SAARLAND UNIVERSITY
FACULTY OF NATURAL SCIENCES AND TECHNOLOGY I

BACHELOR'S THESIS

DECIDING HYPERLTL

Author:
Christopher Hahn

Advisor:
Prof. Bernd Finkbeiner, Ph.D.

Reviewers:
Prof. Bernd Finkbeiner, Ph.D.
Prof. Dr. Gert Smolka

Submitted: 11th February 2016

Revised Version

Saarbrücken, 11th February, 2016

Abstract

Information leakage of a system is an undesirable behaviour, since sensitive data might be open to the public. Building systems that satisfy certain secrecy constraints is a challenging task. To this end, one must be able to write formal specifications of information flow policies. Hyperproperties are a concept to express such policies, e.g. observational determinism.

Trace properties, which are a set of possible execution traces, can be formalized in linear temporal logic (LTL). Hyperproperties are a set of trace properties and, hence, establish relations between possible execution traces of a system. HyperLTL is a recently introduced extension of LTL for expressing such hyperproperties in a consistent and compact manner.

We consider two satisfiability problems, where the empty model is included and excluded respectively. This thesis solves these and determines every decidable fragment of HyperLTL. We give an algorithm for checking equivalence of formulae from the rich fragment of universally quantified HyperLTL. We also present the minimal fragments of HyperLTL for which these satisfiability problems are undecidable.

To conclude this thesis, a prototype satisfiability checker is presented and evaluated, which solves the satisfiability problem for every decidable fragment of HyperLTL and can furthermore solve the question whether two universally quantified HyperLTL formulae are equivalent.

Acknowledgements

My deepest gratitude goes to my advisor Prof. Bernd Finkbeiner, Ph.D. for guiding me through the journey of writing this thesis. I am very thankful for his supervision and the uncountable meetings that not only pushed me forwards in the topic and as a computer scientist but also shaped me as a person. His support and guidance mean a lot to me.

I thank Prof. Dr. Gert Smolka for reviewing this thesis and also for designing the lectures “Programming 1” and “Introduction to Computational Logic”, which introduced me to the magic of computer science.

I would like to thank my parents, family and friends. To name but a few: Kathrin, Clara and Yannick for their guidance through my first semesters. Fabian for playing domino with me. Hazem for introducing me to the topic. Norine for being Hyper. Jana for believing in me. Edgar for proofreading this thesis. Jens, Joris, Jan, Thomas and everyone I worked with over the last two years for the countless hours of incredibly fruitful discussions and coffee breaks.

In particular, I thank my dearest friend André and my love Vivien for always being by my side through rough as well as great times.

Contents

Chapter 1

Introduction

In the field of modern technology, the role of computers has changed dramatically over the last few years. If one thinks of a program, one usually means a piece of code that gets an input and outputs the result of a computation. For example the following **sequential program** computes the factorial of a natural number.

```
fun factorial n = if n = 0
                  then 1
                  else n * factorial(n−1)
```

The factorial program can be described as a set, putting its input and output into relation, a so-called **input-output relation**.

$$\{(0,1),(1,1),(2,2),(3,6),(4,24),(5,120),\ldots\}$$

But today, computing systems are embedded in a lot of everyday objects like medical technology or even simple household items. In contrast, such a system reacts to certain events and provides computational results based on its **environment**. An example of such a **reactive system** is a coffee machine, shown in Figure 1.1. We model the coffee machine as a system that "runs forever" in the sense that there exists no input-output relation but only communication with its environment.

Grasping a reactive system is more complicated. We will model it as its set of possible **infinite execution traces**. Even if the coffee machine only runs for months, or even hours, assuming that a run through such a system is infinite is robust. It abstracts away from details like time of usage and provides an uniform model. Such

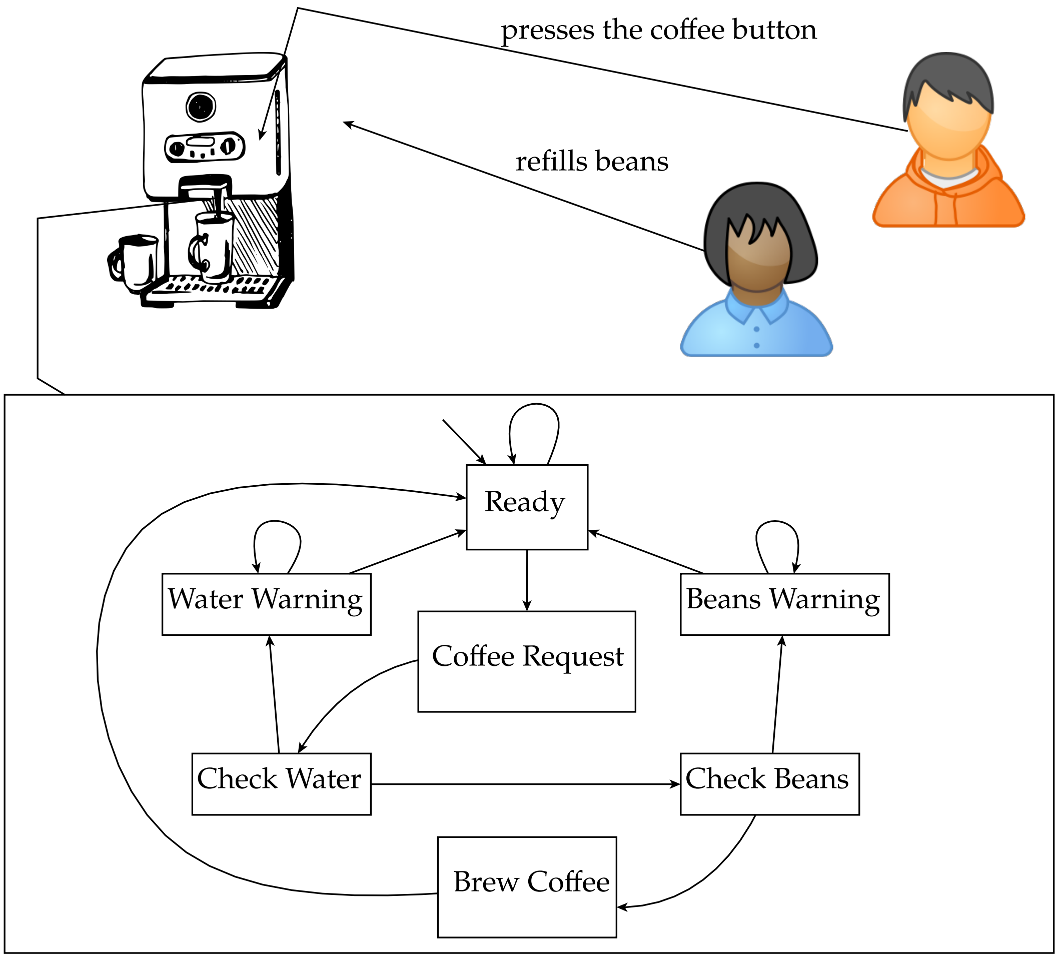

Figure 1.1: Sketch of a Coffee Machine as a Reactive System

an infinite execution trace is represented as an infinite sequence of sets of possible events, called **atomic propositions**. For example, consider the following sketch of an infinite computation trace.

$$\{Ready\}\{CoffeeRequest\}\{CheckWater\}\{CheckBeans\}\{BeansWarning\}\{Ready\}\dots$$

This trace represents that a person demands a coffee after the coffee machine idles for a while. But since there are not enough beans in the machine, it only goes back to "Ready" mode when more beans are provided by the environment, denoted by a self-loop on the "Warning" states. After that a coffee can successfully be brewed.

Providing formal methods for reasoning on such reactive systems is a complex but important field of modern computer science. The well-studied fields of model

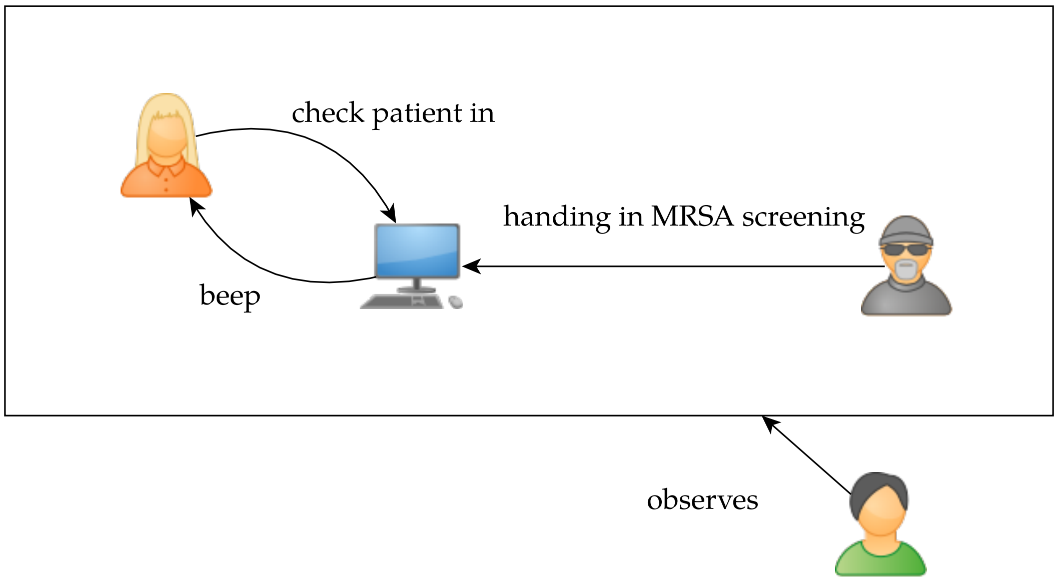

Figure 1.2: Sketch of a Hospital Check-in with an Observing Visitor

checking [14] and synthesis [11] of reactive systems provide solutions for this task.

Scope of this thesis. Besides the desire for building correct software and systems, the need for privacy and security is moving more and more to the foreground. Avoiding that unauthorized parties might infer private data out of observable changes in the system is highly desirable and will be the scope of this thesis.

It might seem acceptable that an observer sees what kind of coffee one prefers. But safety-critical systems, which contain sensitive user data, surround us every day. Consider for example a hospital management system that manages patient check-ins and -outs. Some hospitals require a so-called "Methicillin Resistant Staphylococcus Aureus" (MRSA) [4] screening to determine if a patient has a highly contagious disease and, therefore, to prevent an outbreak.

Imagine that a patient needs to hand in the MRSA screening while checking in. The process is sketched in Figure 1.2. To remind the staff of necessary precautions, the system outputs a "beep" tone and displays a warning every time a patient with a positive screening result tries to check in. While the warning is only visible to the staff, the "beep" tone is an output which is observable by anyone. This behaviour of the system is represented in the extract of the following traces. They represent check-ins of a contagious and a non contagious patient respectively.

$$\ldots\{\neg \mathit{checksIn}\}\{\mathit{checksIn}, \mathit{contagious}\}\{\mathit{beep}, \mathit{checkedIn}\}\{\neg \mathit{checksIn}\}\{\neg \mathit{checksIn}\}\ldots$$
$$\ldots\{\neg \mathit{checksIn}\}\{\mathit{checksIn}, \neg \mathit{contagious}\}\{\mathit{checkedIn}\}\{\neg \mathit{checksIn}\}\{\neg \mathit{checksIn}\}\ldots$$

Therefore, a visitor that observes multiple check-ins might be able to realize a relation between the "beep" tone and the patients status. By simply paying attention to the check-in process, the visitor can infer the patients status. This is an undesirable behaviour of the system and could be solved by omitting the "beep" tone. Since the "beep" tone is a more powerful reminder, one might have to accept a trade off between certain functionality and information leakage.

To protect the privacy of the patients, the hospital management system needs to ensure that a visitor is unable to infer such highly private data. Finding a suited formalism to express such a secrecy constraint is a hard task. Intuitively such a formalism must be able to establish a relation between different computation traces of the system.

Whereas formal verification of reactive systems is already a hard task, checking a system against such secrecy constraints is especially complicated. Unfortunately, linear temporal logic (LTL) [29] and its related logics (CTL [13] and CTL* [19]) are not able to formalize important security policies. For example, so-called "**observational determinism**" [32] [28] states that a reactive systems behaves deterministically for a low-security observer. Hence, high-security changes in the system cannot be observed and sensitive data cannot be inferred. For formalizing this property a logic must enable the possibility of setting certain traces into relation.

Related work. There are a lot of approaches in recent research to formalize such security policies. In 2004, Barthe et. al. [8] used the concept of self-composition to verify noninterference properties [23], which later was generalized by Terauchi and Aiken [41] in 2005 to express any property that ranges over a pair of execution traces. Dimitrova, Finkbeiner et. al. presented SecLTL [17], which extends LTL with a so-called "Hide Operator" lifting the expressiveness of security policies. Unfortunately it is not expressive enough to formalize every security policy. Andersen presented an extension to the modal μ-calculus [25], called polyadic μ-calculus [5]. Furthermore, Balliu et. al. [7] present a formalism to express information flow policies in the framework of so-called epistemic logic [20], with the usage of the so-called "Knowledge Operator".

All of the above approaches suffer from severe drawbacks, either the lack in expressiveness or unintuitive use. In 2008, Clarkson and Schneider [15] presented the concept of hyperproperties and classified different security policies. Trace properties, which can be expressed with LTL for example, are sets of traces. In contrast, a hyperproperty is a set of trace properties.

