf{extend}_{k'}(s \cdot e)) = \mathsf{unzip}(s \cdot e)$ for all $s \in S, e \in E$, i.e., their membership queries must be answered equally. Now it follows by the construction of S', E' that $|\mathsf{row}_{\mathcal{O}}(S)| = |\mathsf{row}_{\mathcal{O}}(S')|$.

■

Using the last lemmas we conclude that L^*_{Hyper} terminates after at most n equivalence queries.

Lemma 4.3
L^*_{Hyper} terminates after at most n equivalence queries where n is the size of the minimal permutation-complete k-BPA for **S**.

Proof Let $\mathcal{O} = (S, E, \Delta)$ be the current observation table and let k_L denote the current arity of the alphabet. First, we show that during the algorithm the size of $\mathsf{row}(S)$ increases by at least one in every iteration of the main loop.

Consider $\mathcal{O}$ to be non-consistent or non-closed at the beginning of the main loop in L^*_{Hyper}. We distinguish the two cases:

1. $\mathcal{O}$ is not closed:
   There exists $s \in S$ and $a \in \Sigma^{k_L}$ such that $\mathsf{row}(s \cdot a) \notin \mathsf{row}(S)$. Hence, adding $s \cdot a$ to S increases the size of $\mathsf{row}(S)$ by one.
2. $\mathcal{O}$ is not consistent:
   There exists $s_1, s_2 \in S$ with $\mathsf{row}(s_1) = \mathsf{row}(s_2)$ and s_1 and s_2 can be distinguished by some sequence $a \cdot e$. Thus, after adding $a \cdot e$ to E the size of $\mathsf{row}(S)$ increased by at least one.

Further, note that the elements in the observation table are never deleted. Thus, according to Lemma 2.10, the table can be turned closed or consistent at most $n - 1$ times. Next, we claim that the observation table $\mathcal{O}$ is inconsistent after adding a counterexample $T \subseteq \Sigma^*$ to $\mathcal{O}$. We denote by X_{old} the state of X before adding the counterexample and by X_{new} the corresponding state after adding it and all its prefixes to $\mathcal{O}$. Let t be the chosen representation of T.

W.l.o.g., there exists a unique sequence of rows $r = r_0 \dots r_n \in \mathsf{row}_{old}(S_{old})^*$ corresponding to a run of u in $\mathcal{O}_{old}$ constructed by $r_0 = \mathsf{row}(\epsilon)$ and $r_i = \mathsf{row}(s_{i-1} \cdot u[i-1])$ for all $0 < i \leq n$ with $\Delta(r_n) = 1$ and there exists a sequence of rows $r' = r'_0 \dots r'_n \in \mathsf{row}_{new}(S_{new})^*$ corresponding to a run on u with $\Delta(r'_n) = 0$. The initial states of r and r' are equal.

Thus, there exists an $i > 0$ with $r_i \neq r'_i$ and for all $j < i$. $r_j = r'_j$. Moreover, $t[0, i-1]$ is not an element of S_{old} as otherwise there could not be a difference in r_{i-1} and r'_{i-1}. $\mathcal{O}$ is closed and consistent and therefore, there exists a sequence $s \in S$ with $\mathsf{row}_{old}(s) = r_{i-1}$ and s is an element of S_{new} with $\mathsf{row}_{new}(s) = \mathsf{row}_{old}(s)$. Hence, $\mathsf{row}_{new}(s) = r'_{i-1}$ but $r_i = \mathsf{row}_{new}(s \cdot t[i]) \neq \mathsf{row}_{new}(t[0, i]) = r'_i$, $t[0, i]$ is in S as all prefixes of t were added to S. Thus, after adding a counterexample the table is inconsistent.

As it was shown in Lemma 4.1, the size of the output automaton n is larger than all automata learned in-between and the information gathered for an arity k is carried over to the table of larger arities whenever Algorithm 4.3 is called, i.e., the size of $\mathsf{row}(S)$ is preserved, according to Lemma 4.2. Thus, by Lemma 2.10 at most n counterexamples can be added to $\mathcal{O}$.

A counterexample is added to the table in a failing equivalence query or in a failing completeness check but since every completeness check must be preceded by a successful equivalence query at most $n-1$ such queries can be posed before $|\mathsf{row}(S)| = n$. Therefore, L^*_{Hyper} terminates after at most n equivalence queries.

■

For the purpose to obtain correctness of the algorithm, we exploit the definition of the teacher and prove the correctness of Algorithm 4.4. The following lemma rephrases the permutation-completeness problem and we use it afterward to verify Algorithm 4.4.

Lemma 4.4
Let $\mathcal{A}_L$ be a finite automaton of arity k_L accepting one representation of every bad-prefix of **S**. $\mathcal{A}_L$ is a permutation-complete k-BPA for **S** if and only if for all functions $\varsigma \in \{1,\dots,k_L\} \to \{1,\dots,k_L\} :\ \mathcal{L}(\mathcal{A}_L^{\varsigma}) \subseteq \mathcal{L}(\mathcal{A}_L)$.

Proof

($\Rightarrow$) Let $\mathcal{A}_L = (Q, \Sigma^k, q_0, F, \delta)$ be a permutation-complete k-BPA for some k-safety property **S**, i.e., $\forall T \in \mathsf{BadPref}_k(\mathbf{S})$ every representation of T is accepted by $\mathcal{A}_L$. Let $\varsigma : \{1,\dots,k_L\} \to \{1,\dots,k_L\}$ be chosen arbitrarily and let $t \in \mathcal{L}(\mathcal{A}_L^{\varsigma})$. A word is accepted by $\mathcal{A}_L^{\varsigma}$ iff its permuted projection according to ς is accepted by $\mathcal{A}_L$. Thus, $t_\varsigma \in \mathcal{L}(\mathcal{A}_L)$ and $\mathsf{unzip}(t_\varsigma)$ is a bad prefix with $\mathsf{unzip}(t_\varsigma) \subseteq \mathsf{unzip}(t)$. Hence, $\mathsf{unzip}(t)$ is a bad-prefix and $\mathcal{A}_L$ is a permutation-complete k-BPA for **S**. It immediately follows that $t \in L(\mathcal{A}_L)$.

($\Leftarrow$) Let $\mathcal{A}_L = (Q, \Sigma^{k_L}, q_0, F, \delta)$ be a k_L-BPA for **S**, i.e., $\mathcal{A}_L$ accepts at least one representation t of every $T \in \mathsf{BadPref}_{k_L}(\mathbf{S})$. In particular, $\mathcal{A}_L$ accepts one representation of every bad prefix of minimal size. Let $t \in \mathcal{L}(\mathcal{A}_L)$ be chosen arbitrarily, i.e., $\mathsf{unzip}(t) \in \mathsf{BadPref}(\mathbf{S})$. For all $t' \in (\Sigma^{k_L})^{|t|}$ with $\mathsf{unzip}(t) \subseteq \mathsf{unzip}(t')$ there exist a function $\varsigma : \{1,\dots,k_L\} \to \{1,\dots,k_L\}$ such that $t'_\varsigma = t$. Thus, by construction of $\mathcal{A}^{\varsigma}$: $t' \in \mathcal{L}(\mathcal{A}^{\varsigma})$. This implies $t' \in \mathcal{L}(\mathcal{A}_L)$. Therefore, all representations of every superset of t are contained in $\mathcal{L}(\mathcal{A}_L)$ and as t was chosen arbitrarily it follows that $\mathcal{A}_L$ accepts every representation of every $T \in \mathsf{BadPref}_k(\mathbf{S})$. Hence, $\mathcal{A}_L$ is a permutation-complete k-BPA for **S**.

■

Lemma 4.5
Let $\mathcal{A}_L$ be a finite automaton of arity k_L accepting one representation of every bad prefix of **S**. Algorithm 4.4 checks if $\mathcal{A}_L$ is permutation-complete in time polynomial in $|\mathcal{A}_L|$ and space polynomial in k_T. In case $\mathcal{A}_L$ is not permutation-complete it provides a counterexample of size at most $|\mathcal{A}_L|^2$.

Proof Correctness of Algorithm 4.4 follows immediately from Lemma 4.4.

Next, we proof the claimed running time of Algorithm 4.4. First observe that all functions $\varsigma : \{1, \ldots, k_L\} \to \{1, \ldots, k_L\}$ can be iterated in space polynomial in k_L. The construction of $\mathcal{A}_L^\varsigma$ is in time polynomial in $\mathcal{A}_L$ by traversing $\mathcal{A}_L$ once and adapting the transition relations according to ς. Thus, the size of $\mathcal{A}^\varsigma$'s state space is upper bounded by the state space of $\mathcal{A}_L$. $\mathcal{L}(\mathcal{A}^\varsigma) \subseteq \mathcal{L}(\mathcal{A}_L)$ can be verified by constructing the cross-product $\mathcal{A}_\times = (Q \times Q, \Sigma^{k_L}, (q_0, q_0), F^\varsigma \times \bar{F}, \delta_\times)$ of $\mathcal{A}_L^\varsigma = (Q^\varsigma, \Sigma^{k_L}, q_0, F^\varsigma, \delta^\varsigma)$ and $\overline{\mathcal{A}}_L = (Q, \Sigma^{k_L}, q_0, \overline{F}, \overline{\delta})$ with $\delta_\times((q, q'), a) = a(\delta^\varsigma(q, a), \overline{\delta}(q', a))$. This is possible in time quadratic in $|\mathcal{A}_L|$. Afterward, an emptiness check on $\mathcal{A}_\times$ can be performed in time linear in $|Q|^2$: Using a breadth-first search, we receive a counterexample t of minimal length in case one exists. This minimal counterexample is of length at most $|Q|^2$ as the minimal counterexample cannot reach any state of $\mathcal{A}_\times$ more than once.