Resolution. In 2014 Clarkson, Finkbeiner et. al. [16] presented two intuitive and expressive logics for formalizing hyperproperties, called HyperLTL and HyperCTL*. They showed that the logics subsume a lot of the approaches mentioned above. The idea is to use either universal or existential quantification over trace variables to express the desired relations between different computation traces. For example, we can now formalize that the hospital management system does not leak any information while checking in patients. $\square$ is the common operator of LTL. $\square\,\phi$ states that the subformula ϕ holds **globally**, i.e. at any position of the trace. The atomic proposition $\mathit{checksIn}$ represents the input that a patient checks in and beep represents the observable "beep" tone of the reactive system.

$$\forall\pi\forall\pi'.(\mathit{checksIn}_\pi \leftrightarrow \mathit{checksIn}_{\pi'}) \rightarrow \square(\mathit{beep}_\pi \leftrightarrow \mathit{beep}_{\pi'})$$

The formula states that all pairs of computation traces have the exact same behaviour for the outgoing "beep" tone, if there is a check-in at the first position. Basically stating that either the "beep" tone is audible after all check-ins or never audible at all, which makes it impossible to infer a patients status the way we described it above. Note that the internal, not observable, behaviour is still allowed to vary. This finally provides an intuitive formalism for the expression of secrecy policies. Hence, we can express so-called **observational determinism** formally, for a set of observable inputs I and a set of observable outputs O.

$$\forall\pi\forall\pi'.(\bigwedge_{in\in I} in_\pi \leftrightarrow in_{\pi'}) \rightarrow \square(\bigwedge_{out\in L} out_\pi \leftrightarrow out_{\pi'})$$

The formula states that if the observable inputs are the same at the first position in a pair of computation traces, then the output remains the same in this pair.

Finkbeiner et. al. [21] already examined the model checking problem for HyperCTL*, which is the question whether a system model satisfies a property given as a HyperLTL formula. They presented an algorithm and complexity results for that task.

Contribution. To contribute to this topic we consider the satisfiability and satisfiability$^+$ problem (excluding the empty model) of HyperLTL in detail. For example, consider the following HyperLTL formula.

$$\exists\pi\exists\pi'.(\square\, a_\pi) \wedge (\square\neg a_{\pi'})$$

This formula states that there exists a trace π where the atomic proposition a does hold on every position, but there also exists a trace π' where the atomic proposition

a **never** holds. A single execution trace is not able to satisfy this property, which is why we are interested in the question if there exists a **set** of above mentioned execution traces that satisfies the given HyperLTL formula. A possible trace set satisfying the HyperLTL formula is the following, where sets of the first trace only contain the atomic proposition a and the sets of the second trace only contain b's.

$$\begin{aligned} \{\ &\{a\}\{a\}\{a\}\dots, \\ &\{b\}\{b\}\{b\}\dots\ \} \end{aligned}$$

Since a does **not** hold on any position of the second trace, but on every position of the first trace, this trace set does satisfy the given HyperLTL formula. The witness for satisfiability of a HyperLTL formula is therefore a **trace set**. This is a contrast to the satisfiability problem of an LTL formula, since the latter only asks for a single execution trace as a witness.

Checking formalized information flow policies for satisfiability is highly desirable, to determine whether a system would be even capable of satisfying the HyperLTL formula. Writing compact formulae is valuable, as a lot of verification algorithms scale in the size of the formula. Being able to check whether a HyperLTL formulae is equivalent to another one is therefore crucial for finding minimized formalizations of desired information flow policies. We will show that we are in fact capable of deciding this question for the rich fragment of universally quantified HyperLTL. Furthermore, the satisfiability problem of this logic has a close connection to synthesis of reactive systems. A possible solution already provides a model of the system in form of a trace set satisfying the given formula. This relation to synthesis makes the satisfiability problem of HyperLTL and HyperCTL* a highly interesting one for current research.

Structure of this thesis (see Figure 1.3). We provide the necessary preliminaries on decidabilty and complexity theory in Chapter 2 and the preliminaries on trace and hyperproperties as well as the considered logics in Chapter 3. We determine every fragment for which the satisfiability and satisfiability$^+$ problem is decidable and provide interesting complexity results in Chapter 4 and 5. This combines the expressivness of HyperLTL with the decidability of LTL. As an application of the provided results we present an algorithm for checking whether two universally quantified HyperLTL formulae are equivalent and determine its complexity in Section 5.4. As an addition, we define a bounded version of the satisfiability problem for a HyperLTL formula with one quantifier alternation, starting with an existential quantifier, and determine its complexity in Section 5.3. Analogously we define a bounded version of the equivalence check for universally quantified HyperLTL formulae and determine its complexity in Section 5.5. We also identify the minimal fragments of HyperLTL for which the satisfiability and satisfiability$^+$ prob-

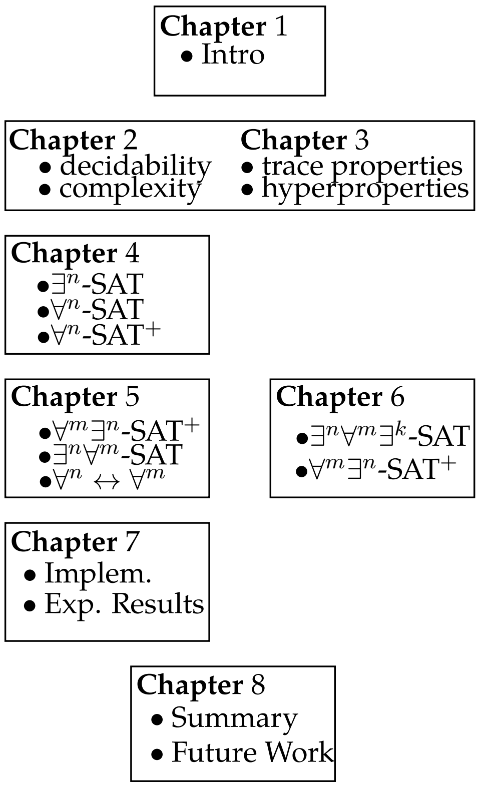

Figure 1.3: Structure of the Main Topics in this Thesis

lem remains undecidable in Chapter 6. We prove undecidability of the considered fragments via a many-one reduction of Post's correspondence problem. The presented results are implemented in the prototype **EAHyper** (Equivalence Checker for Universally Quantified HyperLTL Formulae), which is presented, including experimental results, in Chapter 7. The results of this thesis as well as a presentation of promising future work are summed up in Chapter 8. The basic structure of this thesis is displayed in Figure 1.3, where, for example, $\exists^n\forall^m$ denotes that there is only one quantifier alternation in the formula allowed, but at most n existential and m universal quantifiers.

Chapter 2

Preliminaries on Decidability & Complexity Theory

Kurt Gödel in 1931 [22] shattered the world of mathematics by stating that certain true statements about natural numbers are not provable. Computability theory arose in the 1930's with the groundbreaking work of Alonzo Church and Alan Turing. Independently of each other both of them developed a formalism for computability.

Church introduced the so called **lambda calculus** [10], while Turing introduced **Turing machines** [42]. As it turned out, both models of computation are in fact equivalent [42].

In this chapter we will define the basics of decidability and complexity theory and state the basic theorems that we will need in later chapters to reason about the decidability and complexity of certain problems.

2.1 Decidability Theory

Decidability has a close relation to computability, since one needs to answer the decision problem whether an element is in a given set or not. For this a computable function is needed, otherwise one cannot be sure that the decision procedure always terminates.

Definition 2.1 (Computable Function) A total function f is called **computable** if one can formalize the function in a basic model of computation, e.g. in the lambda calculus or as a Turing machine. □

Definition 2.2 (Decidability) A set X is **decidable** if there exists a computable function f, s.t. $f(x)$ returns true if $x \in X$ and false if $x \notin X$. □

Definition 2.3 (Undecidability) A set X is **undecidable** if there exists no computable function f, s.t. $f(x)$ returns true if $x \in X$ and false if $x \notin X$. □

We will often refer to the question of the decidability of a set as a **decision problem**.

Definition 2.4 (Turing Machine) A Turing machine M is defined as a 7-tuple $(Q, \Gamma, b, \Sigma, \delta, q_0, F)$.

- Q is a finite, non-empty set of states
- Γ is a finite, non-empty set of tape symbols
- $b \in \Gamma$ is the blank symbol (the only symbol allowed to occur on the tape infinitely often at any step during the computation)
- $\Sigma \subseteq \Gamma \backslash \{b\}$ is the set of input symbols
- $\delta : (Q \backslash F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a partial function called the **transition function**, where L is left shift, R is right shift. If δ is not defined on the current state and the current tape symbol, then the machine halts.
- $q_0 \in Q$ is the inital state
- $F \subseteq Q$ is the set of accepting states.

The initial tape content is said to be accepted by M if M eventually halts in a state $f \in F$.

One can define a nondeterministic Turing machine N analogously, where δ is a **transition relation** instead of a function. $\square$

We will usually abstract away from an explicit model of computation, but we will introduce exact definitions if needed. The interested reader may be referred to [26].

2.2 Many-one Reductions

The concept of many-one reductions was introduced by Emil L. Post in his work "Recursively enumerable sets of positive integers and their decision problems" [31]. It is based on the idea that one needs to construct a total computable function that transforms an instance of a decision problem P into another instance of a decision Problem P'.

Definition 2.5 (Many-one Reduction) Given two decision problems P with alphabet Σ and P' with alphabet Σ', we call a total computable function $f : \Sigma^* \to \Sigma'^*$ a **many-one reduction** iff $\forall p . p \in P \leftrightarrow f(p) \in P'$. We write $P \leq P'$ and say "P is reducible to P'". $\square$

Many-one reductions are a common tool in computability and decidability theory, since one can extract crucial information from reductions. We will exploit such properties in later sections, which is why we will prove them first.

Theorem 2.6 (Decidability) *Given two decision problems P and P', if P' is decidable and there exists a many-one reduction f, s.t. $P \leq P'$, then P is decidable too.*

Proof Let P and P' be arbitrary. Assume that P' is decidable and there exists a many-one reduction f, s.t. $P \leq P'$. Since there exists a total computable function, namely f, that can transform every instance of P to an instance of P', P has to be decidable too. ■

Theorem 2.7 (Undecidability) *Given two decision problems P and P', if P is undecidable and there exists a many-one reduction f, s.t. $P \leq P'$, then P' is undecidable too.*

Proof This theorem is a contraposition of the theorem above and therefore holds as well. ■

2.3 Complexity Theory

Complexity theory focuses on classifying computational problems according to their difficulty. Before applying an algorithm or a decision procedure it is interesting how long the computation will last. Especially, it is important to know if a problem has a **feasible** solution that scales.

We define the time a computation needs as the **worst-case** behaviour of a Turing machine on inputs of length n.

Definition 2.8 (Time Complexity) Let $n \in \mathbb{N}$ and a Turing machine M with an input x be given. Assume M halts on x. We denote the number of computation steps that M performs by $\text{Time}_M(x)$. The **time complexity** of M is defined as:

$$\text{Time}_M(n) = max\{\text{Time}_M(x) \mid |x| = n\} \qquad \square$$

Definition 2.9 (Computable Function in Time *t*) We say that a function f is deterministically computable in $t(n)$ iff there is a deterministic Turing machine M that computes f and $\text{Time}_M(n) \leq t(n)$ for all n. □

Definition 2.10 (DTime) Let t be a function.

$$\text{DTime}(t) = \{P \mid P \text{ is deterministically t time decidable}\}$$

For a set T of functions it is defined analagously.

$$\text{DTime}(T) = \bigcup_{t \in T} \text{DTime}(t) \qquad \square$$

Definition 2.11 (Space Complexity) Let $n \in \mathbb{N}$ and a Turing machine M with an input x be given. We denote the maximum of the space consumption of all computation steps of M by $\text{Space}_M(x)$. The **space complexity** of M is defined as:

$$\text{Space}_M(n) = max\{\text{Space}_M(x) \mid |x| = n\} \qquad \square$$

Definition 2.12 (Computable Function in Space *s*) We say that a function f is deterministically computable if there exists a Turing machine M that computes f and $\text{Space}_M(n) \leq s(n)$ for all n. $\square$

Definition 2.13 (DSpace) Let s be a function.

$$\text{DSpace}(s) = \{P \mid P \text{ is deterministically s space decidable}\}$$

For a set T of functions it is defined analagously.

$$\text{DSpace}(S) = \bigcup_{s \in S} \text{DSpace}(s) \qquad \square$$

NTime and **NSpace** are defined analogously for nondeterministic Turing machines.

We can now define the interesting complexity classes, which group problems of the approximately "same" difficulty together.

Definition 2.14 (P and NP)

$$\text{P} = \bigcup_{i \in \mathbb{N}} \text{DTime}(O(n^i))$$
$$\text{NP} = \bigcup_{i \in \mathbb{N}} \text{NTime}(O(n^i)) \qquad \square$$

Definition 2.15 (PSpace)

$$\text{PSpace} = \bigcup_{i \in \mathbb{N}} \text{DSpace}(O(n^i)) \qquad \square$$

Note that **PSpace** is equal to **NPSpace**, as was proved by Walter Savitch in 1970 [34].

Definition 2.16 (EXP)

$$\text{EXP} = \bigcup_{i \in \mathbb{N}} \text{DTime}(O(2^{n^i})) \qquad \square$$

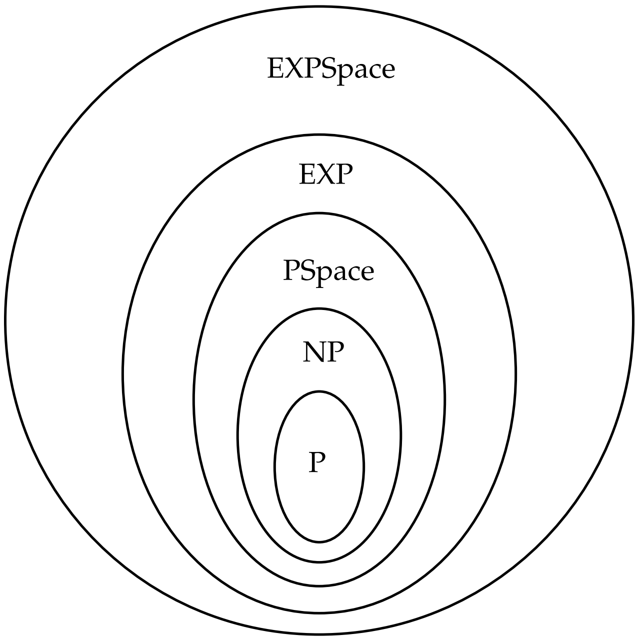

Figure 2.1: Hierarchy of Important Complexity Classes

Definition 2.17 (EXPSpace)

$$\text{EXPSpace} = \bigcup_{i \in \mathbb{N}} \text{DSpace}(O(2^{n^i})) \qquad \square$$

Again by Savitch's theorem [34] it follows that **EXPSpace** is equal to **NEXPSpace**. Figure 2.1 displays the complexity classes and their believed hierarchy. It is still an open problem to determine which of them are in fact equal. For the scope of this thesis it is sufficient to get a basic intuition of the most important complexity classes. In the next section, we will consider the important concept of **many-one polynomial time reductions** that allow us, similar to plain many-one reductions, to exploit already known complexity results to classify not-yet-examined problems.

2.4 Many-one Polynomial Time Reductions

Many-one polynomial time reductions were first used by Richard M. Karp in his famous 21 problems [24]. The basic idea is to construct a function that translates a problem instance into another one. By doing so, one can obtain strong properties from the reduced problem. We will first define those reductions formally before proving the theorems we need for this thesis.

Definition 2.18 (Many-one Polynomial Time Reductions) Let $P, P' \subseteq \Sigma^*$ for a given Σ. A function $f : \Sigma^* \to \Sigma^*$ is called a **many-one polynomial time reduction** from P to P' iff f is polynomial time computable and for all $x \in \Sigma^*$:

$$x \in P \leftrightarrow f(x) \in P' \qquad \square$$

It is important that the reduction is now required to preserve polynomial time computability.

Definition 2.19 (Reducible) P is (many-one) polynomial time reducible to P' iff there exists a many-one polynomial time reduction from P to P'. We denote this by $P \leq_P P'$. □

Definition 2.20 (Hardness of a Problem) A problem P is called **hard** for the presented complexity classes in Figure 2.1 C excluding the class P if

$$\forall \tilde{P} \in C. \tilde{P} \leq_P P$$

We say a problem P is **C-hard**.

Note that for a useful definition of hardness for the complexity class P one should use **LOGSpace** reductions [6], which we do not need in the scope of this thesis and therefore omit here. □

Intuitively **hardness** states that if one can solve a problem being hard for the complexity class C, then any other problem of C can be solved in the given time or space bound. Therefore those problems are the "hardest" ones in C.

Theorem 2.21 *Given a problem P, which is hard for a complexity class C, if P is many-one polynomial time reducible to a problem P', P' is hard for C too.*

Proof Given a problem P, which is hard for a complexity class C and a problem P', s.t. $P \leq_P P'$. Since P is hard for C, we can also translate every problem in C in polynomial time to P. We can translate by definition of $\leq_P$ every instance of P in polynomial time to P'. This means we can translate by Definition 2.20 every problem in C to P' in polynomial time. Therefore P' must be hard for C. ■

Definition 2.22 (Completeness) A problem P is called **complete** for a complexity class C iff

1. $P \in C$
2. P is **C-hard**.

We say a problem P is **C-complete**. □

Chapter 3

Trace Properties vs. Hyperproperties

We begin this section with some basic definitions. In a short introduction to LTL, we distinguish **trace properties** and **hyperproperties** [15], describe why LTL cannot express hyperproperties and state why being able to express such properties is desirable. We will then define a logic recently introduced by M.R. Clarkson, B. Finkbeiner et al. [16], called **HyperLTL**, that can express hyperproperties.

3.1 Trace Properties

We represent a reactive system as the set of all possible infinite execution traces. We refer to such a trace set as the **model** of a reactive system. An LTL formula defines a set of infinite traces through a reactive system (see Figure 3.1).

Definition 3.1 (Trace) Let AP be a set of atomic propositions. An infinite trace t is an element of the set $(2^{AP})^\omega$, which is the set of all possible infinite traces and denoted by TR. $\square$

Definition 3.2 (Trace property) A set $T \subseteq TR$ is called a trace property. $\square$

Although we usually model a reactive system as its set of execution traces, it is sometimes more convenient to model it as a state machine, for example a **Kripke structure**. A trace can be seen as a possible run through this Kripke structure.

Definition 3.3 (Kripke structure) A Kripke structure K is a tuple (S, s_0, δ, AP, L) containing a set of states S, an initial state $s_0 \in S$, a transition function $\delta : S \to 2^S$, a set of atomic propositions AP and a labelling function $L : S \to 2^{AP}$. $\square$

For manipulating traces, we will define some useful notation. Let $t \in TR$ be a trace and $i \in \mathbb{N}$ a natural number. $t[i]$ denotes the i-th element of t. Therefore $t[0]$ represents the starting element of the trace. Let $j \in \mathbb{N}$ and $j \geq i$. $t[i, j]$ denotes the sequence $t[i]\ t[i+1]\ ...\ t[j-1]\ t[j]$. $t[i,\infty]$ denotes the infinite suffix of t starting at position i.

LTL enables reasoning on reactive systems by allowing temporal operators ($\bigcirc, \mathcal{U}$) alongside the usual logical connectors ($\neg$, $\vee$). The formal syntactic definition is given by the following grammar.

Definition 3.4 (LTL, Syntax [29]) Let p be an atomic proposition.

$$\psi, \psi' \ ::= \ p \ | \ \neg\psi \ | \ \psi \vee \psi' \ | \ \bigcirc\ \psi \ | \ \psi\ \mathcal{U}\, \psi' \qquad \square$$

Intuitively $\bigcirc\psi$ means that ψ must hold in the **next** step of the trace through the system, whereas $\psi\,\mathcal{U}\,\psi'$ states that ψ holds **until** ψ' holds. There are several derived operators, where the most important ones are the following two.

$$\begin{aligned} \Diamond\ \psi \ &\equiv\ true\ \mathcal{U}\ \psi \\ \Box\ \psi \ &\equiv\ \neg\ \Diamond\ \neg\ \psi \end{aligned}$$

Intuitively $\Diamond\psi$ states that ψ will **eventually** hold in the future and $\Box$ states that ψ must hold **globally**. The common logical connectors $\rightarrow$, $\vee$ and $\leftrightarrow$ are defined in the usual way:

$$\begin{aligned} \psi \vee \phi &\equiv \neg(\neg\psi \wedge \neg\phi) \\ \psi \rightarrow \phi &\equiv \neg\psi \vee \phi \\ \psi \leftrightarrow \phi &\equiv (\psi \rightarrow \phi) \wedge (\phi \rightarrow \psi) \end{aligned}$$

The formal semantics are defined by the smallest relation $\models$, which states what traces satisfy an LTL formula.

Definition 3.5 (LTL, Semantics [29]) Let $p \in AP$ and $t \in TR$.

$$\begin{array}{lll} t \models p & \text{iff} & p \in t[0] \\ t \models \neg\psi & \text{iff} & t \not\models \psi \\ t \models \psi \vee \phi & \text{iff} & t \models \psi \text{ or } t \models \phi \\ t \models \bigcirc\psi & \text{iff} & t[1,\infty] \models \psi \\ t \models \psi_1\ \mathcal{U}\ \psi_2 & \text{iff} & \text{there exists } i \geq 0\ :\ t[i,\infty] \models \psi_2 \\ & & \text{and for all } 0 \leq j < i \text{ we have } t[j,\infty] \models \psi_1 \end{array} \qquad \square$$

Definition 3.6 (LTL-SAT) Let ψ be an LTL formula. LTL-SAT is the problem to decide whether there exists a trace $t \in TR$, s.t. $t \models \psi$, where $\models$ is the relation given by the definition above. $\square$

Example 3.7 Let AP be $\{a, b\}$. Consider the LTL formula ψ:

$$(\bigcirc (\Box a))\ \mathcal{U}\ (\Box(a \vee b))$$

The first two traces, (3.1) and (3.2), satisfy ψ, whereas the last one, (3.3), does not.

$$\{\}\{a\}\{a\}\{a\}\{a\}\{a,b\}\{a,b\}(\{a\})^\omega \tag{3.1}$$

$$\{b\}^\omega \tag{3.2}$$

$$\{a\}\{a\}(\{\})^\omega \tag{3.3}$$

□

Theorem 3.8 (LTL-SAT is decidable [33]) *Given an LTL formula ψ that ranges over a finite set of atomic propositions AP. To determine if there exists a trace $t \in 2^{AP}$ s.t. $t \models \psi$ is decidable.*

Theorem 3.9 (Complexity of LTL-SAT [38]) *To determine if an LTL formula ψ is satisfiable is **PSpace-complete**.*

3.2 Hyperproperties

LTL implicitely reasons over a single execution trace, stating that all traces through a system fulfil the desired trace property. LTL is therefore unable to refer to multiple traces at the same time. Consider the following example, where we cannot formalize the desired property as a trace property, e.g. above mentioned observational determinism.

Example 3.10 (Observational Determinism [32] [28]) Consider the example of the hospital system from Chapter 1 sketched in Figure 1.2 again. Our goal is to formalize the so-called "observational determinism" property, stating the system behaves deterministically for an observer. For this reason, we must not only quantify over the traces in our system, but also put them into relation. Whereas LTL and its well-studied extensions CTL and CTL* are able to quantify over those traces differently, they cannot refer to them for establishing relations. For example, the property "if a patient checks in at the hospital there is no difference in the observable behaviour, e.g. the 'beep' tone" is not expressible. □

As an extension to trace properties, a hyperproperty is able to define relations between certain traces of a system (see Figure 3.2).

Definition 3.11 (Hyperproperty [15]) A set $H \subseteq 2^{TR}$ is called a hyperproperty. □

In the next section, we will define a logic for hyperproperties formally and formulate small example hyperproperties.

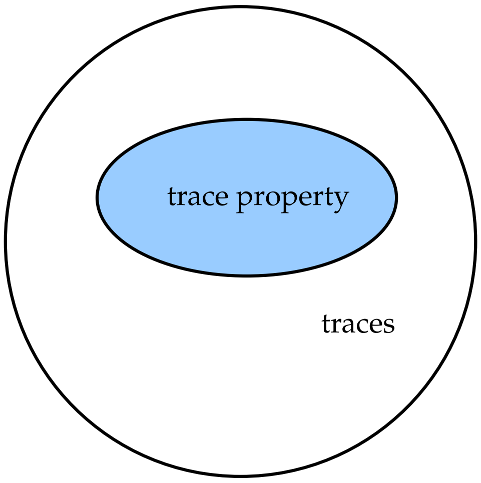

Figure 3.1: A Traceproperty

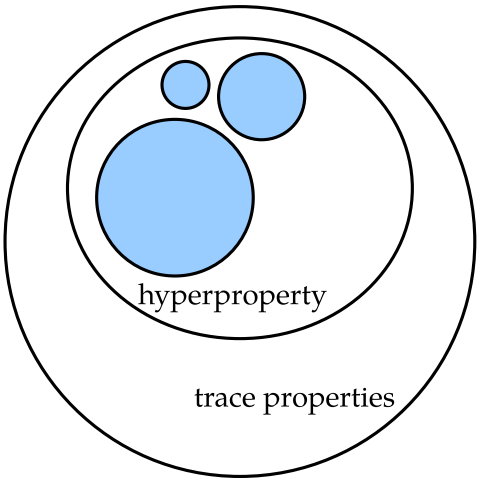

Figure 3.2: A Hyperproperty

3.3 HyperLTL - A Logic for Expressing Hyperproperties

Definition 3.12 (HyperLTL, Syntax [16]) Let π be a **trace variable** from an infinite supply $\mathcal{V}$ of trace variables. Formulae of HyperLTL are defined by the following grammar:

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

The temporal, logical and derived operators behave exactly as explained in Section 3.1. The important novelty in HyperLTL is that one can quantify over trace variables, usually denoted by π. This enables us to express properties like "on all traces ψ must hold", which is expressed by $\forall\pi.\ \psi$. Dually one can express that "there exists a trace such that ψ holds", which is denoted by $\exists\pi.\ \psi$. Example 3.15 provides further intuition.

The formal semantics are given with respect to a model $T \subseteq TR$, with the same syntax for manipulating traces as in Section 3.1. Π is a **trace assignment** from $\mathcal{V}$ to TR, i.e. a partial function mapping trace variables to explicit traces. $\Pi[\pi \mapsto t]$ denotes that π is mapped to t, with everything else mapped with respect to Π. $\Pi[i, \infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i, \infty]$ for all π.

Definition 3.13 (HyperLTL, Semantics [16])

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

Definition 3.14 (HyperLTL-SAT) Let ψ be an LTL formula. LTL-SAT is the problem to decide whether there exists a trace set T s.t. $\Pi \models_T \psi$, where Π is the empty trace assignment and $\models_S$ is the relation given by Definition 3.13. If it is clear from the context, we will omit Π and simply write $T \models \psi$. □

Example 3.15 Given the set of atomic propositions $\{a, b\}$. Consider the following HyperLTL formula, stating that for all traces π there must exist a trace π', s.t. for every occurrence in π of the atomic proposition a, b must hold at the next position in π'.

$$\forall \pi \exists \pi'.\ \square(a_\pi \rightarrow \bigcirc b_{\pi'})$$

The following trace set $\{p_1, p_2\}$ **does not** satisfy the formula, since there exists no trace s.t. b holds at the third position, which is required by the formula, since a holds at the second position in p_1.

$$p_1 : \{\}\{a\}(\{\})^\omega$$
$$p_2 : \{b\}\{b\}\{\}(\{b\})^\omega$$

There are infinitely many solutions for a trace set that satisfies the formula. A possible one is to replace p_2 with $\{b\}^\omega$. Another trace set satisfying the formula would be the singleton $\{p\}$, with p being the following trace.

$$p : (\{a, b\})^\omega$$
□

In this thesis we consider the satisfiability problem for every fragment of HyperLTL and provide multiple examples along the way. In later chapters, we will see that different quantifier patterns have a tremendous impact on the decidability and complexity of the satisfiability problem. We will begin with the easiest fragments where no quantifier alternation, as in Example 3.15, is allowed.

Chapter 4

HyperLTL-SAT for Alternation-free Fragments

For our task to identify every fragment of HyperLTL, for which the satisfiability problem is decidable, we will first take a look at the quantifier alternation-free fragments of HyperLTL. We will show that one can find canonical models for such fragments and that HyperLTL-SAT is decidable for those fragments.