■

We conclude this section by proving the total correctness of L^*_{Hyper} in the following theorem.

Theorem 4.6
L^*_{Hyper} terminates learning a minmial, tight, permutation-complete, deterministic k-BPA $\mathcal{A}_L$ for a regular k-safety hyperproperty **S** and L^*_{Hyper} poses at most n equivalence queries where n is the size of $\mathcal{A}_L$.

Proof According to Lemma 4.3, L^*_{Hyper} terminates after at most n equivalence queries, where n is the size of the minimal k-BPA recognizing the bad prefixes of **S**. The correctness of the output follows from the definition of the teacher and Lemma 4.5. The output automaton is minimal because the definition of a minimally adequate teacher requires that all counterexamples are of minimal arity k and Lemma 2.10. Thus, the total correctness of L^*_{Hyper} follows.

■

4.4 Running Time Analysis

In the following theorem, we provide complexity bounds regarding L^*_{Hyper}. The analysis itself is similar to the running time analysis of L^* with a special interest in the changing arity k.

Theorem 4.7 (Learning of k-Bad-Prefix Automata)
L^*_{Hyper} learns a minmial, tight, permutation-complete, deterministic k-BPA $\mathcal{A}$ for a regular k-safety hyperproperty **S** in time polynomial in n and m and time exponential in k where n is the size of $\mathcal{A}$, m is the length of the longest counterexample, and k is the size of the largest counterexample provided during an equivalence queries.

Proof Besides the size of the output, the algorithm is dependent on the length of the largest counterexample received. We denote by m the maximal length of a counterexample, by n the

size of the output $\mathcal{A}$, and by k_G the maximum size of a counterexample provided by the teacher. $\mathcal{O} = (S, E, \Delta)$ is the observation table used.

At first, we claim that the recursion depth of Algorithm 4.1 is bounded by k_G. By definition, a minimally adequate teacher can provide a counterexample containing more traces then all previous counterexamples at most k_G times and the observation table is extended every time a new counterexample reasoning about more traces is provided and the arity of the observation table never decreases. Thus, the claim holds. We analyze a single recursion step of L^*_{Hyper}. Let us fix some arbitrary $k_L \in \mathbb{N}$ with $k_L \leq k_G$. The analysis stops as soon as a counterexample is provided by the teacher, for which more than k_L traces are needed to express it or the algorithm terminates successfully.

According to Lemma 2.10, the size of $\mathsf{row}(S)$ is bounded by n. Every time the observation table is found non-closed, some sequence $s \in (\Sigma^{k_L})^*$ is added to the set of accessing sequences S. Hence, a non-closed observation table can occur at most $n-1$ times since each resolution of a non-closed table adds one element s to S with $\mathsf{row}(s) \notin S$. The teacher only provides counterexamples as a result of a failed equivalence query. Thus at most $n-1$ counterexamples of size m are added to S according to Lemma 4.3. Regarding an unsuccessful completeness check, the corresponding counterexample can be of size at most n^2 and again, there are at most $n-1$ such counterexamples. Given some counterexample, every prefix of it is added to S and therefore, the size of S is bounded by

$$|S| \leq \underbrace{(n-1)\cdot 1}_{\text{closed/consistent}} + \underbrace{(n-1)\cdot m}_{EQ} + \underbrace{n^2\cdot(n-1)}_{\text{complete}} \in O(n^3 + n\cdot m)$$

Further, the length of the longest element in S denoted by $\max(S)$ is bounded by $|\max(S)| \leq \max(n^2, m)\cdot(n-1) + (n-1) = O(n^3 \cdot m)$ (the longest counterexample for a completeness or an equivalence check and each appended by at most $n-1$ labels every time the table is transformed to become closed or consistent).

Every time an inconsistent observation table is discovered, one element is added to E. Hence, this can happen at most $n-1$ times, since $\mathsf{row}(S) \leq n$ by Lemma 2.10. Thus, the size of E is bounded by $|E| \leq n$ as E is initially of size 1. Therefore, the size of $(S \cup S\cdot\Sigma)\cdot E$ is bounded by

$$|\mathcal{O}| = |(S \cup S\cdot\Sigma)\cdot E| \leq (|S| + |S|\cdot|\Sigma^{k_L}|)\cdot|E| = O((n^4 + n^2\cdot m)\cdot|\Sigma|^{k_L})$$

and the longest sequence is of length at most

$$|\max((S \cup S\cdot\Sigma)\cdot E)| \leq |S| + |E| + 1 = O(n^3\cdot m)$$

Let T be a counterexample of size $k' > k_L$, which is returned from a failed equivalence query. At this point, the arity of $\mathcal{O}$ is adapted to k_L using Algorithm 4.3. The algorithm does not change the number of entries in $\mathsf{row}(S)$ and such a counterexample can be obtained at most k_G times.

Next, we consider the different operations performed in L^*_{Hyper} and their respective running

time. Checking consistency or closedness can be done in time polynomial in the size of $\mathcal{O}$. The corresponding resolution methods consist of adding one element to S or E, which requires posing at most $O((n^3 + n \cdot m) \cdot |\Sigma|^{k_G})$ membership queries of entries of length at most $O(n^2 \cdot m)$. The $|\Sigma|^{k_G}$ term is due to the extension of every sequence by all labels in Σ^k to determine $\Delta(S \cdot \Sigma^k)$, for the learner's current arity k. Both of the methods are performed at most $n-1$ times.

Constructing a DFA $\mathcal{A}$ from a closed and consistent observation table can be performed in polynomial time in the size of the observation table and is done at most $n-1$ times. In case a counterexamples T does not represents a bad-prefix, adding it to $\mathcal{O}$ requires finding a correct representation $t \in (\Sigma^{k_G})^*$ of T. This can be done in time polynomial in $|\mathcal{A}|$ and space polynomial in k_G. Afterward all prefixes of t are added to S involving at most $m \cdot \Sigma^{k_G} \cdot |E|$ membership queries.

Checking completeness for $\mathcal{A}$ can be done in time polynomial in n and space polynomial in k_G as claimed by Lemma 4.5. Resolving the corresponding counterexample involves adding at most n^2 sequences each combined with $n^2 \cdot |\Sigma^{k_G}|$ membership queries. Thus, the time complexity of L^*_{Hyper} is polynomially bounded in n and m and exponential in k_G.

■

4.5 Properties for Learning Tight Bad-Prefix-Automata

Above we discussed the differences between tight and fine bad-prefix automata and concluded that we prefer tight over fine automata in most cases. Now the following question arises: How does the information of the teacher influence the learned automaton? We can show that regarding 'tightness' or 'fineness' of the teacher's knowledge with respect to equivalence queries does not affect size of the output automaton.

Proposition 4.8
Let $\mathcal{A}_L$ be the learned k-bad prefix automaton of arity k_L returned from L^*_{Hyper} for some regular k-safety hyperproperty $\mathbf{S}$ with a teacher answering equivalence queries according to a fine k-bad-prefix automaton $\mathcal{A}'$ and membership queries according to $\mathsf{BadPref}^{\rightarrow}_{k_L}(\mathbf{S})$. Then $\mathcal{A}_L$ is a minmial, tight, permutation-complete, deterministic k-bad-prefix automaton for $\mathbf{S}$.

Proof Assume that the learned permutation-complete automaton $\mathcal{A}_L = (Q, \Sigma^{k_L}, q_0, F, \delta)$ is a fine but not a tight k-bad-prefix automaton for $\mathbf{S}$. We show that every k_L-representation of a bad-prefix for $\mathbf{S}$ is accepted by $\mathcal{A}_L$ and thus, $\mathcal{L}(\mathcal{A}_L) = \mathsf{BadPref}^{\rightarrow}_{k_L}(\mathbf{S})$.

Let $t \in (\Sigma^{k_L})^*$ represent a bad-prefix of $\mathbf{S}$ and let $q_t \in Q$ be the state reached in $\mathcal{A}$ on a run of t; q_t is unique since $\mathcal{A}_L$ is deterministic. Consider the following two cases for q_t:

- $q_t \in F$: Then t is accepted by $\mathcal{A}$ and we are done.
- $q_t \notin F$: Then t is a k_L-representation of a bad-prefix and thus, for every trace $\sigma \geq t$ a run

of σ on $\mathcal{A}_L$ starting in q_L eventually reaches an accepting state. Consequently, for every k_L-representation s of a set of traces of length $|\mathcal{A}_L| + 1$ the run of s starting in q_t must reach an accepting state. Otherwise, there would exist an infinite non-accepting loop, which contradicts t being a representation of a bad-prefix.

Let $t' \in (\Sigma^{k_L})^*$ be the longest sequence such that the run of t' on $\mathcal{A}_L$ starting in q_t is accepted and every prefix of t' is not accepted by $\mathcal{A}_L$; according to the above discussion, such a trace must exist. Let $\tilde{t} = t'[0, |t'| - 2]$. By the choice of t' the state $q_{\tilde{t}}$ reached by a run on $\tilde{t}$ starting in q_t is not accepting. However, every state reachable from $q_{\tilde{t}}$ is accepting.

Next, let s be the sequence associated with $q_{\tilde{t}}$ in the observation table $\mathcal{O}$, since $\mathcal{A}_L$ is deterministic and closed every extension of s exists in $S \cup S \cdot \Sigma^{k_L}$ and it is accepted, i.e., $\Delta(s, \epsilon) = 1$. Thus, s must be a k_L-representation of a bad-prefix because all Σ^{k_L} extensions of s are accepted and the corresponding membership query involving $\mathsf{unzip}(s)$ must have been answered positively. Consequently, $q_{\tilde{t}}$, the state associated with $\mathsf{row}(s)$ is accepting. This contradicts our choice of s. Thus, such a state s does not exist and every state that is not accepting in $\mathcal{A}_L$ can only be reachable on a run of a k_L-representation t that does not represent a bad-prefix. Hence, $\mathcal{A}_L$ accepts every representation of every k-bad-prefix for $\mathbf{S}$ which classifies $\mathcal{A}_L$ as tight.

■

Chapter 5

Application to HyperLTL

To this point, we defined a learning algorithm for k-BPAs. Yet, algorithms answering the posed questions were not touched. Thus, the following chapter is dedicated to answering queries 'efficiently'. In Chapter 6, we will show that the task of answering membership queries is even for 1-safety hyperproperties undecidable in general. Nonetheless, for HyperLTL formulas in the $\forall$-fragment membership and equivalence queries can be decided. We restrict ourselves to safe HyperLTL formulas (Definition 3.5). This fragment appears convenient since deciding whether a formula is a part of it can solved efficiently.

Regarding equivalence queries, a first, straight-forward approach would involve constructing a k-BPA for the HyperLTL formula φ and afterward, performing the earlier introduced equivalence check on the learned and constructed automata. However, this approach is false since the learned automaton must is not necessarily a k-BPA, which the earlier equivalence check requires. Further, it is possible to perform a more efficient algorithm that only constructs nondeterministic k-BPAs for φ. In addition, membership queries can be resolved efficiently due to a reduction to checking bad-prefixes for an LTL formula.

Let k_L denote the arity of the deterministic automaton $\mathcal{A}_L = (Q_L, \Sigma^{k_L}, q_0, F, \delta)$ learned so far. Let $\varphi = \forall\pi_1 \dots \forall\pi_k.\ \psi$ be a safe HyperLTL formula describing the target-language and let $\mathcal{B}_\varphi$ be the nondeterministic safety automaton for φ constructed using Lemma 3.6. Let k_G be the minimal k such that $\mathcal{L}(\varphi)$ is k-safe and let k_T denote the arity of φ, i.e., the number of trace-quantifiers in φ.

According to Kupferman and Lampert, an nondeterministic bad-prefix automaton fine for ψ can be constructed in time $2^{O(|\psi|)}$ [23]. Thus, adapting the construction in Lemma 3.6 we obtain a fine, nondeterministic k-BPA for φ of size exponential in $|\psi|$:

Lemma 5.1
Let $\varphi = \forall\pi_1 \dots \forall\pi_{k_T}.\ \psi$ be a safe HyperLTL formula. We can construct a non-permutation-complete, fine, nondeterministic k_T-BPA $\mathcal{N}_\varphi$ for φ in time $2^{O(|\psi|)}$.

5.1 Deciding Membership Queries

We start with the task of resolving membership queries provided a set of finite traces T. We reduce these queries to deciding bad-prefixes of LTL formulas. For T to be a bad-prefix of φ at least one k_T-representation $t \in (\Sigma^{k_T})^*$ of a subset of T must be a bad-prefix of ψ.

Lemma 5.2
Given a safe HyperLTL formula $\varphi = \forall \pi_1 \dots \forall \pi_{k_T}.\, \psi$ and a set of finite traces $T = \{t_1, \dots, t_k\} \subseteq \Sigma^n$. T is a bad-prefix of φ if and only if there exist an assignment $\varsigma : \{1, \dots, k_T\} \to \{1, \dots, k\}$ such that $\mathsf{zip}_\varsigma(T)$ is a bad-prefix of ψ.

Proof

($\Leftarrow$) There exist an assignment $\varsigma : \{1, \dots, k_T\} \to \{1, \dots, k\}$ and $\mathsf{zip}_\varsigma(T)$ is a bad-prefix of ψ. Thus, for every set $T' \subseteq (\Sigma)^\omega$ extending T ($T' \geq T$) there exists a trace assignment $\Pi : \mathcal{V} \to T'$ assigning to each π_i a trace $t' \in T'$ that extends $t_{\varsigma(i)}$. By assumption, $\Pi \not\models_\emptyset \psi$ and therefore, $T' \not\models \forall \pi_1 \dots \forall \pi_{k_T}.\, \psi$.

($\Rightarrow$) Proof by contraposition. Assume that for all functions $\varsigma : \{1, \dots, k_T\} \to \{1, \dots, k\}$ we have $\mathsf{zip}_\varsigma(T) \notin \mathsf{BadPref}(\psi)$, i.e., $\exists t' \in (\Sigma^{k_T})^\omega.\, t' \geq \mathsf{zip}_\varsigma(T) \wedge t' \models_\emptyset \psi$ (remember: $t' \models_\emptyset \psi$ refers to the isomorphic trace assignment Π). Let $\varsigma : \{1, \dots, k_T\} \to \{1, \dots, k\}$. Then,

$$
\begin{aligned}
& \forall \varsigma. \exists t' \in (\Sigma^{k_T})^\omega.\, t' \geq \mathsf{zip}_\varsigma(T) \wedge t' \models_\emptyset \psi \\
\Rightarrow\ & \forall \varsigma. \exists t' \in (\Sigma^{k_T})^\omega.\, t' \geq \mathsf{zip}_\varsigma(T) \wedge \exists \Pi : \mathcal{V} \to \mathsf{unzip}(t').\, \Pi \models_\emptyset \psi \\
\Leftrightarrow\ & \forall \varsigma. \exists t' \in (\Sigma^{k_T})^\omega.\, t' \geq \mathsf{zip}_\varsigma(T) \wedge \neg(\forall \Pi : \mathcal{V} \to \mathsf{unzip}(t').\, \Pi \not\models_\emptyset \psi) \\
\Rightarrow\ & \forall \varsigma. \exists t' \in (\Sigma^{k_T})^\omega.\, t' \geq \mathsf{zip}_\varsigma(T) \wedge \neg(\emptyset \not\models_{\mathsf{unzip}(t')} \varphi) \\
\Leftrightarrow\ & \forall \varsigma. \exists t' \in (\Sigma^{k_T})^\omega.\, t' \geq \mathsf{zip}_\varsigma(T) \wedge \emptyset \models_{\mathsf{unzip}(t')} \varphi \\
\Leftrightarrow\ & \forall \varsigma. \mathsf{unzip}(t) \notin \mathsf{BadPref}(\varphi) \\
\Leftrightarrow\ & \mathsf{unzip}(t) \notin \mathsf{BadPref}(\varphi)
\end{aligned}
$$

This contradicts our assumption and therefore $T \in \mathsf{BadPref}(\varphi)$ implies the existence of some $\varsigma : \{1, \dots, k_T\} \to \{1, \dots, k\}$ such that $\mathsf{zip}_\varsigma(T) \in \mathsf{BadPref}(\psi)$.

■

Following the idea of Lemma 5.2 membership queries can be answered in a similar fashion to membership queries for a safe LTL formula. We summarize the complexity of this approach in the following theorem.

Theorem 5.3
Let T be a set of traces, with each trace being of length n, and let $\varphi = \forall \pi_1 \dots \forall \pi_{k_T}.\, \psi$ a safe HyperLTL formula. The problem of deciding whether T is a bad-prefix of φ can be solved in time polynomial in $|n|$ and space polynomial in $|\psi|$ and $k_T \cdot \log(|T|)$.

Proof The problem of deciding whether $t \in \Sigma$ is a bad-prefix for a safe LTL formula φ is in space polynomial in $|\varphi|$ and time polynomial in $|t|$ according to Baier and Katoen [8]. Combining their result and Lemma 5.2, we can decide for all $\varsigma : \{1, \dots, k_T\} \to \{1, \dots, |T|\}$ whether $\mathsf{zip}_\varsigma(T)$ is a bad-prefix of ψ in space polynomial in $k_T \cdot \log(|T|)$, since there are at most $|T|^{k_T}$ many different functions ς which each can be described in $\log(|T|^{k_T}) = O(k_T \cdot \log(|T|))$ bits .

■

For the purpose of completeness, we present a second algorithm solving membership queries symbolically. To this end, we employ the decidability results of HyperLTL by Finkbeiner and Hahn [15]. In practice, the second algorithm is expected to outperform the first one due to its dependence on SAT solving and the efficiency of state-of-the-art SAT solvers [9].