Definition 4.1 (Alternation-free Fragments) We call a HyperLTL formula ϕ alternation-free iff the quantifier prefix only consists of a finite number of either universal quantifiers, we call this fragment $\forall^n$, or existential quantifiers, which we call $\exists^n$. □

Definition 4.2 (Equisatisfiability) We call a (Hyper)LTL formula ϕ equisatisfiable to another (Hyper)LTL formula ψ iff satisfiability of ϕ by a trace (set) T implies satisfiability of ψ by a trace (set) T' and vice versa. Note that T and T' must not necessarily be equal. □

4.1 $\exists^n$ - Fragment

By Definition 3.6, the satisfiability problem of an LTL formula ψ is implicitely existentially quantified and therefore equisatisfiable to the corresponding HyperLTL formula $\exists\pi.\psi'$. In ψ' every occurence of an atomic proposition a is replaced by a_π. In this section we will show that the satisfiability problem for $\exists^n$ (which subsumes $\exists^1$) can be reduced to plain LTL-SAT and therefore remains decidable. We will also investigate the complexity, which is **PSpace-complete**.

Example 4.3 ($\exists^n$-Formula) Consider the following example for $AP = \{a, b\}$.

$$\exists\pi\exists\pi'.\ a_\pi \wedge \Box\neg b_\pi \tag{4.1}$$

$$\wedge\ (\Box(a_\pi \rightarrow \bigcirc b_{\pi'})) \tag{4.2}$$

$$\wedge\ (\Box(b_{\pi'} \rightarrow a_\pi)) \tag{4.3}$$

The subformula (4.1) denotes that trace π starts with an atomic proposition a but is not allowed to contain a b at any position. (4.2) states a dependency between traces π and π', which is not possible in plain LTL. We are now able to express that if there is an a at any position of trace π, then there must be a b in the next position of trace π'. Analogously (4.3) requires that if there is a b in π' there has to be an a at the exact same position in the other trace, π. One possible trace set satisfying this HyperLTL formula is $\{p, p'\}$, sketched below, where the arrows denote the dependencies mentioned above.

$$\begin{array}{l} p : \{a\}\{a\}\{a\}\ \dots \\ \qquad \searrow\uparrow\searrow\uparrow\searrow\ \dots \\ p' : \{\ \}\{b\}\{b\}\{b\}\ \dots \end{array}$$ □

As it turns out, we can solve the satisfiability problem for a $\exists^n$ formula by exploiting its connection to plain LTL. The idea of the construction is to replace every occurrence of a trace-labelled atomic proposition a, i.e. a_π, with a fresh atomic proposition. The resulting LTL formula can then be checked for satisfiability and from the resulting trace $\tilde{p}$ the witnesses for $\exists\pi_1, \dots, \exists\pi_n$ can be reconstructed.

Construction 4.4 ($\psi_\exists$) Let AP be $\{a_1, \dots, a_m\}$ for $m \in \mathbb{N}$. Let $\exists\pi_1, \dots, \exists\pi_n.\psi$ be an $\exists^n$ HyperLTL formula for $n \in \mathbb{N}$. We create an LTL formula $\psi_\exists$ by replacing all trace-labelled atomic proposition $a_{1_{\pi_1}}, \dots, a_{1_{\pi_n}}, \dots, a_{m_{\pi_1}}, \dots, a_{m_{\pi_n}}$ with fresh atomic propositions of $\widetilde{AP}$ and by discarding the quantifier prefix completely. The substitution is represented by the mapping $s \in AP \times \{1, \dots, n\} \to \widetilde{AP}$, where $\{1, \dots, n\}$ represents the quantifiers. Note that it suffices that $|\widetilde{AP}| = n \cdot m$. □

Definition 4.5 (Projection, *pr*) Let AP be a set of atomic propositions. Let $s \in AP \times \{1, \dots, n\} \to \widetilde{AP}$ be a substitution that maps trace-labelled atomic propositions to fresh ones. Let t be a trace that ranges over $\widetilde{AP}$ and ψ an $\exists^n$ HyperLTL formula with quantifier prefix length n. We also define $s(M, i)$ for a set $M \subseteq AP$ and $0 \leq i \leq n$ as $\{s(p, i) \mid p \in M\}$. The projection pr_i^s for the ith quantifier is defined as the trace $\tilde{t}$, s.t. $\forall j \in \mathbb{N}.\tilde{t}[j] = s^{-1}(t[j], i)$. This denotes that only the atomic propositions that correspond to the ith quantifier are kept in the trace. $pr^s(t) := \bigcup_{i=0}^{n} pr_i^s(t)$ therefore splits the trace t into a trace set containing n traces. □

Example 4.6 Consider the HyperLTL formula from Example 4.3 again, which ranges over $\{a, b\}$:

$$\exists\pi\exists\pi'.\ a_\pi \wedge \Box\neg b_\pi \wedge (\Box(a_\pi \to \bigcirc b_{\pi'})) \wedge (\Box(b_{\pi'} \to a_\pi))$$

By introducing a fresh set of atomic propositions $\widetilde{AP} = \{c, d, c', d'\}$ the substitution s with $a_\pi \mapsto c$, $b_\pi \mapsto d$, $a_{\pi'} \mapsto c'$ and $b_{\pi'} \mapsto d'$ yields the following LTL formula.

$$c \wedge \Box\neg d \wedge (\Box(c \to \bigcirc d')) \wedge (\Box(d' \to c))$$

For a better intuition we can simplify this formula by using the definition of implication.

$$c \wedge \square \neg d \wedge (\square(\neg c \vee \bigcirc d')) \wedge (\square(\neg d' \vee c))$$

This plain LTL formula is satisfied by the following trace $\tilde{p}$, which ranges over $\widetilde{AP}$. Notice that d and d' are different atomic propositions and therefore the subformula $\square \neg d$ still holds.

$$\tilde{p} : \{c\}\{c, d'\}\{c, d'\}\{c, d'\} \ldots$$

By mapping the substituted, fresh atomic propositions back with the defined projection pr^s, we can construct witnesses for π and π', by splitting $\tilde{p}$ into two traces p_π and $p_{\pi'}$, where for every position in these traces only those atomic propositions hold that where labelled with π or π' respectively.

$$p_\pi : \{a\}\{a\}\{a\}\{a\} \ldots$$
$$p_{\pi'} : \{\ \}\{b\}\{b\}\{b\} \ldots$$

□

Lemma 4.7 (Equisatisfiability of $\exists^n$ and $\psi_\exists$) *A given $\exists^n$ HyperLTL formula $\exists\pi_1, ..., \exists\pi_n.\ \psi$ is satisfiable by a trace set $pr(t)$ iff the plain LTL formula $\psi_\exists$ is satisfied by a trace t.*

Proof Let AP be a set of atomic propositions. Let $s \in AP \times \{1, \ldots, n\} \to \widetilde{AP}$ as defined as in Construction 4.4.

$\rightarrow$:
Let $\exists\pi_1, \ldots, \exists\pi_n.\ \psi$ be an $\exists^n$ HyperLTL formula for an $n \in \mathbb{N}$. Assume there exists a trace set T, s.t. $TR \supseteq T \models \exists\pi_1, \ldots, \pi_n.\ \psi$. Let $p_1, \ldots, p_n$ be the witnesses of $\pi_1, \ldots, \pi_n$. We define $\tilde{t}$ as the trace that is generated by zipping the traces $p_1, \ldots, p_n \in T$ together, which means that $\forall j \in \mathbb{N}.\ \tilde{t}[j] := \bigcup_{i=1}^{n} s(p_i[j], i)$. Since $\tilde{t}$ is exactly the trace that satisfies $\psi_\exists$ by definition of Construction 4.4, $\tilde{t} \models \psi_\exists$ must hold. This means we found a witness for satisfiability, namely $\tilde{t}$.

$\leftarrow$:
Let $\exists\pi_1, ..., \exists\pi_n.\ \psi$ be a $\exists^n$ HyperLTL formula for an $n \in \mathbb{N}$ and $\psi_\exists$ be the corresponding LTL formula from Construction 4.4. Assume $\tilde{t}$ satisfies $\psi_\exists$. We need to show that we can find witnesses $p_1, \ldots, p_n$, s.t. $\{p_1, \ldots, p_n\} \models \exists\pi_1, ..., \exists\pi_n.\ \psi$. We use Definition 4.5 to construct the desired trace set $\{p_1, \ldots, p_n\}$, which is $pr^s(\tilde{t})$. Let k be an arbitrary position in $\tilde{t}$, i be an arbitrary quantifier position and a be an arbitrary atomic proposition, where $s(a, i) = a'$. We distinguish two cases. Assume a' holds at position k in $\tilde{t}$. The projection pr^s simply projects a' to a at position k

of the witness of the ith quantifier, i.e. of p_i. If a' does not hold at position k, a' is not projected to position k in p_i, i.e. a does not hold. Satisfiability is therefore preserved by the projection pr^s of Definition 4.5.

With the Construction 4.4 above we have shown that $\exists\pi_1, ..., \exists\pi_n.\ \psi$ and $\psi_\exists$ are equisatisfiable. ■

Theorem 4.8 (Decidability of $\exists^n$-SAT) *HyperLTL-SAT is decidable for formulae of the $\exists^n$ fragment.*

Proof Given a $\exists^n$ HyperLTL formula ψ, we reduce ψ with Construction 4.4 above to an LTL formula $\psi_\exists$. We established an equisatisfiability relation between ψ and $\psi_\exists$ with Lemma 4.7. Since there are computable algorithms for determining if an LTL formula is satisfiable, we can decide the satisfiability of ψ too. ■

Theorem 4.9 (Complexity) *HyperLTL-SAT is **PSpace-complete** for the $\exists^n$ fragment.*

Proof We use Construction 4.4 to transform our problem to a plain LTL formula in polynomial space. By Theorem 3.9 LTL-SAT is in **PSpace** and thus $\exists^n$ HyperLTL-SAT is in **PSpace** too.
Hardness. We prove hardness by a reduction from LTL-SAT, which is **PSpace-hard** by Theorem 3.9. An LTL formula ϕ is reduced to a HyperLTL formula $\exists\pi.\ \phi(\pi)$, where $\phi(\pi)$ denotes that every atomic proposition occurring in ϕ is labelled with π. By definition the resulting formulae $\exists\pi.\ \phi(\pi)$ is satisfiable iff ϕ is satisfiable. By Theorem 2.21, HyperLTL-SAT for $\exists^n$ is **PSpace-hard**. By Definition 2.22 HyperLTL-SAT for $\exists^n$ is **PSpace-complete**. ■

In this section we proved that there exists a canonical model for $\exists^n$ HyperLTL, for which the satisfiability problem is decidable. The construction and the proof were based on the idea of zipping the different traces together. After finding a trace that satisfies the plain LTL formula resulting from Construction 4.4, we can reconstruct the trace set again with the projection from Definition 4.5. In the next section, we will explore the universally quantified fragment of HyperLTL, its satisfiability problem, as well as its relation to LTL satisfiability.

4.2 $\forall^n$ - Fragment

This fragment is very rich in the sense that there are many interesting hyperproperties that can be expressed in it. We can formulate observational determinism from Example 3.10 formally in HyperLTL. We will see that the satisfiability problem for this fragment is trivially solvable by the empty trace set. However as soon as this solution is excluded we are still able to provide a canonical model for this fragment.

Example 4.10 (Observational Determinism) Formulating that a low-security user, such as a visitor in a hospital, is not allowed to infer any high-security data, can be easily done in HyperLTL. Assume we have a set of atomic propositions split into high security inputs, denoted by H, low security inputs, denoted by L and low security outputs, denoted by O.

$$\forall \pi \forall \pi'. \; (\bigwedge_{in \in L} in_\pi \leftrightarrow in_{\pi'}) \rightarrow \Box (\bigwedge_{out \in O} out_\pi \leftrightarrow out_{\pi'})$$

No matter how the **high**-security inputs are chosen, as long as the **low**-security inputs are the same in a pair of computation traces, the observable outputs remain the same on this pair too. □

It is remarkable that the $\forall^n$ fragment of HyperLTL can express the impact of secrecy constraints on the functional requirement of a reactive system. We will illustrate this further in the following example.

Example 4.11 Consider the atomic propositions $\{out, in\}$, where the proposition **out** represents an observable output. The system is not allowed to reveal any information stored in **out** by behaving nondeterministically. The proposition **in** represents an input, for example given by an admin, that triggers the output.

$$\forall \pi . \forall \pi'. \; \Box (in_\pi \rightarrow \bigcirc out_\pi) \tag{4.4}$$

$$\wedge \Box (out_\pi \leftrightarrow out_{\pi'}) \tag{4.5}$$

It is important to note that the constraint that the system should behave deterministicly for an observer influences the other traces through the reactive system. The subformula in (4.4) states that the input only triggers the output for the trace π. However since (4.5) establishes a secrecy constraint, the output must change on all possible computation traces through the system. □

The satisfiability problem for such an universally quantified formula is not challenging, since a system would never dare to progress, if there is a possibility of failing and no reward for progressing. Therefore, if this property is the only requirement of a system, the system can fulfil this property by simply doing nothing.

Theorem 4.12 (HyperLTL-SAT for $\forall^n$) *Given a $\forall^n$ HyperLTL formula ψ, to determine if the formula is satisfied by a trace set S is decidable.*

Proof Every $\forall^n$ HyperLTL formula is trivially satisfied by the empty set. ■

Besides this trivial solution for $\forall^n$-SAT we can observe an interesting relation between the satisfiability of a $\forall^n$ formula and an $\exists^1$ HyperLTL formula (respectively an LTL formula), if we exclude the empty trace set. Since a property, expressed by an universally quantified formula, is often not the only property that a system needs to ensure, assuming that the resulting model will not be empty is robust.

Definition 4.13 (HyperLTL-SAT$^+$) Let a HyperLTL formula ϕ, that ranges over a set of atomic propositions AP be given. HyperLTL-SAT$^+$ is defined as the question wether there exists a **non-empty** trace set T, s.t. $T \models \phi$. □

Construction 4.14 (ϕ^{-l}) Given a HyperLTL formula $Q.\ \phi$, where Q is an arbitrary quantifier prefix, delete all trace variables. We denote the resulting formula by ϕ^{-l}. □

Lemma 4.15 *Let a $\forall^n$ HyperLTL formula $\forall\pi_0...\pi_n.\ \phi$ be given.*

$$\forall\, T \in 2^{TR}.((T \models \forall\pi_0...\pi_n.\ \phi) \rightarrow \forall s \in T.\ s \models \phi^{-l})$$

Stating that if there is a trace set T satisfying a $\forall^n$ HyperLTL formula, then every element of this set is a solution for the resulting LTL formula if one deletes the labels from the atomic propositions and discards the quantifier prefix.