Theorem 5.4
Let T be a set of finite traces and let $\varphi = \forall \pi_1 \dots \forall \pi_{k_T}.\ \psi$ be a safe HyperLTL formula. T is a bad-prefix of φ if and only if the following HyperLTL formula is unsatisfiable:

$$\varphi' := \exists \pi'_1 \dots \exists \pi'_n \underbrace{\forall \pi_1 \dots \forall \pi_{k_T}.\ \pi'_1 \geq t_1 \wedge \dots \wedge \pi'_n \geq t_n \wedge \psi(\pi_1, \dots, \pi_{k_T})}_{\Psi(\pi'_1, \dots, \pi'_n)}$$

Proof

($\Rightarrow$) Let $T = \{t_1, \dots t_n\} \subseteq \Sigma^*$ be a bad-prefix of φ. Thus, for all $T' \subseteq \Sigma^\omega$ with $T \leq T'$ it holds: $T' \not\models \varphi$, i.e., there does not exist a set of traces T' extending T and satisfying φ. Therefore, no traces $\pi'_1, \dots, \pi'_n$ exist that satisfy ψ and φ' is unsatisfiable.

($\Leftarrow$) Let $T = \{t_1, \dots t_n\} \subseteq \Sigma^*$ be a set of traces and let φ' be unsatisfiable. We distinguish the following two cases:

1. φ is unsatisfiable:
   Thus, no set of infinite traces can satisfy φ and T is, like every other non-empty set of traces, a bad-prefix of φ.
2. φ is satisfiable:
   Then, the conjunction of $\pi'_1 \geq t_1 \wedge \dots \wedge \pi'_n \geq t_n$ and $\psi(\pi_1, \dots, \pi_{k_T})$ is unsatisfiable in the context of the given quantifiers. Thus, there does not exists a set of traces $T' \subseteq \Sigma^\omega$ having n traces that extend $t_1, \dots, t_n$ and $T' \models \varphi$. Therefore, T satisfies the definition of a bad-prefix of φ.

■

According to the results of Finkbeiner and Hahn in [15] and since φ' is in the bounded $\exists^*\forall^*$ fragment of HyperLTL, Theorem 5.4 grants us an algorithm deciding membership queries in space exponential in $|\varphi'| = n \cdot m \cdot |\Sigma| + k_t + |\varphi|$ where m is the length of the traces in T.

5.2 Deciding Equivalence Queries

In this section, we focus on the resolution of equivalence queries. We proceed as follows:

1. Check if every word accepted by $\mathcal{A}_L$ is a representation of a bad-prefix of φ.
2. Check whether every set of traces violating φ has a bad-prefix whose representation is accepted by $\mathcal{A}_L$

In order to decide whether every sequence accepted by $\mathcal{A}_L$ is a representation of a bad-prefix, we do not employ the equivalence check presented in Theorem 3.11 since it involves constructing a deterministic k-bad-prefix automaton for φ, which is of size doubly exponential in $|\varphi|$. We start by solving 1.:

Lemma 5.5
Given a safe HyperLTL formula $\varphi = \forall\pi_1 \dots \forall\pi_{k_T}.\ \psi$ and a deterministic automaton $\mathcal{A}_L = (Q, \Sigma^{k_L}, q_0, F, \delta)$. The problem of checking if every word w accepted by $\mathcal{A}_L$ is a k_L-representation of a bad-prefix of φ and otherwise providing a minimal counterexample is in time polynomial in $|\mathcal{A}_L|$, exponential in $|\psi|$ and doubly exponential in k_T.

Proof We transform φ into a HyperLTL formula $\varphi' = \forall\pi_1 \dots \forall\pi_{k_T}.\ \psi'(\pi_1, \dots, \pi_{k_T})$ where

$$\psi'(\pi_1, \dots, \pi_{k_T}) = \bigwedge_{\varsigma:\{1,\dots,k_T\}\to\{1,\dots,k_T\}} \psi(\pi_{\varsigma(1)}, \dots, \pi_{\varsigma(k_T)})$$

Note that the trace property described by ψ' is permutation-complete with respect to Σ^{k_T}, i.e., for all $t \in (\Sigma^{k_T})^*$, it holds $t \models_{\text{LTL}} \psi' \Leftrightarrow \emptyset \models_{\mathsf{unzip}(T)} \varphi$ where the LTL semantic is adjusted such that a_{π_i} holds if a holds in the i-th component. The size of φ' is exponential in k_T and we can construct a nondeterministic safety automaton $\mathcal{N}_{\varphi'} = (Q_\varphi, \Sigma_\varphi^{k_T}, q_{0,\varphi}, F_\varphi, \Delta_\varphi)$ over Σ^{k_L} accepting all infinite sequences that represent a set T of at most k_T traces such that $T \models \varphi$ (see Lemma 5.1). The size of $\mathcal{N}_{\varphi'}$ is exponential in the size of ψ'.

In order to check whether $\mathcal{A}_L$ accepts any sequence that does not represent a bad-prefix, we construct the nondeterministic product automaton $\mathcal{A}_\otimes = \{Q \times Q_\varphi, \Sigma^{k_L}, (q_0, q_{0,\varphi}), F \times F_\varphi, \delta_\otimes\}$ of $\mathcal{A}_L$ and $\mathcal{N}_\varphi$ with the following transition relation:

$$\Delta_\otimes((q, q_\varphi), a) = \{(q', q'_\varphi) \mid q' = \delta(q, a) \wedge q'_\varphi \in \Delta(q_\varphi, \mathsf{extend}_{k_T}(a))\},$$

for all $a \in \Sigma^{k_L}$. The automaton $\mathcal{A}_\otimes$ accepts an infinite word $w \in (\Sigma^{k_L})^\omega$ if and only if $w \in \mathcal{L}(\mathcal{N}_{\varphi'})$, i.e., $\mathsf{unzip}(w) \models \varphi$, and there exists a run of w on $\mathcal{A}_L$ that visits infinitely many accepting states.

Thus, if $\mathcal{L}(\mathcal{A}_\otimes) \neq \emptyset$, then there exists an infinite sequence that is not a bad-prefix and it is accepted by $\mathcal{A}_L$. Finding a minimal bad-prefix can then be achieved by performing a BFS looking for an accepting state in $\mathcal{A}_\otimes$ with a nested DFS aiming for accepting loops, similar to the nested DFS algorithm for model-checking LTL given by Baier and Katoen [8]. In case an accepting loop is found: Provide the path to the accepting state, from which the nested-search was started as a

bad-prefix. Such a nested-BFS is in time polynomial in the size of $\mathcal{A}_\otimes$ and the counterexample is of size at most linear in $|\mathcal{A}_\otimes|$.

Thus the entire algorithm is in time polynomial in $|\mathcal{A}_L|$, exponential in $|\psi|$ and doubly exponential in k_T.

■

Next, we are concerned with the problem of deciding whether $\mathcal{A}_L$ accepts a k_L-representation of every bad-prefix of φ. Therefore, we give an extension of Theorem 3.11 and show that for one direction of the equivalence the complexity does not change in case one k-BPA is deterministic and the other one is nondeterministic:

Lemma 5.6
Let $\varphi = \forall\pi_1 \ldots \forall\pi_{k_T}.\ \psi$ be a safe HyperLTL formula and let $\mathcal{A}_L$ be a deterministic automaton. The problem of deciding whether $\mathcal{A}_L$ recognizes a representation of a subset of every bad-prefix $T \in \mathsf{BadPref}_{k_T}(\varphi)$ and otherwise providing a minimal counterexample can be solved in time polynomial in $|\mathcal{A}_L|$, exponential in $|\psi|$, and doubly exponential in k_T.

Proof For φ, we can construct a fine, nondeterministic k-BPA $\mathcal{N}_\varphi = (Q_\varphi, \Sigma^{k_T}, q_{0,\varphi}, F_\varphi, \Delta_\varphi)$ in time exponential in $|\psi|$, according to Lemma 5.1. In Theorem 3.11, we provided an equivalence check which solves the current problem with respect to two deterministic k-bad-prefix automata. However, since only one of the two automata is complemented only $\mathcal{A}_L$ has to be deterministic whereas $\mathcal{N}_\varphi$ can be nondeterministic as well. And the running-time bound follows from Theorem 3.11.

In case the equivalence does not hold, a counterexample of minimal length can easily be produced in the size of the cross-product automaton, similar to Lemma 5.5.

■

A counterexample T obtained from Lemma 5.6 (a bad-prefix not accepted by $\mathcal{A}_L$) is in general not a counterexample of minimal size but only of minimal length. Nevertheless, according to Lemma 5.6 no representation of a subset of T is accepted by $\mathcal{A}_\mathcal{L}$. Thus, by transforming T to the minimal subset of traces $T' \leq T$ such that T' is a counterexample, we obtain a minimal one:

Lemma 5.7
Given a safe HyperLTL formula φ and $T = \{t_1, \ldots, t_{k'}\} \in \mathsf{BadPref}_{k_T}(\varphi)$ a finite set of traces of equal length. Finding a minimal bad-prefix $T' \leq T$ is in time polynomial in n and space polynomial in $|\psi|$, k_T.

Proof For some $i, j \in \mathbb{N}$ with $i, j \leq n$, we define $T[i,j] := \{t[i,j] \mid t \in T\}$. The length of a bad-prefix T is minimal if $T[0, n-1]$ is not a bad-prefix. Hence, we can use membership queries to reduce the length of T and $\log(n)$ queries suffice to obtain a bad-prefix of minimal length

using a binary search. According to Theorem 5.3, this is computable in time polynomial in $\log(n)$ and space polynomial in $|\psi|$, k_T.

Further, the number of traces in T is not necessary minimal. REDUCE produces a bad-prefix $T' \subseteq T$ such that T' is of minimal size, i.e., every subset of T' is not a bad-prefix. Note that given a bad-prefix T of minimal length the length of the minimal bad-prefix $T' \leq T$ is equal to the length of T. Thus, it suffices to reduce the length once.