Proof Let a $\forall^n$ HyperLTL formula $\forall\pi_0, \ldots, \pi_n.\ \phi$ over a set of atomic propositions AP be given. Let $T \in 2^{TR}$ be an arbitrary trace set satisfying $\forall\pi_0, \ldots, \pi_n.\ \phi$. We distinguish two cases. If T is the empty set, then the statement trivially holds. If T is not empty, let $s \in T$ be arbitrary. By assumption the singleton $\{s\}$ satisfies $\forall\pi_0, \ldots, \pi_n.\ \phi$ by choosing s as a witness for all trace variables $\pi_0, \ldots, \pi_n$. Hence s satisfies ϕ^{-l}. ■

Lemma 4.16 ($\forall^n$ Equisatisfiability$^+$ to ϕ^{-l}) *Let a HyperLTL formula $\forall\pi_0, \ldots, \pi_n.\ \phi$ be given. $\forall\pi_0, \ldots, \pi_n.\ \phi$ is satisfiable$^+$ iff the LTL formula ϕ^{-l} is satisfiable.*

Proof Let a $\forall^n$ HyperLTL formula $\forall\pi_0...\forall\pi_n.\ \phi$ and its corresponding LTL formula ϕ^{-l} be given.

$\rightarrow$:
Assume $\forall\pi_0...\forall\pi_n.\ \phi$ is satisfiable$^+$ by a trace set T. By definition T is not empty. Choose an arbitrary trace $t \in T$. By Lemma 4.15 t is therefore a witness for the satisfiability of ϕ^{-l}, i.e. ϕ^{-l} is satisfiable.

$\leftarrow$:
Assume the LTL formula ϕ^{-l} is satisfiable by a trace t. A trace set satisfying $\forall\pi_0...\forall\pi_n.\ \phi$ is $\{t\}$, since the universal-quantification only ranges over a single trace.

Therefore a $\forall^n$ HyperLTL formula $\forall\pi_0...\forall\pi_n.\ \phi$ is equisatisfiable$^+$ to its corresponding LTL formula ϕ^{-l}. ■

Using equisatisfiability of a $\forall^n$ HyperLTL formula ϕ and its corresponding LTL formula ϕ^{-l}, we can decide HyperLTL-SAT$^+$ for a $\forall^n$ formula.

Theorem 4.17 *HyperLTL-SAT$^+$ is decidable for the $\forall^n$ fragment.*

Proof Given a $\forall^n$ HyperLTL formula ϕ, we can reduce this problem by constructing ϕ^{-l}, which is a plain LTL formula for which the satisfiability problem can be solved. Since we showed equisatisfiability$^+$ of ϕ^{-l} and ϕ in Lemma 4.16, HyperLTL-SAT$^+$ is decidable. ∎

Theorem 4.18 (Complexity) *HyperLTL-SAT$^+$ is **PSpace-complete** for the $\forall^n$ fragment.*

Proof We use Construction 4.14 to transform our problem to an LTL formula in polynomial space. By Theorem 3.9 LTL-SAT is in **PSpace** and thus $\forall^n$ HyperLTL-SAT is in **PSpace** too.
Hardness. We prove hardness with a reduction from LTL-SAT, which is **PSpace-hard** by Theorem 3.9. An LTL formula ϕ is reduced to a HyperLTL formula $\forall\pi.\ \phi(\pi)$, where $\phi(\pi)$ denotes that every atomic proposition occurring in ϕ is labelled with π. The resulting formulae $\forall\pi.\ \phi(\pi)$ is satisfiable$^+$ by the singleton $\{s\}$ iff ϕ is satisfiable by s. By Theorem 2.21, HyperLTL-SAT for $\forall^n$ is **PSpace-hard**. By Definition 2.22 HyperLTL-SAT for $\forall^n$ is **PSpace-complete**. ∎

In this chapter, we proved that one can find canonical models for HyperLTL formulae without any quantifier alternation, by establishing interesting equisatisfiability relations between those different fragments. Suprisingly the satisfiability problem for both alternation-free HyperLTL fragments can be reduced to LTL-SAT, establishing **PSpace-completeness** for every problem we have considered so far. We will leave this comfort zone in the next section by allowing a single quantifier alternation. We will put the results to work by showing that equivalence of two $\forall^n$ HyperLTL formulae can be decided.

Chapter 5

HyperLTL-SAT for One Alternation & Equivalence of Universally Quantified HyperLTL Formulae

This chapter is split in three sections. In the first section we consider the satisfiability problem for HyperLTL formulae with one quantifier alternation that start with an universal quantifier. The counterpart, formulae with one quantifier alternation that start with an existential quantifier, is subject of the second section.

We will conclude this chapter with an interesting application of those results, by showing that we can decide whether two universally quantified HyperLTL formulae are equivalent. Since verifying an implementation against a specification is a difficult task, the ability to invoke a procedure that gets rid of redundancy in a specification is highly desirable.

5.1 $\forall^m\exists^n$ HyperLTL-SAT

Definition 5.1 ($\forall^m\exists^n$) Let n and m be natural numbers. A HyperLTL formula is in the $\forall^m\exists^n$ fragment iff it is of the form $\forall\pi_0 ... \forall\pi_m\exists\pi'_0 ... \exists\pi'_n.\ \psi$, where only one quantifier alternation is allowed, starting with an universal quantifier. □

Theorem 5.2 *The satisfiability problem for a HyperLTL formula whose quantifier prefix starts with an universal quantifier is decidable.*

Proof Let a HyperLTL formula $\forall\pi Q.\ \phi$ be given, where Q is an arbitrary quantifier prefix. By definition this formula is satisfied by the empty set. ■

Corollary 5.3 (Decidability of $\forall^m\exists^n$) *Let a $\forall^m\exists^n$ HyperLTL formula $\forall\pi_0 ... \forall\pi_m \exists\pi'_0 ... \exists\pi'_n.\ \psi$ be given. Determining if there exists a trace set S, s.t. $S \models \forall\pi_0 ... \forall\pi_m \exists\pi'_0 ... \exists\pi'_n.\ \psi$ is decidable.*

However the HyperLTL-SAT$^+$ problem for the $\forall^m\exists^n$ fragment is not as easy. In fact we can prove its undecidability based on a reduction we consider in Chapter 6. We therefore postpone this question to the corresponding Section 6.3. The fragment of HyperLTL that allows one quantifier alternation starting with an existential quantifier will be discussed in the next section.

5.2 $\exists^n\forall^m$ HyperLTL-SAT

Definition 5.4 ($\exists^n\forall^m$) Let m and n be natural numbers. A HyperLTL formula is in the $\exists^n\forall^m$ fragment iff it is of the form $\exists\pi_0...\exists\pi_n\forall\pi'_0...\forall\pi'_m.\ \psi$, where only one quantifier alternation is allowed, starting with an existential quantifier. □

Example 5.5 Consider the following $\exists^n\forall^m$ HyperLTL formula ranging over $\{a, b\}$:

$$\exists\pi\forall\pi'\forall\pi''.\ a_\pi \tag{5.1}$$
$$\wedge\ (a_\pi \rightarrow \bigcirc b_{\pi'}) \tag{5.2}$$
$$\wedge\ \Box(b_{\pi''} \rightarrow \bigcirc b_{\pi''}) \tag{5.3}$$

The subformula (5.1) states that a trace π starting with the atomic proposition a has to exist in the model. It is important to remark that (5.2) and (5.3) must hold for every trace in the model, and therefore especially for the witness of π itself. A trace set satisfying this formula is, for example, $\{q, q'\}$, with q and q' being the following traces.

$$q:\ \{a\}(\{b\})^\omega$$
$$q':\ (\{b\})^\omega$$

The trace set $\{p\}$ also does satisfy the HyperLTL formula, where p is the following trace.

$$p: \{a\}(\{b\})^\omega$$

We can also simplify the HyperLTL formula to the following one, which is equisatisfiable to the one mentioned above, since the universal quantifiers must only range over a single trace.

$$\begin{aligned}&\exists\pi.\ a_\pi\\ &\quad\wedge\ (a_\pi \rightarrow \bigcirc b_\pi)\\ &\quad\wedge\ \Box(b_\pi \rightarrow \bigcirc b_\pi)\end{aligned}$$

□

From this example, we can infer two properties of the $\exists^n\forall^m$ fragment of HyperLTL. Firstly, for every trace set T, which is satisfying a given HyperLTL formula $\exists\pi_0...\exists\pi_n\forall\pi'_0...\forall\pi'_m.\ \psi$, there exists a subset $T \supseteq T' \models \exists\pi_0...\exists\pi_n\forall\pi'_0...\forall\pi'_m.\ \psi$ s.t. $|T'|$ is at most n. Secondly, one can get rid of redundant universal quantifiers. In the following, we will in fact see that we can get rid of every universal quantifier.

Theorem 5.6 *Let a HyperLTL formula $\exists\pi_0\dots\exists\pi_n\forall\pi'_0\dots\forall\pi'_m.\ \psi$ be given.*

$$\begin{aligned}&\forall T \in 2^{TR}.\ (T \models \exists\pi_0...\exists\pi_n\forall\pi'_0...\forall\pi'_m.\ \psi) \rightarrow \\ &\qquad \exists T' \subseteq T.\ (T' \models \exists\pi_0...\exists\pi_n\forall\pi'_0...\forall\pi'_m.\ \psi) \wedge |T'| \leq n\end{aligned}$$

Stating that for all trace sets S that satisfy the formula, there exists a subset S' that still satisfies the formula, but for which the cardinality is bound by the number of existential quantifiers.

Proof Let an $\exists^n\forall^m$ HyperLTL formula $\exists\pi_0\dots\exists\pi_n\forall\pi'_0\dots\forall\pi'_m.\ \psi$ be given. Let T be arbitrary and let T satisfy the given formula. This means there exist at most n witnesses $p_0,\dots,p_n$ for $\pi_0\dots\pi_n$. Choose T' as $\{p_0,\dots,p_n\}$. Since the remaining trace variables are universally quantified, T' must satisfy the given formula as well. ■

Considering satisfiability, we can now assume, by Theorem 5.6, without loss of generality that every model satisfying an $\exists^n\forall^m$ formula has at most n elements. The idea for deciding the satisfiability of an $\exists^n\forall^m$ HyperLTL formula is to unroll every possible assignment of the trace variables bound by universal quantifiers.

Construction 5.7 (Substitution with All Possibilities) We define a function sp that takes an $\exists\pi_0\dots\exists\pi_n\forall\pi'_0\dots\forall\pi'_m.\ \psi$ and yields an $\exists^n$ HyperLTL formula ψ' of the following shape, where $\psi[\pi'_i\backslash\pi_i]$ denotes that the trace variable π'_i in ψ is replaced by π_i.

$$\begin{aligned}\exists\pi_0\dots\exists\pi_n.\ \bigwedge_{i=1}^{n}(&\bigwedge_{j=1}^{n}\psi[\pi'_1\backslash\pi_j][\pi'_2\backslash\pi_i]\dots[\pi'_{n-1}\backslash\pi_i][\pi'_m\backslash\pi_i]\\ &\bigwedge_{j=1}^{n}\psi[\pi'_1\backslash\pi_i][\pi'_2\backslash\pi_j]\dots[\pi'_{n-1}\backslash\pi_i][\pi'_m\backslash\pi_i]\\ &\dots\\ &\bigwedge_{j=1}^{n}\psi[\pi'_1\backslash\pi_i][\pi'_2\backslash\pi_i]\dots[\pi'_{n-1}\backslash\pi_i][\pi'_m\backslash\pi_j])\end{aligned}$$ □

Example 5.8 Consider the following $\exists^n\forall^m$ formula ranging over $\Sigma = \{a,b,c,d\}$:

$$\exists\pi_0\exists\pi_1\forall\pi'_0\forall\pi'_1.\ (\Box a_{\pi'_0} \wedge \Box b_{\pi'_1}) \wedge (\Box c_{\pi_0} \wedge \Box d_{\pi_1})$$
$$sp(\exists\pi_0\exists\pi_1\forall\pi'_0\forall\pi'_1.\ (\Box a_{\pi'_0} \wedge \Box b_{\pi'_1}) \wedge (\Box c_{\pi_0} \wedge \Box d_{\pi_1}))\text{ yields :}$$

$$\exists\pi_0\exists\pi_1.\ (\Box a_{\pi_0} \land \Box b_{\pi_0}) \land (\Box c_{\pi_0} \land \Box d_{\pi_1}) \tag{5.4}$$
$$\land(\Box a_{\pi_1} \land \Box b_{\pi_0}) \land (\Box c_{\pi_0} \land \Box d_{\pi_1}) \tag{5.5}$$
$$\land(\Box a_{\pi_0} \land \Box b_{\pi_1}) \land (\Box c_{\pi_0} \land \Box d_{\pi_1}) \tag{5.6}$$
$$\land(\Box a_{\pi_1} \land \Box b_{\pi_1}) \land (\Box c_{\pi_0} \land \Box d_{\pi_1}) \tag{5.7}$$

We replace the universally quantified trace variables with every possible combination of existentially quantified variables. As the subformulae (5.4 - 5.7) indicate, this approach blows up the formula. Note that subformulae of the original formula that are isolated from universally quantified trace variables stay exactly the same, namely $(\Box c_{\pi_0} \land \Box d_{\pi_1})$.

A possible model that satisfies this example formula is the trace set $\{p_0, p_1\}$.

$$p_0 : (\{a, b, c\})^\omega$$
$$p_1 : (\{a, b, d\})^\omega$$

A second possible trace set satisfying this formula is the singleton $\{p\}$, where p is a witness for both trace variables π_0 and π_1.

$$p : (\{a, b, c, d\})^\omega$$ □

Note that this construction transforms an $\exists^n\forall^1$ HyperLTL formula to an $\exists^n$ HyperLTL formula, which is a special case. Therefore the formula of Example 5.5 could be reduced to $\exists\pi.\ a_\pi \land (a_\pi \to \bigcirc b_\pi) \land \Box(b_\pi \to \bigcirc b_\pi)$.