Algorithm 5.1 Reduce

Input: Set of traces $T = \{t_1, \ldots, t_n\} \subseteq \Sigma^m$, teacher $\mathcal{T}$
Output: $T' \subseteq \Sigma^{m'}$ such that $MQ_{\mathcal{T}}(T')$ and for all $\tilde{T}' \leq T' : \neg MQ_{\mathcal{T}}(\tilde{T}')$

1: **for** $i = 1, i \leq n, i = i + 1$ **do**
2: $\quad T := T \setminus \{t_i\}$
3: $\quad$ **if not** $MQ_{\mathcal{T}}(T)$ **then**
4: $\quad\quad T := T \cup \{t_i\}$
5: **return** T

We prove the correctness of REDUCE by contraposition. Let T_{in} denote an arbitrary input and T_{out} denote the corresponding output of REDUCE. Assume there is a bad-prefix $T' \subsetneq T_{out}$. Thus there exists a $t \in T_{out}$ such that $t \notin T'$. Hence for all $\tilde{T} \supseteq T_{out}$: $\tilde{T} \setminus \{t\}$ is a bad-prefix. Therefore, $t \notin T_{out}$ since every membership query with $\tilde{T} \setminus \{t\}$ must resolve to true. Hence, we have a contradiction and T_{out} is of minimal size.

REDUCE poses at most k_T many membership queries with trace sets of length at most n. Thus, REDUCE is in time polynomial in n and space polynomial in $|\varphi|$, k_T according to Theorem 5.3. ■

The equivalence-check provided so far is only concerned about whether for every bad-prefix of φ a representation of a subset is accepted by $\mathcal{A}_L$. In other words, an automaton satisfying the equivalence check must not necessarily be a k-bad-prefix automaton for φ because such an automaton must accept one representation for every k-bad-prefix.

In order to better understand this subtle difference consider the following 2-safety hyperproperty $\varphi = \forall\pi.\forall\pi'.a_\pi \wedge b_{\pi'}$ and the automaton $\mathcal{A}$ depicted in Figure 5.1. Every set of traces that violates φ does have a prefix whose representation is accepted by $\mathcal{A}$. Yet, $\mathcal{A}$ is not a 2-BPA for φ because no representation of $\{\{b\}, \{a, b\}\}$ is accepted by it.

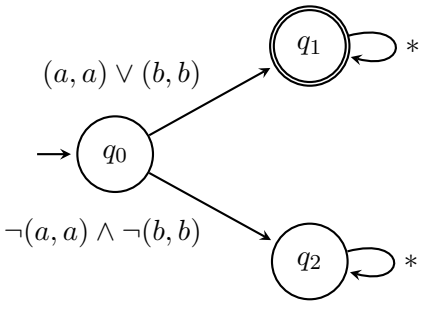

Figure 5.1: Not a 2-BPA for $\varphi = \forall\pi.\forall\pi'.a_\pi \wedge b_{\pi'}$

However, if $\mathcal{A}_L$ accepts a k-representation of a subset of every bad-prefix, the arity k_L suffices to express every bad-prefix of φ. Thus, at this point we can replace Lemma 5.6 by the following algorithm. Lemma 5.8 provides counterexamples until $\mathcal{A}_L$ accepts a k-representation for every bad-prefix $T \in \mathsf{BadPref}_{k_G}(\mathbf{S}_k)$, under the assumption of having the minimal arity k_G. For the next theorem,

it is crucial that all automata constructed during the learning algorithm are bad-prefix automata over the alphabet Σ^{k_L} (not k_L-bad-prefix automata over Σ).

The following theorem does not only check whether one k-representation of every bad-prefix is accepted by $\mathcal{A}_L$ but it decides whether for every representation of a bad-prefix a permutation is accepted by $\mathcal{A}_L$. The set of all permutations of a k-representation $t \in (\Sigma^k)^*$ is defined as follows:

$$\mathsf{Perm}(t) = \{t_\varsigma \mid \varsigma : \{1, \dots, k\} \to \{1, \dots, k\} \text{ and } \varsigma \text{ is a permutation}\}$$

Thus, the following construction is stronger than the required property for a minimally adequate teacher. To help understand the difference, consider the following example:

Example: Let $\Sigma = \mathcal{P}(\{a, b\})$ be the alphabet and $T = \{\{a\}, \{b\}\}$ a bad-prefix for some k-safety hyperproperty **H**. A 3-bad-prefix automaton $\mathcal{A}$ for **H** must accept one 3-representation of T, for example: $(\{a\}, \{a\}, \{b\}) \in \mathcal{L}(\mathcal{A})$ and it is not required to accept any representation extending $(\{a\}, \{b\}, \{b\})$. But an automaton that accepts one permutation of every 3-representation of a 3-bad-prefix of **H** would have to accept such a representation.

Lemma 5.8
Let $\varphi = \forall \pi_1 \dots \forall \pi_{k_T}.\ \psi$ be a safe HyperLTL formula and $\mathcal{A}_L$ a deterministic bad-prefix automaton of arity k_L. The problem of deciding whether for every representation of a bad-prefix of φ a permutation is accepted by $\mathcal{A}_L$ and otherwise providing a minimal counterexample is in time polynomial in $|\mathcal{A}_L|$, exponential in $|\psi|$, k_T, and doubly exponential in k_L.

Proof Let $\mathcal{A}_L = (Q_L, \Sigma^{k_L}, q_{0,L}, F_L, \delta_L)$ be a deterministic bad-prefix automaton. We denote by $\mathcal{S}_L = (Q_L^{\mathcal{S}}, \Sigma^{k_L}, q_{0,L}^{\mathcal{S}}, F_L^{\mathcal{S}}, \delta_L^{\mathcal{S}})$ the dual safety automaton for $\mathcal{A}_L$; $\mathcal{S}_L$ exists and is efficiently computable since $\mathcal{A}_L$ is a deterministic bad-prefix automaton: Merge all accepting states into one non accepting state with a self loop as its only outgoing transition and make all other states accepting. Let $\mathcal{N}_\varphi = (Q_\varphi, \Sigma^{k_T}, q_0, F_\varphi, \Delta_\varphi)$ be the fine, nondeterministic k-BPA for φ (Lemma 5.1).

Since $\mathcal{A}_L$ is not necessarily being permutation-complete, it is not sufficient to check whether one sequence $t \in (\Sigma^{k_L})^*$ that represents a bad-prefix is not accepted by $\mathcal{A}_L$. However, we have to check whether no k_L-representation of a bad-prefix is accepted by $\mathcal{A}_L$. In order to accomplish this goal, we first construct the following safety automaton $\mathcal{S}_L^\varsigma = (Q_L^\varsigma, q_{0,L}^\varsigma, \Sigma^{k_L}, \delta_L^\varsigma)$ that accepts a trace t if and only if all representations of all its subsets of $\mathsf{unzip}(t)$ are accepted by $\mathcal{S}_L$:

- $Q_L^\varsigma = \{q_{0,L}^\varsigma\} \cup \bigtimes_{i \in \{1,\dots,k_L!\}} Q_{L,i}^{\mathcal{S}}$ where each $Q_{L,i}^{\mathcal{S}}$ is a copy of $Q_L^{\mathcal{S}}$.
- Let $\varsigma_1, \dots, \varsigma_{k_L!} : \{1, \dots, k_L\} \to \{1, \dots, k_L\}$ be a collection of all k_L-permutations. The transition relation δ_L^ς allows the following transitions.
  - $q_{0,L}^\varsigma \xrightarrow{t} (q_1, \dots, q_{k_L!})$ iff $\forall i \in \{1, \dots, k_L!\}.\ q_{0,L}^{\mathcal{S}} \xrightarrow{\varsigma_i(t)} q_i$.
  - $(q_1, \dots, q_{k_L!}) \xrightarrow{t} (q_1', \dots, q_{k_L!}')$ iff $\forall i \in \{1, \dots, k_L!\}.\ q_i \xrightarrow{\varsigma_i(t)} q_i' \in \delta_L^{\mathcal{S}}$.

The size of $\mathcal{S}_L^\varsigma$ depends linearly on $\mathcal{S}_L$ and doubly exponential on k_L since constructing the product automaton of $\mathcal{A}$ and $\mathcal{A}'$ yields a result of size $|\mathcal{A}| \cdot |\mathcal{A}'|$. Thus in our case the iterative construction sums up to: $|S_L^\varsigma| = |\mathcal{S}_L|^{k_L!} + 1 \leq |\mathcal{S}_L|^{k_L^{k_L}} + 1 = |\mathcal{S}_L|^{2^{O(k_L \log(k_L))}}$.