Lemma 5.9 (Equisatisfiability of $\exists^n\forall^m$ and $sp(\exists^n\forall^m)$) *A HyperLTL formula $\exists\pi_0 \ldots \exists\pi_n\forall\pi'_0 \ldots \forall\pi'_m.\ \phi$ is satisfiable iff $sp(\exists\pi_0 \ldots \exists\pi_n\forall\pi'_0 \ldots \forall\pi'_m.\ \phi)$ is satisfiable.*

Proof Let a HyperLTL formula $\exists\pi_0 \ldots \exists\pi_n\forall\pi'_0 \ldots \forall\pi'_m.\ \phi$ be given.

$\to$:
Assume $\exists\pi_0 \ldots \exists\pi_n\forall\pi'_0 \ldots \forall\pi'_m.\ \phi$ is satisfied by a trace set T. There exists, by Theorem 5.6, a trace set $\{p_0, \ldots, p_n\}$, that satisfy the formula, where $p_0, \ldots, p_n$ are witnesses for $\pi_0, \ldots, \pi_n$ respectively. Choose the trace set $\{p_0, \ldots, p_n\}$ as a witness. It satisfies $sp(\exists\pi_0 \ldots \exists\pi_n\forall\pi'_0 \ldots \forall\pi'_m.\ \phi)$ by construction, since there are only **finitely** many traces and universal-quantifiers, which can simply be unrolled.

$\leftarrow$:
Assume $sp(\exists\pi_0 \ldots \exists\pi_n\forall\pi'_0 \ldots \forall\pi'_m.\ \phi)$ is satisfied by a trace set T. There exists, by Theorem 5.6, a trace set $\{p_0, \ldots, p_n\}$, that satisfies the formula, where $p_0, \ldots, p_n$ are witnesses for $\pi_0, \ldots, \pi_n$ respectively. By Construction 5.7 the resulting formula covers every possible combination of trace assignments for the universal-quantified variables $\pi'_0 \ldots \pi'_m$ by unrolling the **finite** quantifier prefix. So $\{p_0, \ldots, p_n\}$ can be chosen as a witness for the satisfiability of $\exists\pi_0 \ldots \exists\pi_n\forall\pi'_0 \ldots \forall\pi'_m.\ \phi$. ■

Theorem 5.10 ($\exists^n\forall^m$-SAT) *The satisfiability of a given $\exists^n\forall^m$ HyperLTL formula is decidable.*

Proof We can reduce a given $\exists^n\forall^m$ HyperLTL formula ϕ to an $\exists^n$ formula, using the Construction 5.7. Since $\exists^n$ is decidable by Theorem 4.8, HyperLTL-SAT is decidable for the $\exists^n\forall^m$ fragment too. ■

Theorem 5.11 (Complexity) *Let an $\exists^n\forall^m$ HyperLTL formula ϕ be given. To determine whether there exists a trace set T, s.t. $T \models \phi$ lies in **EXPSpace** and is **PSpace-hard**.*

Proof The satisfiability problem of $\exists^n\forall^m$ can be decided by a Turing machine in $O(n^m)$ space, by unrolling every possibility, and lies therefore in **EXPSpace**.
Hardness. We reduce from the satisfiability problem of an $\exists^n$ HyperLTL formula, which is **PSpace-hard** by Theorem 4.9. It is a special case of an $\exists^n\forall^m$ formula with m equal to 0. By Theorem 2.21 HyperLTL-SAT for $\exists^n$ is therefore **PSpace-hard**. ■

Note that it is still open whether the satisfiability problem for $\exists^n\forall^m$ is **EXP-hard** or in **PSpace**. See Section 8.2 for a short discussion on this topic.

We restrict this potential exponential blow up, by defining a bounded version of the problem for which the satisfiability problem is **PSpace-complete**, which will be discussed in the next section.

5.3 b-Bounded $\exists^n\forall^m$ HyperLTL-SAT

We define a bounded version of the satisfiability problem for $\exists^n\forall^m$, by bounding the number of universal-quantifiers that may occur in the HyperLTL formula by a constant $b \in \mathbb{N}$.

Definition 5.12 (b-Bounded $\exists^n\forall^m$ HyperLTL-SAT) Let a HyperLTL formula $\exists\pi'_0 \ldots \pi'_n\forall\pi_0 \ldots \pi_m.\ \psi$ be given. b-Bounded $\exists^n\forall^m$-SAT is defined as the problem of finding a trace set T, s.t. $T \models \exists\pi'_0 \ldots \pi'_n\forall\pi_0 \ldots \pi_m.\ \psi$, where $m \leq b$. □

Theorem 5.13 *b-Bounded $\exists^n\forall^m$ HyperLTL-SAT is **PSpace-complete**.*

Proof A b-Bounded HyperLTL formula $\exists^n\forall^m$ can be unrolled by a Turing machine in $O(n^b)$, hence the unrolling remains polynomial. The satisfiability problem of a b-Bounded HyperLTL formula $\exists^n\forall^m$ can therefore be decided in **PSpace**, since the unrolled formula is $\exists^n$ HyperLTL formula, for which the satisfiability can be decided in **PSpace** by Theorem 4.9.
Hardness. We reduce from the satisfiability problem of an $\exists^n$ HyperLTL formula, which is **PSpace-hard** by Theorem 4.9. It is a special case of an $\exists^n\forall^b$ formula with m being equal to 0 and $\forall b \in \mathbb{N}.\ 0 \leq b$. By Theorem 2.21 b-Bounded HyperLTL-SAT for $\exists^n\forall^m$ is therefore **PSpace-hard**. By Definition 2.22 b-Bounded HyperLTL-SAT for $\exists^n\forall^m$ is **PSpace-complete**. ■

With this knowledge, we can now discuss a major application of the proven theorems, namely equivalence checking of universally quantified HyperLTL formulae. In the next section, we will formally define equivalence of HyperLTL formulae. We will prove that the question whether two universally quantified HyperLTL formulae are equivalent is decidable, and consider the complexity of equivalence checking.

5.4 Equivalence of Universally Quantified HyperLTL Formulae

Based on the results, we will work our way towards an algorithm that decides if two universally quantified HyperLTL formulae are equivalent. We will refer to this decision problem as $\forall^n \leftrightarrow \forall^m$.

Definition 5.14 (Equivalence of HyperLTL Formulae) We call two HyperLTL formulae ψ and ϕ equivalent iff every trace set T satisfying ψ also satisfies ϕ and vice versa. $\square$

It is important to note that we are no longer considering only equisatisfiability but equivalence, which means that there exists no trace set T that satisfies exactly one of the HyperLTL formulae. This would be possible for equisatisfiability. We will start with an important lemma that we will need for the proof of the main Theorem 5.16. We denote a HyperLTL formula ψ in which $\pi_1 \dots \pi_n$ are the only trace variables by $\psi(\pi_1 \dots \pi_n)$.

Lemma 5.15 *If there exists a trace set T that satisfies both a given $\forall^n$ HyperLTL formula $\forall \pi_0, \dots, \pi_n.\ \phi(\pi_0, \dots, \pi_n)$ and an $\exists^m$ HyperLTL formula $\exists \pi'_0, \dots, \pi'_m.\ \psi(\pi'_0, \dots, \pi'_m)$, then the $\exists^m\forall^n$ HyperLTL formula $\exists \tilde{\pi}_0, \dots, \tilde{\pi}_m \forall \pi_0, \dots, \pi_n.\ \phi(\pi_0, \dots, \pi_n) \wedge \psi(\tilde{\pi}_0, \dots, \tilde{\pi}_m)$ is satisfiable, and vice versa.*

Proof Let a $\forall^n$ HyperLTL formula $\forall \pi_0, \dots, \pi_n.\ \phi(\pi_0, \dots, \pi_n)$, an $\exists^m$ HyperLTL formula $\exists \pi'_0, \dots, \pi'_m.\ \psi(\pi'_0, \dots, \pi'_m)$ and an $\exists^m\forall^n$ HyperLTL formula $\exists \tilde{\pi}_0, \dots, \tilde{\pi}_m$ $\forall \pi_0, \dots, \pi_n.\ \phi(\pi_0, \dots, \pi_n) \wedge \psi(\tilde{\pi}_0, \dots, \tilde{\pi}_m)$ be given.

$\rightarrow$:
Assume there exists a trace set T that satisfies both $\forall \pi_0, \dots, \pi_n.\ \phi(\pi_0, \dots, \pi_n)$ and $\exists \pi'_0, \dots, \pi'_m.\ \psi(\pi'_0, \dots, \pi'_m)$. Therefore witnesses for $\pi'_0, \dots, \pi'_m$, namely $p'_0, \dots, p'_m \in T$, must exist. Since $\pi_0, \dots, \pi_n$ are universally quantified trace variables, ϕ must hold for all traces in T, especially for $p'_0, \dots, p'_m$. Choose those traces again as witnesses for $\tilde{\pi}_0, \dots, \tilde{\pi}_m$, since ψ does not contain the universally quantified trace variables $\pi_0, \dots, \pi_n$. $\exists \tilde{\pi}_0, \dots, \tilde{\pi}_m\ \forall \pi_0, \dots, \pi_n.\ \phi(\pi_0, \dots, \pi_n) \wedge \psi(\tilde{\pi}_0, \dots, \tilde{\pi}_m)$ is therefore satisfied by $\{p'_0, \dots, p'_m\}$.

$\leftarrow$:
Assume $\exists\tilde{\pi}_0, \dots, \tilde{\pi}_m\ \forall\pi_0, \dots, \pi_n.\ \phi(\pi_0, \dots, \pi_n) \wedge \psi(\tilde{\pi}_0, \dots, \tilde{\pi}_m)$ is satisfiable by a trace set T. Therefore witnesses for $\tilde{\pi}_0, \dots, \tilde{\pi}_m$, namely $p'_0, \dots, p'_m \in T$, must exist and importantly ϕ holds on those traces, since it only depends on universally quantified trace variables. Choose those traces again as witnesses for $\pi'_0, \dots, \pi'_m$. Therefore the trace set $\{p'_0, \dots, p'_m\}$ satisfies both $\forall\pi_0, \dots, \pi_n.\ \phi(\pi_0, \dots, \pi_n)$ and $\exists\pi'_0, \dots, \pi'_m.\ \psi(\pi'_0, \dots, \pi'_m)$. ■

Theorem 5.16 (Equivalence of a $\forall^n$ and a $\forall^m$ HyperLTL Formula) *The question if a $\forall^n$ HyperLTL formulae $\forall\pi_0 \dots \forall\pi_n.\ \psi$ and a $\forall^m$ HyperLTL formula $\forall\pi'_0 \dots \forall\pi'_m.\ \phi$ are equivalent is decidable.*

Proof Let two HyperLTL formulae $\forall\pi_0 \dots \forall\pi_n.\ \psi$ and $\forall\pi'_0 \dots \forall\pi'_m.\ \phi$ be given. For determining if $\forall\pi_0 \dots \forall\pi_n.\ \psi$ is equivalent to $\forall\pi'_0 \dots \forall\pi'_m.\ \phi$, we will answer the equivalent question whether $\forall\pi_0 \dots \forall\pi_n.\ \psi \leftrightarrow \forall\pi'_0 \dots \forall\pi'_m.\ \phi$ is a tautology. To this end, we will check if its negation, $\neg(\forall\pi_0 \dots \forall\pi_n.\ \psi \leftrightarrow \forall\pi'_0 \dots \forall\pi'_m.\ \phi)$, is satisfiable. By splitting the equivalence into two implications, we can simplify the formula and reduce the problem to satisfiability checking for plain LTL formulae.

$$\begin{aligned}
&\neg(\forall\pi_0 \dots \pi_n.\ \psi \leftrightarrow \forall\pi'_0 \dots \pi'_m.\ \phi)\\
\overset{\text{Def.}\leftrightarrow}{\equiv}\ &\neg(((\forall\pi_0 \dots \pi_n.\ \psi) \rightarrow (\forall\pi'_0 \dots \pi'_m.\ \phi)) \wedge ((\forall\pi'_0 \dots \pi'_m.\ \phi) \rightarrow (\forall\pi_0 \dots \pi_n.\ \psi)))\\
\overset{\text{Def.}\rightarrow}{\equiv}\ &\neg((\neg(\forall\pi_0 \dots \pi_n.\ \psi) \vee (\forall\pi'_0 \dots \pi'_m.\ \phi)) \wedge (\neg(\forall\pi'_0 \dots \pi'_m.\ \phi) \vee (\forall\pi_0 \dots \pi_n.\ \psi)))\\
\overset{\text{Def.}\neg}{\equiv}\ &\neg(((\exists\pi_0 \dots \pi_n.\ \neg\psi) \vee (\forall\pi'_0 \dots \pi'_m.\ \phi)) \wedge ((\exists\pi'_0 \dots \pi'_m.\ \neg\phi) \vee (\forall\pi_0 \dots \pi_n.\ \psi)))\\
\overset{\text{De M.}}{\equiv}\ &\neg((\exists\pi_0 \dots \pi_n.\ \neg\psi) \vee (\forall\pi'_0 \dots \pi'_m.\ \phi)) \vee \neg((\exists\pi'_0 \dots \pi'_m.\ \neg\phi) \vee (\forall\pi_0 \dots \pi_n.\ \psi))\\
\overset{\text{De M.}}{\equiv}\ &(\neg(\exists\pi_0 \dots \pi_n.\ \neg\psi) \wedge \neg(\forall\pi'_0 \dots \pi'_m.\ \phi)) \vee (\neg(\exists\pi'_0 \dots \pi'_m.\ \neg\phi) \wedge \neg(\forall\pi_0 \dots \pi_n.\ \psi))\\
\overset{\text{Def.}\neg}{\equiv}\ &((\forall\pi_0 \dots \pi_n.\ \psi) \wedge (\exists\pi'_0 \dots \pi'_m.\ \neg\phi)) \vee ((\forall\pi'_0 \dots \pi'_m.\ \phi) \wedge (\exists\pi_0 \dots \pi_n.\ \neg\psi))
\end{aligned}$$

It is our goal to check this formula for satisfiability, in order to decide equivalence of the formulae. By Lemma 5.15 it is sufficient to check the following formula for satisfiability.

$$(\exists\pi'_0 \dots \pi'_m \forall\pi_0 \dots \pi_n.\ \psi \wedge \neg\phi) \vee (\exists\pi_0 \dots \pi_n \forall\pi'_0 \dots \pi'_m.\ \phi \wedge \neg\psi)$$

By applying Theorem 5.9 to each disjunct, this formula is satisfiable iff the following formula is satisfiable.

$$sp(\exists\pi'_0 \dots \pi'_m \forall\pi_0 \dots \pi_n.\ \psi \wedge \neg\phi) \vee sp(\exists\pi_0 \dots \pi_n \forall\pi'_0 \dots \pi'_m.\ \phi \wedge \neg\psi)$$

By Theorem 4.7, using Construction 4.4 this formula is satisfiable iff the following formula is satisfiable.

$$\psi_\exists(sp(\exists\pi'_0 \dots \pi'_m \forall\pi_0 \dots \pi_n.\ \psi \wedge \neg\phi)) \vee \psi_\exists(sp(\exists\pi_0 \dots \pi_n \forall\pi'_0 \dots \pi'_m.\ \phi \wedge \neg\psi))$$

Note that both disjuncts are plain LTL formula. We can therefore invoke a procedure to determine if the first or the second disjunct is satisfiable. If this is the case, then the HyperLTL formulae $\forall\pi_0 ... \forall\pi_n.\ \psi$ and $\forall\pi'_0 ... \forall\pi'_m.\ \phi$ are not equivalent. Otherwise they are. ∎

The algorithm for equivalence checking that can be inferred from this proof is implemented in the tool **EAHyper**, which is presented in Section 7.