Next, we construct a permutation-complete nondeterministic k_T bad-prefix automaton for φ: $\mathcal{N}_\varphi^\tau = (Q_\varphi^\tau, \Sigma^{k_T}, q_{0,\varphi}^\tau, F_\varphi^\tau, \Delta_\varphi^\tau)$ using $\mathcal{N}_\varphi$:

- Let $\tau_i : \{1, \dots k_T\} \to \{1, \dots, k_T\}$ be $k_T^{k_T}$ pairwise different functions. Define $\mathcal{N}_\varphi^{\tau_i} = (Q_{\varphi,i}, q_{0,\varphi}^i, \Sigma^{k_T}, F_{\varphi,i}, \Delta_{\varphi,i})$ where:
  - $Q_{\varphi,i}$ and $F_{\varphi,i}$ are equal to Q_φ and F_φ up to a labeling of every state by an index i
  - $\Delta_{\varphi,i}(q_i, a) = \{q_i' \mid q' \in \Delta_\varphi(q, \tau_i(a))$ and q_i, q_i' are the i-labeled copies of $q, q'\}$
- $Q_\varphi^\tau = q_{0,\varphi}^\tau \cup \bigcup_{i \in \{1,\dots,k_T^{k_T}\}} Q_{\varphi,i}$
- $F_\varphi^\tau = \bigcup_{i \in \{1,\dots,k_T^{k_T}\}} F_{\varphi,i}$
- Define Δ_φ^τ as follows:
  - $\Delta_\varphi^\tau(q_{0,\varphi}^\tau, \epsilon) = \{q_{0,\varphi}^{\tau_i} \mid 1 \leq i \leq k_T^{k_T}\}$
  - $\Delta_\varphi^\tau(q_\varphi^\tau, a) = \Delta_\varphi^{\tau_i}(q_\varphi^\tau, a)$ for $q_\varphi^\tau \in Q_{\varphi,i}$ and $a \in (\Sigma^{k_T})^*$

Every word accepted by $\mathcal{N}_\varphi^\tau$ is a k_T-representation of a bad-prefix of φ. The size of $\mathcal{N}_\varphi^\tau$ is exponential in k_T and $|\varphi|$.

At this point $\mathcal{S}_L^\varsigma$ and $\mathcal{N}_\varphi^\tau$ are of different arity. In order to compare their languages we construct $\mathsf{extend}_{k_T}(\mathcal{S}_L^\varsigma) = (Q_L^\varsigma, q_{0,L}^\varsigma, \Sigma^{k_T}, \delta_{L,ext}^\varsigma)$ by extending all words accepted by $\mathcal{S}_L^\varsigma$, i.e., $\delta_{L,ext}^\varsigma(q, s) = q'$ iff $\delta_{L,ext}^\varsigma(q, \mathsf{extend}_{k_T}(s)) = q'$ for $q, q' \in Q_L^\varsigma$ and $s \in \Sigma^{k_L}$.

Next, construct the product automaton $\mathcal{N}_\otimes$ of $\mathsf{extend}_{k_T}(\mathcal{S}_L^\varsigma)$ and $\mathcal{N}_\varphi^\tau$. Afterward an emptiness check on $\mathcal{N}_\otimes$ solves the above problem. This can is computed in time polynomial in the size of $\mathcal{N}_\otimes$.

In case a word w in $\mathcal{L}(\mathcal{N}_\otimes)$ exists, we found a counterexample since $\mathsf{unzip}(w)$ is bad-prefix and no permutation of w is accepted by $\mathcal{A}_L$. Given such a counterexample t of length n, we can reduce t to a counterexample of minimal length by iteratively asking membership queries for the prefixes of t this can be achieved using $\log(n)$ many membership-queries. The counterexample can be of length at most $l_{\max} = 2^{O(|\psi|)} \cdot 2^{O(k_T)} \cdot O(|\mathcal{A}_L|) \cdot 2^{2^{O(k_L \log(k_L))}}$. Thus, we perform at most $O(|\psi|) + O(k_T) + O(\log(|\mathcal{A}_L|)) + 2^{O(k_L \log(k_L))}$ many membership queries with sequences of length $l_{\max}$.

■

We can finally conclude this section and give an algorithm for answering equivalence queries with respect to being minimally adequate. The algorithm is given in Algorithm 5.2 and leads to the following theorem.

Algorithm 5.2 EQ

Input: Syntactically safe HyperLTL formula φ, DFA $\mathcal{A}_L$
Output: Counterexample if $\mathcal{A}$ is not a k-bad-prefix automaton for φ

1: **if not** $\forall x \in \mathcal{L}(\mathcal{A}_L).\ \mathsf{unzip}(x) \in \mathsf{BadPref}(\varphi)$ **then** ▷ Lemma 5.5
2: **return** a counterexample
3: **if not** $\forall T \in \mathsf{BadPref}(\varphi).\exists x \in \mathcal{L}(\mathcal{A}_L).\ \mathsf{unzip}(x) \leq T$ **then** ▷ Lemma 5.6
4: minimize the counterexample ▷ Lemma 5.7
5: **return** a counterexample
6: **if not** $\forall t \in \mathsf{BadPref}_{\overrightarrow{k_L}}(\varphi).\exists s \in \mathsf{Perm}(t).\ s \in \mathcal{L}(\mathcal{A}_L)$ **then** ▷ Lemma 5.8
7: **return** a counterexample
8: **accept**

Proposition 5.9
Let $\varphi = \forall\pi_1 \ldots \forall\pi_{k_T}.\ \psi$ be a safe HyperLTL formula and let $\mathcal{A}_L$ be a DFA with arity k_L. The problem of deciding whether $\mathcal{A}$ is a k_L-bad-prefix automaton for φ plus for every representation of a bad-prefix of φ a permutation is accepted by $\mathcal{A}_L$ and otherwise computing a minimal counterexample, is in time polynomial in $|A_L|$, exponential in $|\psi|$, and doubly exponential in k_T.

Using the proposed algorithms for answering queries and the algorithm L^*_{hyper} introduced in Chapter 4, we obtain an efficient algorithm for constructing k-BPAs:

Theorem 5.10
Learning a minmial, tight, permutation-complete, deterministic k-BPA $\mathcal{A}_G$ for a safe HyperLTL formula $\varphi = \forall\pi_1 \ldots \forall\pi_{k_T}.\ \psi$ is in time polynomial in $|A_G|$, exponential in $|\psi|$, and doubly exponential in k_T.

Proof Let $n = |\mathcal{A}_G|$. The size of the longest counterexample provided during execution of L^*_{Hyper} is bounded by n since all queries are answered by minimal counterexamples. Theorem 4.7 provides that the running time of L^*_{Hyper} is polynomial in n and exponential in k_G. Therefore, the number of queries posed by L^*_{Hyper} is polynomially bounded by n and exponentially bounded by k_G. All membership query contains at most k_G traces each of length at most $|Q|$. Thus, every membership query can be resolved in time polynomial in n and space polynomial in $|\psi|$ and k_T according to Theorem 5.3. All together the membership queries can be decided in polynomial time in $|\mathcal{A}_G|$, exponential time in k_G, and polynomial space in $|\psi|$), k_T.

Further, at most $|Q|-1$ equivalence queries are posed that each can be resolved in time polynomial in $|A_G|$, exponential in $|\psi|$, and doubly exponential in k_T.

■

Chapter 6

Learnability of Hyperproperties

In the following chapter, we will provide some basic results about the theoretical boundaries of learning traces- and hyperproperties. These results do not immediately affect the learning algorithm proposed in this thesis but rather affect possible future extensions of it.

6.1 Learning Hypersafety

First, we show that for the general class of k-safety hyperproperties, membership queries are undecidable in general. This result even holds for 1-safety hyperproperties, i.e., safety trace properties.

Theorem 6.1
Answering membership queries for safety properties is undecidable in general.

Proof Let $\Sigma = \{0, 1\}$ and let $\mathcal{L} \subseteq \Sigma^*$ be the binary encoding of the halting problem and let there be an enumeration $M_1, M_2, \dots$ of the language $\mathcal{L}$. Based on $\mathcal{L}$ we define the following undecidable safety language $\mathcal{L}'$:

$$\mathcal{L}' = \left\{ 0^{|M_i|} \$ M_i \$ \sigma \mid M_i \in \mathcal{L} \wedge \sigma \in \{0,1\}^\omega \right\} \cup \{0,1\}^\omega$$

We show that $\mathcal{L}'$ is a safety language, i.e., every trace not in $\mathcal{L}'$ has a finite bad-prefix. Therefore, let $\tau = \tau_1 \tau_2 \dots \in \{0, 1, \$\}^\omega$. We consider the following cases:

- $\tau \in \{0,1\}^\omega : \tau \in \mathcal{L}'$
- $\tau \in \{0,1,\$\}^\omega$ with $\#_\$(\tau) = 1$: Let $\tau = w\$\sigma \in \Sigma^* \cdot \{\$\} \cdot \Sigma^\omega$. The prefix of τ of length $2|w|+2$ is a bad-prefix since its last label was unequal to $\$$.
- $\tau \in \{0,1,\$\}^\omega$ with $\#_\$(\tau) > 2$: The minimal prefix of τ with $\#_\$ = 3$ is a bad-prefix.
- $\tau \in \{0,1,\$\}^\omega$ with $\#_\$(\tau) = 2$:

- $\tau \in \mathcal{L}' : \checkmark$
- $\tau \notin \mathcal{L}'$: Choose some m such that $\#_{\$}(\tau_1 \dots \tau_m) = 1$, $\tau_m = \$$ and n such that $\#_{\$}(\tau_{m+1} \dots \tau_{(m+1)+n}) = 1$, $\tau_{(m+1)+n} = \$$. If $m \neq n$, then $\tau_1 \dots \tau_{(m+1)+n}$ is a bad-prefix. If $m = n$, then $\tau_{m+1} \dots \tau_{(m+1)+n-1} \notin \mathcal{L}$ and thus $\tau_1 \dots \tau_{(m+1)+n}$ is a bad-prefix.