Theorem 5.17 (Complexity) *$\forall^n \leftrightarrow \forall^m$ lies in **EXPSpace** and is **PSpace-hard**.*

Proof Let two universally quantified HyperLTL formulae $\forall\pi_0 \dots \forall\pi_n.\ \psi$ and $\forall\pi'_0 \dots \forall\pi'_m.\ \phi$ be given. By Theorem 5.16 the equivalence problem reduces to $(\exists\pi'_0 \dots \pi'_m \forall\pi_0 \dots \pi_n.\ \psi \wedge \neg\phi) \vee (\exists\pi_0 \dots \pi_n \forall\pi'_0 \dots \pi'_m.\ \phi \wedge \neg\psi)$ and can thus be solved by a Turing machine in $O(n^m + m^n)$. Therefore the problem lies in **EXPSpace**.
Hardness. We reduce from HyperLTL-SAT$^+$ for the $\forall^n$ fragment, which is **PSpace-hard** by Theorem 4.18. Given a HyperLTL formula $\forall\pi_0 \dots \forall\pi_n.\ \psi$, there exists a trace set T s.t. $T \models \forall\pi_0 \dots \forall\pi_n.\ \psi$, iff $\forall\pi_0 \dots \forall\pi_n.\ \psi \leftrightarrow \forall\pi.\ a_\pi \wedge \neg a_\pi$ is false. By Theorem 2.21 $\forall^n \leftrightarrow \forall^m$ is therefore **PSpace-hard**. ∎

Analogously to the bounded version of $\exists^n\forall^m$ HyperLTL-SAT in Section 5.3, we can define a bounded version of $\forall^n \leftrightarrow \forall^m$ as well. We will examine this problem and its complexity in the next section.

5.5 b-Bounded Equivalence of Universally Quantified HyperLTL Formulae

To limit the exponential blow up, we can define a restricted version of equivalence checking by bounding the number of universal-quantifiers. We show that this problem is **PSpace-complete**.

Definition 5.18 (b-Bounded $\forall^n \leftrightarrow \forall^m$) Given two universally quantified HyperLTL formulae $\forall\pi_0 \dots \forall\pi_{b_1}.\ \psi$ and $\forall\pi'_0 \dots \forall\pi'_{b_2}.\ \phi$. b-Bounded $\forall^n \leftrightarrow \forall^m$ is defined as the problem to determine whether $\forall\pi_0 \dots \forall\pi_{b_1}.\ \psi \leftrightarrow \forall\pi'_0 \dots \forall\pi'_{b_2}.\ \phi$, with $b_1, b_2 \in \mathbb{N}$, s.t. $b_1, b_2 \leq b$, is a tautology. □

Theorem 5.19 (Complexity) *b-Bounded $\forall^n \leftrightarrow \forall^m$ is **PSpace-complete**.*

Proof Given two universally quantified HyperLTL formulae $\forall\pi_0 \dots \forall\pi_{b_1}.\ \psi$ and $\forall\pi'_0 \dots \forall\pi'_{b_2}.\ \phi$, with $b_1, b_2 \in \mathbb{N}$, s.t. $b_1, b_2 \leq b$. The b-Bounded satisfiability problem of the $\exists^n\forall^m$ fragment lies in **PSpace** by Theorem 5.13. By Theorem 5.16 equivalence checking for the given HyperLTL formulae reduces to the satisfiability of $(\exists\pi'_0 \dots \pi'_{b_2} \forall\pi_0 \dots \pi_{b_1}.\ \psi \wedge \neg\phi) \vee (\exists\pi_0 \dots \pi_{b_1} \forall\pi'_0 \dots \pi'_{b_2}.\ \phi \wedge \neg\psi)$. Therefore b-Bounded

$\forall^n \leftrightarrow \forall^m$ lies in **PSpace** too.
Hardness. We reduce from HyperLTL-SAT for the $\forall^n$ fragment, which is **PSpace-hard** by Theorem 4.18. Given a HyperLTL formula $\forall\pi_0 \ldots \forall\pi_n.\ \psi$, there exists a trace set T, s.t. $T \models \forall\pi_0 \ldots \forall\pi_n.\ \psi$, iff $\forall\pi_0 \ldots \forall\pi_n.\ \psi \leftrightarrow \forall\pi.\ a_\pi \wedge \neg a_\pi$ is false. Note that it is irrelevant what bound was chosen beforehand, since m equals 0. By Theorem 2.21 b-Bounded $\forall^n \leftrightarrow \forall^m$ is therefore **PSpace-hard**. By Definition 2.22 b-Bounded $\forall^n \leftrightarrow \forall^m$ is **PSpace-complete**. ■

In this section we considered the satisfiability problem of HyperLTL if we face exactly one quantifier alternation, proved its decidability and provided results on the complexity of this problem. But we postponed the satisfiability$^+$ problem with one quantifier alternation starting with an universal quantifier to the next section. For a quantifier prefix of an arbitrary length, the $\exists^n\forall^m$ HyperLTL-SAT problem lies in **EXPSpace** and is **PSpace-hard**. We proved that bounding the number of universal-quantifiers establishes **PSpace-completeness**.
Last but not least we provided a proof for the decidability of equivalence checking of universally quantified HyperLTL formulae. For an arbitrary number of quantifiers the problem lies in **EXPSpace** and is **PSpace-hard**. If we bound the maximal number of universal quantifiers, the problem becomes **PSpace-complete**.

In the next section we will consider the HyperLTL-SAT problem for two quantifier alternations starting with an existential quantifier. Furthermore we continue the discussion on the HyperLTL-SAT$^+$ problem for the $\forall^m\exists^n$ fragment of the logic.

Chapter 6

The Full Logic

In this section we will consider two problems. We will first take a look at HyperLTL-SAT for a formula with at least two quantifier alternations starting with an existential quantifier. We will begin with a short intuition as to why finding a canonical model, like the ones for the fragments above, is not possible. Unfortunately it turns out that HyperLTL-SAT becomes undecidable if we face two quantifier alternation. For proving this, we will introduce the so-called **Post's Correspondence Problem (PCP)**, from which we will be reducing. After that, we will pick up the problem we postponed in Section 5.1, namely the HyperLTL-SAT$^+$ problem for a $\forall^n\exists^m$ formula. We therefore will determine the minimal fragments of HyperLTL for which the satisfiability problem is undecidable.

It is impossible to find a canonical model for the full logic, like we did in the previous chapters. With two quantifier alternations allowed, the universal quantifier has too much impact on the size of the model.

Example 6.1 (Infinite Model) Consider the following $\exists^1\forall^1\exists^1$ HyperLTL formula ranging over $\{a\}$:

$$\exists\pi_1\forall\pi\exists\pi'.\ a_{\pi_1} \tag{6.1}$$

$$\wedge\,\square(a_\pi \rightarrow \bigcirc\square\neg a_\pi) \tag{6.2}$$

$$\wedge\,\square(a_\pi \rightarrow \bigcirc\, a_{\pi'}) \tag{6.3}$$

The first subformula (6.1) states that there exists some trace in the model that starts with an a. Subformula (6.2) requires that there is at most one a in each trace of the model. Subformula (6.3) states that if a holds at any position i of any trace, there exists a trace where a holds at position $i+1$. Any model that satisfies this formula must therefore be infinite. The model satisfying this formula is the trace

	I	II	III
α	a	ab	bba
β	baa	aa	bb

Table 6.1: Example PCP Instance

set T consisting of the following traces $t_1, t_2, t_3, t_4 \dots$

$$\begin{aligned} t_1 &: \{a\}(\{\})^\omega \\ t_2 &: \{\}\{a\}(\{\})^\omega \\ t_3 &: \{\}\{\}\{a\}(\{\})^\omega \\ t_4 &: \{\}\{\}\{\}\{a\}(\{\})^\omega \\ &\dots \end{aligned}$$ □

While this is not a proof for undecidability, it gives an intuition as to why allowing more then one quantifier alternation hardens the problem. In the following we will tackle the satisfiability problem for the complete fragment and we will see that it is in fact undecidable.

6.1 Post's Correspondence Problem

In Emil L. Post's work "A Variant of a Recursively Unsolvable Problem" [30] he presents a problem consisting of two equally long lists containing finite words from an alphabet. The goal is to find an index sequence, such that the concatenation of the elements corresponding to the indices is the same for both lists.

Definition 6.2 (Posts Correspondence Problem (PCP)) Let there be two lists α and β ranging over a given alphabet Σ, with $|\alpha| = |\beta| \geq 2$. We will denote the elements with $\alpha_1, ..., \alpha_n$ and $\beta_1, ..., \beta_n$ respectively. The problem is to find an index sequence $(i_k)_{1 \leq k \leq K}$ with $K \geq 1$ and $1 \leq i_k \leq n$ for all k, such that $\alpha_{i_1} \dots \alpha_{i_K} = \beta_{i_1} \dots \beta_{i_K}$. □

Example 6.3 A possible PCP instance is shown in Table 6.1. There are two lists α and β given. A helpful intuition is to think of the columns as domino stones, where one can pick the stones I, II or III repeatedly. The goal is to build a sequence, such that the resulting word in the first line matches the word in the second line.

III	II	III	I
bba	ab	bba	a
bb	aa	bb	baa

This is a possible solution, since both the first and second line of the stones display the word *bbaabbbaa*. We can therefore find an index sequence, namely $(3, 2, 3, 1)$.

Interestingly, if a PCP instance has a solution with an index sequence is, there must be infinite solutions, since every repetition of is is a solution of the instance. We will express such a domino stone as a sequence of tuples, for example $(b, b)(b, b)(a, \#)$ represents the domino stone I, where # is a blank symbol. The first position of the tuple represents the α component and therefore the first line of the stone. Whereas the second position of the tuple represents the β component and therefore the second line of the stone.

We introduced an undecidable problem that is well-suited for reduction to other problems. Before we give the formal reduction that proves undecidability of the satisfiability problem of the complete logic, we will strengthen our intuition by translating the Example 6.3 into a $\exists^1\forall^1\exists^1$ HyperLTL formula in the next section.

6.2 Undecidability of HyperLTL-SAT with Two Quantifier Alternations

We will define some notation that helps keeping the formula readable.

Definition 6.4 (Notation) Given an arbitrary PCP instance and two atomic propositions a and $\dot{a}$. The dot marks that a finite element of α or β starts there. We will write $\tilde{a}$ if we do not care if this symbol is an a or $\dot{a}$, which alternatively can be expressed by a disjunction. We will use $*$ as notation for an arbitrary symbol of the alphabet. We assume w.l.o.g. that only singletons are allowed as elements of the trace, which could be achieved by adding the conjunction $\bigwedge_{x \in \Sigma \setminus y} \neg x$ to every atomic proposition y. We omit this and the set braces to preserve readability. □

Example 6.5 (PCP to HyperLTL-SAT) Consider the PCP instance from Example 6.3, namely: $\Sigma = \{a, b\}$. Two lists α, with $\alpha_1 = a$, $\alpha_2 = ab$ and $\alpha_3 = bba$ and β, with $\beta_1 = bba$, $\beta_2 = aa$ and $\beta_3 = bb$ (see Table 6.1). A possible solution for this PCP instance would be $(3, 2, 3, 1)$, since $\alpha_3\alpha_2\alpha_3\alpha_1 = bbaabbbaa$ and $\beta_3\beta_2\beta_3\beta_1 = bbaabbbaa$ as well.

We can reduce this PCP instance to the question whether the following HyperLTL formula is satisfiable. Let $\Sigma = (\{a, b, \dot{a}, \dot{b}\} \cup \{\#\})^2$, where the dot says that a new α_i or β_i starts there. We write $\tilde{a}$ and $*$ as syntactic sugar as defined in Definition 6.4. We use the symbol # to denote that the trace "ends" there. The elements of α will be represented in the first component of the tuple and the elements of β in the second component. The resulting HyperLTL formula is shown in Figure 6.1 and an example set satisfying this formula is shown in Figure 6.2.