Thus, $\mathcal{L}'$ is a safety-language. $\mathcal{L}'$ is undecidable since $\mathcal{L}$ was chosen to be the binary encoding of the halting problem.

■

The following corollary immediately follows from Theorem 6.1 as the classes of 1-safety hyperproperties and safety trace properties coincide [10].

Corollary 6.2
Deciding membership queries for k-safety hyperproperties is undecidable in general.

6.2 Learning HyperLTL

Despite the undecidability result from Corollary 6.2, there are important classes of safety languages for which membership queries are decidable, for example, safety languages expressed in LTL. The problem can be solved by extending the LTL formula by the bad-prefix and checking the satisfiability of the new formula, which is decidable for the full class of LTL [30]. For the safety fragment of LTL, even equivalence queries are decidable. Given a bad-prefix automaton $\mathcal{A}$ and an LTL formula ψ. We can construct the canonical minimal, tight, deterministic bad-prefix automata $\mathcal{A}'$ and $\mathcal{A}_\varphi$ corresponding to $\mathcal{A}$ and the set of bad-prefixes of ψ, respectively. These two constructions can be computed efficiently according to Kupferman and Vardi [24].

Aiming to adopt the above approach to the context of hyperproperties, the following question arises: Does a HyperLTL formula φ expresses a k-safety hyperproperty? For LTL, the corresponding question—whether a formula expresses a safety property—is decidable due to a result by Maretic et al. showing that LTL admits a safety-liveness decomposition [27]. In addition, Clarkson and Schneider proved that such a separation into a pure hypersafety and a pure hyperliveness part exists for hyperproperties [10]. We reduce the satisfiability of HyperLTL to deciding whether a given HyperLTL formula describes a hypersafety property. Thus, according to Finkbeiner and Hahn's undecidability result for the satisfiability of HyperLTL, our problem is undecidable:

Theorem 6.3
The problem of checking whether a HyperLTL formula φ expresses a k-safety hyperproperty is undecidable in general.

Proof We reduce the satisfiability of a HyperLTL formula φ to the problem of deciding whether $\mathcal{L}(\varphi)$ is a k-safety hyperproperty. Since the satisfiability of HyperLTL was shown to be undecidable (Theorem 2.8), our claim follows. First, observe that every unsatisfiable HyperLTL formula describes the hyperproperty $\mathcal{L}(\varphi) = \emptyset$ and $\mathcal{L}(\varphi)$ is k-safety for all $k \in \mathbb{N}$. More precisely, **H** is 0-safety since its the set of bad-prefixes is the set of all finite sets of finite traces $\mathcal{P}(\Sigma^*)$ Thus, applying the definition of subset-closedness of hypersafety properties the claim follows for all k.

Given a HyperLTL formula $\varphi = Q_1\pi_1 \dots Q_n\pi_n.\ \psi$ over the alphabet Σ. We construct the formula $\varphi' = Q_1\pi_1 \dots Q_n\pi_n.\ \psi \wedge \Diamond a_{\pi_1}$ over the alphabet $\Sigma' = \Sigma \dot\cup \{a\}$, i.e., $a \notin \Sigma$. We claim that φ is unsatisfiable if and only if $\mathcal{L}(\varphi')$ expresses a k-safety hyperproperty. Therefore we distinguish the following two cases:

- φ is unsatisfiable:
  φ' is unsatisfiable as well. Thus, φ' is k-safety for any $k \in \mathbb{N}$.

- φ is satisfiable:
  Let $T \subseteq \Sigma^\omega$ be a set of traces such that $\emptyset \models_T \varphi$. Construct $T' \subseteq {\Sigma'}^\omega$ $T' = \{t' \mid \exists t \in T.\ t' \equiv_\Sigma t \wedge \forall i \in \mathbb{N}.\ t'[i] \models a\}$. Since $T \models \varphi$ it follows: $T' \models \varphi'$. Thus, the satisfiability of φ implies the satisfiability of φ'. Besides, $\mathcal{L}(\varphi')$ is not a k-safety hyperproperty because the extra conjunct enforces every or some trace, depending on Q_1, to satisfy a eventually, which is a liveness requirement.

■

The same proof shows that checking whether a formula describes a safety hyperproperty is undecidable.

Due to the work of Finkbeiner and Hahn, a classification of undecidable fragments follows from our reduction since it is independent of quantifier alternations. Besides, the decidability of whether a hyperproperty given in HyperLTL is safe, a general-purpose learning algorithm for hyperproperties requires the decidability of queries for formulas in HyperLTL. We can show that equivalence queries for hyperproperties expressed in HyperLTL are undecidable in general.

Proposition 6.4
Equivalence queries are undecidable for the full class of hyperproperties expressed in HyperLTL.

Proof Assume that there exists a teacher answering membership and equivalence queries for φ. Then, we can answer the satisfiability of φ by asking whether $\varphi \equiv \mathsf{False}$. This contradicts the decidability results for HyperLTL-SAT by Finkbeiner and Hahn [15].

■

In contrast to the negative decidability results, we give a characterization of an important class of HyperLTL formulas for which it is decidable whether their language is k-safe, according to Agrawal and Bonakdarpour.

Lemma 6.5 [1]
Let φ be a HyperLTL formula of the form $\varphi = \forall\pi_1 \ldots \forall\pi_k.\ \psi_1 \vee \cdots \vee \psi_n$, where $\psi_1, \ldots, \psi_n$ are safety formulas under LTL semantics. Then φ expresses a k-safety hyperproperty.

Since we can decide in LTL the expressed language is a safety language, it follows that we can decide for the above HyperLTL formulas whether they are k-safety. Further, it is important to note that the above classification is not tight, for example, $\forall\pi.\forall\pi'.\ a_\pi \wedge \Diamond a_{\pi'}$ is not in the above class, even tough it is k-safe.

We extend the above classification and give a decision procedure for the entire $\forall$-fragment of HyperLTL. Observe that being k-safety does not coincide with Finkbeiner et al.'s notion of monitorability for HyperLTL formulas in the $\forall$-fragment [17]: $\forall\pi.\forall\pi'.a_\pi \wedge \Diamond a_{\pi'}$ is monitorable but not k-safety.

Theorem 6.6
Let $\varphi = \forall\pi_1 \ldots \forall\pi_k.\ \psi$ be a HyperLTL formula. Whether or not $\mathcal{L}(\varphi)$ is a k-safety hyperproperty can be decided in space polynomial in $|\psi|$ and space exponential in k.

Proof First, note that for HyperLTL formulas in the $\forall$-fragment, the described hyperproperty is safe if and only if it is k-safe. Following the definition of k-safety hyperproperties, it holds that $\mathcal{L}(\varphi)$ is k-safe if and only if

$$\forall T.\ \Big[T \not\models \varphi \Rightarrow \Big[\exists T' \leq T.\ |T'| \leq k \wedge \forall \tilde{T'} \geq T'.\ \tilde{T'} \not\models \varphi\Big]\Big]$$

Our first claim is: φ is safe for all sets of size at most k, i.e., all sets T that we consider are of size at most k if and only if φ is safe.

($\Leftarrow$) Let T be some set of traces that violates φ. There exists a set $T' \leq T$ with $|T'| \leq k$ and thus φ must reject every set of traces extending T' including those of size k. Hence, φ is safe for traces of size at most k.

($\Rightarrow$) Let φ be k-safe for all sets of traces of size at most k. Then, a set of traces violating φ must have a bad-prefix T' of size at most k. Further, every set of traces extending T has this prefix T' and thus φ is safety.

Further, we claim that φ is safe for a set of traces T of size i if and only if the following formula is safe with respect to LTL semantics:

$$\psi_i(\pi_1, \ldots, \pi_i) := \bigwedge_{\varsigma:\{1,\ldots,k\}\to\{1,\ldots,i\}} \psi(\pi_{\varsigma(1)}, \ldots, \pi_{\varsigma(k)})$$

We rephrase the foregoing claim as follows for all $m \leq k$ and $T = \{t_1, \ldots, t_m\}$:

$$T \models \varphi \text{ if and only if } (t_1, \ldots, t_m) \models \psi_m$$

($\Rightarrow$) Following the semantic of HyperLTL $T \models \varphi$ implies: For all Π with $\mathsf{Traces}(\Pi) \subseteq T$. $\Pi \models_\emptyset \psi$. Thus, $(t_1, \ldots, t_m) \models \psi_m(\pi_1, \ldots, \pi_m)$.

($\Leftarrow$) $(t_1, \ldots, t_m) \models \psi_m$ implies that for all $\varsigma : \{1, \ldots, k\} \to \{1, \ldots, m\}$: $(t_{\varsigma(1)}, \ldots, t_{\varsigma(k)}) \models \psi(\pi_1, \ldots, \pi_k)$. Hence, for all trace assignments Π with $\mathsf{Traces}(\Pi) \subseteq T$. $\Pi \models_\emptyset \psi$. And thus by definition $T \models \varphi$.

Hence we can follow:

φ is safe of size m or upto m if and only if ψ_m is safe under the semantics of LTL

Deciding whether ψ_k is safe can be done is space polynomial in $|\psi_k|$ according to Maretic et al. [27]. The result coincides with whether φ is a k-safety hyperproperty.

■

Chapter 7

Conclusion

This thesis emerged from two partially related topics in formal methods. First, we investigated a canonical representation for the class of hyperproperties. Secondly, we extended the learning paradigm based around Angluin's L^* algorithm to hyperproperties.

For the purpose of a canonical representation of hyperproperties, we introduced k-safety and k-bad-prefix automata as a representation for regular k-safety hyperproperties in Chapter 3. We looked at different kinds of k-bad-prefix automata. Following the tradition of bad-prefix automata, we considered tight and fine automata as well as permutation-complete k-bad-prefix automata. Moreover, we proposed a construction algorithm for minmial, tight, permutation-complete, deterministic k-bad-prefix automata for a regular k-safety hyperproperty handed as an LTL formula or a nondeterministic k-safety automaton. Further, we defined an equivalence relation together with an efficient method for checking equivalences.

Thereafter, we extended Dana Angluin's L^* algorithm to learning k-bad-prefix automata in Chapter 4 starting from an unknown regular k-safety hyperproperty together with an unknown minimal $k \in \mathbb{N}$. The algorithm was accompanied by an example run of L^*_{Hyper} learning a k-bad-prefix automaton for the safe HyperLTL formula $\varphi = \forall\pi.\forall\pi'.\ a_\pi \wedge \square(a_\pi \leftrightarrow a_{\pi'})$. Afterward, we provided the termination, correctness, and runtime analysis of the proposed algorithm.

We extended the learning algorithm with respect to the teacher in Chapter 5 by providing algorithms that decide membership and equivalence queries. The teacher enables the entire framework to being able to understand specifications given in the safe fragment of HyperLTL.

In Chapter 6, we provided a summary of results regarding the general task of learning trace properties as well as hyperproperties. We proved that equivalence queries for hypersafety properties expressed in HyperLTL are in general undecidable and that distinguishing between safety and liveness of hyperproperties expressed in HyperLTL is undecidable as well. These results leave a gap for possible learning algorithms in this area.

7.1 Future Work

An important step in the future will be to implement the introduced learning framework L^*_{Hyper} together with the proposed algorithms to decide queries. This would allow for the study of minimal, permutation-complete k-bad-prefix automata in the field of research and compare the learning-based construction to the first automata-transformation-based construction for k-safety hyperproperties, which requires more space in theory. Besides, the application of k-bad-prefix automata to the field of runtime verification can be of interest. Of course, an extension of the teacher to larger fragments of regular k-safety hyperproperties is desired as well as an extension of the entire framework to larger classes of hyperproperties apart from regular k-safety hyperproperties.

Following the tradition of applications for learning algorithms, the proposed L^*_{Hyper} algorithm can be applied to the field of formal verification. Therefore, many applications of L^* can now be revisited and inspected with respect to k-safety hyperproperties, for example, the learning of assumptions for composed systems based on the assume-guarantee-paradigm to verify safety properties, which could be applied to regular k-safety hyperproperties [12].

We emphasized earlier that k-safety and k-bad-prefix automata are, to the best of our knowledge, the first step towards a canonical representation of hyperproperties. It will be of huge interest in the future to extend this step to more expressive fragments of hyperproperties.

Besides the decidability results which we provided in the foregoing chapter, there are many open problems in this area of research. To name only but a few:

- Are our algorithms, introduced in the context of k-safety hyperproperties, tight with respect to their running time complexity?
- How large is the set of HyperLTL formulas for which we can decide whether the described language is a k-safety hyperproperty?
- Do the questions of being k-safety and being satisfiable coincide for HyperLTL with respect to satisfiability?
- Can every k-safety hyperproperty expressible in HyperLTL be expressed in the $\forall$-fragment of HyperLTL? We strongly believe this statement holds.

List of Definitions

References

[1] 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*. 2016, pp. 239–252.

[2] Rajeev Alur, Pavol Černý, and Steve Zdancewic. "Preserving Secrecy Under Refinement." In: *Automata, Languages and Programming*. Springer Berlin Heidelberg, 2006, pp. 107–118.

[3] Dana Angluin. "Learning Regular Sets from Queries and Counterexamples." In: *Inf. Comput.* 75.2 (1987), pp. 87–106.

[4] Dana Angluin. "Negative results for equivalence queries." In: *Machine Learning* 5.2 (June 1990), pp. 121–150.

[5] Dana Angluin. "Queries and Concept Learning." In: *Machine Learning* 2.4 (1987), pp. 319–342.

[6] Dana Angluin, Timos Antonopoulos, and Dana Fisman. "Query Learning of Derived Omega-Tree Languages in Polynomial Time." In: *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*. 2017, 10:1–10:21.

[7] Dana Angluin and Dana Fisman. "Learning Regular Omega Languages." In: *Algorithmic Learning Theory - 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*. 2014, pp. 125–139.

[8] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[9] Armin Biere et al. "Theory and Practice of SAT Solving (Dagstuhl Seminar 15171)." In: *Dagstuhl Reports* 5.4 (2015), pp. 98–122.

[10] Michael R. Clarkson and Fred B. Schneider. "Hyperproperties." In: *Journal of Computer Security* 18.6 (2010), pp. 1157–1210.

[11] Michael R. Clarkson et al. "Temporal Logics for Hyperproperties." In: *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*. 2014, pp. 265–284.

[12] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. "Learning Assumptions for Compositional Verification." In: *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*. 2003, pp. 331–346.

[13] Deepak D'Souza et al. "Model-checking Trace-based Information Flow Properties." In: *J. Comput. Secur.* 19.1 (Jan. 2011), pp. 101–138.

[14] Samuel Drews and Loris D'Antoni. "Learning Symbolic Automata." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 173–189.

[15] Bernd Finkbeiner and Christopher Hahn. "Deciding Hyperproperties." In: *27th International Conference on Concurrency Theory (CONCUR 2016)*. Vol. 59. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016, 13:1–13:14.

[16] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. "Algorithms for Model Checking HyperLTL and HyperCTL*." In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*. 2015, pp. 30–48.

[17] Bernd Finkbeiner et al. "Monitoring Hyperproperties." In: *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*. 2017, pp. 190–207.

[18] Bernd Finkbeiner et al. "Synthesizing Reactive Systems from Hyperproperties." In: *CAV (1)*. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 289–306.

[19] Paul Gastin and Denis Oddoux. "Fast LTL to Büchi Automata Translation." In: *Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 53–65.

[20] Joseph A. Goguen and José Meseguer. "Security Policies and Security Models." In: *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*. 1982, pp. 11–20.

[21] John Hopcroft. "An n log n algorithm for minimizing states in a finite automaton." In: *Theory of machines and computations*. Elsevier, 1971, pp. 189–196.

[22] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[23] Orna Kupferman and Robby Lampert. "On the Construction of Fine Automata for Safety Properties." In: *Automated Technology for Verification and Analysis, 4th International Symposium, ATVA 2006, Beijing, China, October 23-26, 2006.* 2006, pp. 110–124.

[24] Orna Kupferman and Moshe Y. Vardi. "Model Checking of Safety Properties." In: *Formal Methods in System Design* 19.3 (2001), pp. 291–314.

[25] Yong Li et al. "A Novel Learning Algorithm for Büchi Automata Based on Family of DFAs and Classification Trees." In: *TACAS (1)*. Vol. 10205. Lecture Notes in Computer Science. 2017, pp. 208–226.

[26] Oded Maler and Amir Pnueli. "On the Learnability of Infinitary Regular Sets." In: *Inf. Comput.* 118.2 (1995), pp. 316–326.

[27] Grgur Petric Maretic, Mohammad Torabi Dashti, and David A. Basin. "LTL is closed under topological closure." In: *Inf. Process. Lett.* 114.8 (2014), pp. 408–413.

[28] Luan Viet Nguyen et al. "Hyperproperties of Real-valued Signals." In: *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*. MEMOCODE '17. Vienna, Austria: ACM, 2017, pp. 104–113.

[29] Srinivas Pinisetty, Gerardo Schneider, and David Sands. "Runtime Verification of Hyperproperties for Deterministic Programs." In: *Proceedings of the 6th Conference on Formal Methods in Software Engineering*. FormaliSE '18. Gothenburg, Sweden: ACM, 2018, pp. 20–29.

[30] Kristin Y. Rozier and Moshe Y. Vardi. "LTL satisfiability checking." In: *STTT* 12.2 (2010), pp. 123–137.

[31] Claude Sammut and Ranan B. "Learning concepts by asking questions." In: *In*. Morgan Kaufmann, 1986, pp. 167–192.

[32] Fred B. Schneider. "Enforceable Security Policies." In: *ACM Trans. Inf. Syst. Secur.* 3.1 (Feb. 2000), pp. 30–50.

[33] Moshe Y. Vardi. "An automata-theoretic approach to linear temporal logic." In: *Logics for Concurrency: Structure versus Automata*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 238–266.

[34] Moshe Y. Vardi. "Verification of Concurrent Programs: The Automata-Theoretic Framework." In: *Annals of Pure and Applied Logic*. 1987, pp. 167–176.

[35] Hirotoshi Yasuoka and Tachio Terauchi. "Quantitative information flow as safety and liveness hyperproperties." In: *Theor. Comput. Sci.* 538 (2014), pp. 167–182.

[36] Steve Zdancewic and Andrew C. Myers. "Observational Determinism for Concurrent Program Security." In: *In Proc. 16th IEEE Computer Security Foundations Workshop*. 2003, pp. 29–43.