The subformula (6.4) denotes that there exists a trace that starts with $(\dot{a}, \dot{a})$ or $(\dot{b}, \dot{b})$. Intuitively this means that there must exist a domino stone whose α and β component start the same, otherwise the PCP instance has no solution in the first place.

$$\exists \pi_1 \forall \pi \exists \pi'. \Bigg(\Big((\dot{a}, \dot{a})_{\pi_1} \vee (\dot{b}, \dot{b})_{\pi_1} \Big) \tag{6.4}$$

$$\wedge \Box ((\tilde{a}, \tilde{a})_{\pi_1} \vee (\tilde{b}, \tilde{b})_{\pi_1}) \mathcal{U} \Box (\#, \#)_{\pi_1} \Bigg) \tag{6.5}$$

$$\wedge \Diamond \Box (\#, \#)_{\pi} \tag{6.6}$$

$$\wedge \Bigg(\Bigg(\Big(((\dot{a}, \dot{b})_{\pi} \wedge \bigcirc (\dot{*}, a)_{\pi} \wedge \bigcirc \bigcirc (\tilde{*}, a)_{\pi} \wedge \bigcirc \bigcirc \bigcirc (\tilde{*}, \dot{*})_{\pi}) \tag{6.7}$$

$$\vee ((\dot{a}, \dot{b})_{\pi} \wedge \bigcirc (\#, a)_{\pi} \wedge \bigcirc \bigcirc (\#, a)_{\pi} \wedge \bigcirc \bigcirc \bigcirc (\#, \#)_{\pi}) \Big) \tag{6.8}$$

$$\wedge \Box (\bigcirc (\tilde{a}, *)_{\pi} \rightarrow (\tilde{a}, *)_{\pi'}) \tag{6.9}$$

$$\wedge \Box (\bigcirc (\tilde{b}, *)_{\pi} \rightarrow (\tilde{b}, *)_{\pi'}) \tag{6.10}$$

$$\wedge \Box (\bigcirc (\#, *)_{\pi} \rightarrow (\#, *)_{\pi'}) \tag{6.11}$$

$$\wedge \Box (\bigcirc \bigcirc \bigcirc (*, \tilde{a})_{\pi} \rightarrow (*, \tilde{a})_{\pi'}) \tag{6.12}$$

$$\wedge \Box (\bigcirc \bigcirc \bigcirc (*, \tilde{b})_{\pi} \rightarrow (*, \tilde{b})_{\pi'}) \tag{6.13}$$

$$\wedge \Box (\bigcirc \bigcirc \bigcirc (*, \#)_{\pi} \rightarrow (*, \#)_{\pi'}) \Bigg) \tag{6.14}$$

$$\vee \Bigg(\Big(((\dot{a}, \dot{a})_{\pi} \wedge \bigcirc (b, a)_{\pi} \wedge \bigcirc \bigcirc (\dot{*}, \dot{*})_{\pi}) \tag{6.15}$$

$$\vee ((\dot{a}, \dot{a})_{\pi} \wedge \bigcirc (b, a)_{\pi} \wedge \bigcirc \bigcirc (\#, \#)_{\pi}) \Big) \tag{6.16}$$

$$\wedge \Box (\bigcirc \bigcirc (\tilde{a}, *)_{\pi} \rightarrow (\tilde{a}, *)_{\pi'}) \tag{6.17}$$

$$\wedge \Box (\bigcirc \bigcirc (\tilde{b}, *)_{\pi} \rightarrow (\tilde{b}, *)_{\pi'}) \tag{6.18}$$

$$\wedge \Box (\bigcirc \bigcirc (\#, *)_{\pi} \rightarrow (\#, *)_{\pi'}) \tag{6.19}$$

$$\wedge \Box (\bigcirc \bigcirc (*, \tilde{a})_{\pi} \rightarrow (*, \tilde{a})_{\pi'}) \tag{6.20}$$

$$\wedge \Box (\bigcirc \bigcirc (*, \tilde{b})_{\pi} \rightarrow (*, \tilde{b})_{\pi'}) \tag{6.21}$$

$$\wedge \Box (\bigcirc \bigcirc (*, \#)_{\pi} \rightarrow (*, \#)_{\pi'}) \Bigg) \tag{6.22}$$

$$\vee \Bigg(\Big(((\dot{b}, \dot{b})_{\pi} \wedge \bigcirc (b, b)_{\pi} \wedge \bigcirc \bigcirc (a, \dot{*})_{\pi} \wedge \bigcirc \bigcirc \bigcirc (\dot{*}, \tilde{*})_{\pi}) \tag{6.23}$$

$$\vee ((\dot{b}, \dot{b})_{\pi} \wedge \bigcirc (b, b)_{\pi} \wedge \bigcirc \bigcirc (a, \#)_{\pi} \wedge \bigcirc \bigcirc \bigcirc (\#, \#)_{\pi}) \Big) \tag{6.24}$$

$$\wedge \Box (\bigcirc \bigcirc \bigcirc (\tilde{a}, *)_{\pi} \rightarrow (\tilde{a}, *)_{\pi'}) \tag{6.25}$$

$$\wedge \Box (\bigcirc \bigcirc \bigcirc (\tilde{b}, *)_{\pi} \rightarrow (\tilde{b}, *)_{\pi'}) \tag{6.26}$$

$$\wedge \Box (\bigcirc \bigcirc \bigcirc (\#, *)_{\pi} \rightarrow (\#, *)_{\pi'}) \tag{6.27}$$

$$\wedge \Box (\bigcirc \bigcirc (*, \tilde{a})_{\pi} \rightarrow (*, \tilde{a})_{\pi'}) \tag{6.28}$$

$$\wedge \Box (\bigcirc \bigcirc (*, \tilde{b})_{\pi} \rightarrow (*, \tilde{b})_{\pi'}) \tag{6.29}$$

$$\wedge \Box (\bigcirc \bigcirc (*, \#)_{\pi} \rightarrow (*, \#)_{\pi'}) \Bigg) \tag{6.30}$$

$$\vee \Box (\#, \#)_{\pi} \Bigg) \tag{6.31}$$

Figure 6.1: PCP Translation to HyperLTL

Start

$\pi_1 :\ (\dot{b}, \dot{b})(b, b)(a, \dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#)\dots$

Delete III

$\pi'_1 :\ (\dot{a}, \dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#)\dots$

Delete II

$\pi'_2 :\ (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#)\dots$

Delete III

$\pi'_3 :\ (\dot{a}, \dot{b})(\#, a)(\#, a)(\#, \#)(\#, \#)\dots$

Delete I

$\pi'_4 :\ (\#, \#)(\#, \#)\dots$

End

Figure 6.2: Trace Set Satisfying the Example Formula

The main idea of the reduction lies in subformula (6.5). We require that there exists a trace π_1 whose α and β component match at all positions. It also ensures that the trace ends synchronously with $(\#, \#)^\omega$. This is very important because it guarantees that the word constructed from the α component is equal to the word constructed from the β component. Subformula (6.6) ensures that every trace eventually ends with the termination symbol $\#$. It is important to notice here that all traces besides π_1 are allowed to end asynchronously.

The only thing that is left to ensure is that trace π_1 only consists of valid domino stones. Therefore we need to encode every valid stone in the formula. The subformula from line (6.7) to (6.14) encodes stone I. Stone II and III are encoded in (6.15) to (6.22) and (6.23) to (6.30) respectively. The stone encodings are combined by a disjunction. The first two lines of every stone encoding (for example (6.7) and (6.8)) ensure that every trace starts with a valid stone and that either the stone is followed by another stone, ensured by the dotted symbols, or the trace (possibly asynchronously) ends.

We consider stone III as an example to explain how the remaining lines, here (6.25) to (6.30), work. The idea is that every trace that starts with stone III is copied but stone III is deleted from the beginning, resulting in a new trace. Thereby, the first **three** α components and the first **two** β components of the new trace are deleted, causing asynchronous behaviour. The example set shown in Figure 6.2 shows this behaviour for π_1, which starts with stone III. By deleteing stone III from π_1 and

$$\exists \pi_1 \forall \pi \exists \pi'. \, (\bigvee_{i=1}^{n} (\dot{a}_i, \dot{a}_i)_{\pi_1}) \tag{6.32}$$

$$\wedge \, \Box(\bigvee_{i=1}^{n} (\tilde{a}_i, \tilde{a}_i)_{\pi_1}) \, \mathcal{U} \, \Box(\#, \#)_{\pi_1} \tag{6.33}$$

$$\wedge \, \Diamond \Box(\#, \#)_{\pi} \tag{6.34}$$

$$\wedge \, \Bigg(\bigg(\bigvee_{i=1}^{k} \Big(\text{StoneOrEnd}(\pi, m_{\alpha_i}, m_{\beta_i}, *) \tag{6.35}$$

$$\vee \, \text{StoneOrEnd}(\pi, m_{\alpha_i}, m_{\beta_i}, \#) \Big) \tag{6.36}$$

$$\bigwedge_{x \, \in \, \Sigma \cup \{\#\}} \Box(\bigcirc^{m_{\alpha_i}} (\tilde{x}, *)_{\pi} \rightarrow (\tilde{x}, *)_{\pi'}) \tag{6.37}$$

$$\bigwedge_{x \, \in \, \Sigma \cup \{\#\}} \Box(\bigcirc^{m_{\beta_i}} (*, \tilde{x})_{\pi} \rightarrow (*, \tilde{x})_{\pi'}) \bigg) \tag{6.38}$$

$$\vee \, \Box(\#, \#)_{\pi} \Bigg) \tag{6.39}$$

Figure 6.3: Reduction Formula ϕ_{reduc}

shifting every position accordingly, we obtain π_1'. Since π_1' starts with a valid stone, namely II (satisfying subformulae (6.15) and (6.16)), subformula (6.17) to (6.22) need to be satisfied as well. Those require that there exists another trace where stone II is deleted analogously. This argument is repeated until the copied trace is $(\#, \#)^{\omega}$ (line (6.31)), which is the only possibility for "termination" of the recursive structure of the formula and especially the only possibility that the π_1 ends synchronously with $(\#, \#)^{\omega}$. □

Generalizing this example, we can identify the minimal fragment for which HyperLTL-SAT is undecidable, which is exactly $\exists^1\forall^1\exists^1$.

Theorem 6.6 (Undecidability of HyperLTL-SAT for $\exists^1\forall^1\exists^1$) *Let a HyperLTL formula ϕ be given. Finding a set of traces $T \in 2^{TR}$, s.t. $T \models \phi$ is undecidable in general.*

Proof (Reduction from Post's Correspondence Problem) Let a PCP instance with $\Sigma = \{a_1, a_2, ..., a_n\}$ and two lists α and β be given. We denote the ith element of a list with α_i or β_i respectively. We choose our alphabet as follows: $\Sigma' = (\Sigma \cup \{\dot{a}_1, \dot{a}_2, ..., \dot{a}_n\} \cup \#)^2$. We add the corresponding dotted symbol for every symbol of

$$
\begin{aligned}
\text{StoneOrEnd}(\pi, m_{\alpha_i}, m_{\beta_i}, x) \ = \ & \\
& \textbf{if } m_{\alpha_i} \ > \ m_{\beta_i} \\
& \quad (\dot{\alpha}_i[0], \dot{\beta}_i[0])_\pi \\
& \quad \bigwedge_{j=1}^{mi_i} (\bigcirc^j (\tilde{\alpha}_i[j+1], \tilde{\beta}_i[j+1])_\pi) \\
& \quad \wedge \bigcirc^{mi_i+1} (\tilde{\alpha}_i[mi_i+2], \dot{x})_\pi \\
& \quad \bigwedge_{j=mi_i+2}^{ma_i} (\bigcirc^j (\tilde{\alpha}_i[j+1], x)_\pi) \\
& \quad \wedge \bigcirc^{ma_i+1} (\dot{x}, x)_\pi \\
& \textbf{if } m_{\beta_i} \ > \ m_{\alpha_i} \\
& \quad (\dot{\alpha}_i[0], \dot{\beta}_i[0])_\pi \\
& \quad \bigwedge_{j=1}^{mi_i} (\bigcirc^j (\tilde{\alpha}_i[j+1], \tilde{\beta}_i[j+1])_\pi) \\
& \quad \wedge \bigcirc^{mi_i+1} (\dot{x}, \tilde{\beta}_i[mi_i+2])_\pi \\
& \quad \bigwedge_{j=mi_i+2}^{ma_i} (\bigcirc^j (x, \tilde{\beta}_i[j+1])_\pi) \\
& \quad \wedge \bigcirc^{ma_i+1} (x, \dot{x})_\pi \\
& \textbf{if } m_{\alpha_i} \ = \ m_{\beta_i} \\
& \quad (\dot{\alpha}_i[0], \dot{\beta}_i[0])_\pi \\
& \quad \bigwedge_{j=1}^{mi_i} (\bigcirc^j (\tilde{\alpha}_i[j+1], \tilde{\beta}_i[j+1])_\pi) \\
& \quad \wedge \bigcirc^{ma_i+1} (\dot{x}, \dot{x})_\pi
\end{aligned}
$$

Figure 6.4: Formula Generator for a Next Stone or the End

Σ and a blank symbol "#" to encode finiteness. We will use $\tilde{a}$, $*$ and $\dot{a}$ as syntactic sugar, by Definition 6.4. Intuitively the dot marks the start of a word of α or β, i.e. of α_i or β_i. The first component of the tuple represents the word we construct from α and the second component represents the word we construct from β. Let k be the length of α and β (note that they are required to have the same length). Let m_{α_i} be the length of α_i and m_{β_i} the length of β_i. We define $\bigcirc^0(\phi)$ as ϕ and $\bigcirc^n(\phi)$ as $\bigcirc\bigcirc^{n-1}(\phi)$. Let $mi_i := min\{m_{\alpha_i}, m_{\beta_i}\}$ and $ma_i := max\{m_{\alpha_i}, m_{\beta_i}\}$. The resulting formula ϕ_{reduc} can be found in Figure 6.3 and the formula generator in Figure 6.4, which expresses that either a new stone starts or the trace comes to an end. The call of the formula generator "StoneOrEnd" in line (6.35) and (6.36) generalizes the idea displayed in the first two lines of all the stone encodings of the example in Figure 6.1 ((6.7) & (6.8), (6.15) & (6.16), (6.23) & (6.24)), stating that either a valid stone or the end follows. ϕ_{reduc} furthermore states that there exists a trace that starts dotted and whose α and β components are equal at every position and that ends synchronously with $(\#, \#)^\omega$ ((6.32) to (6.34)). To ensure that the witness of π_1 only consists of valid stones, lines (6.37) and (6.38) state that for every trace there exists a shifted trace with the first stone deleted.

Correctness. We prove correctness of the reduction by showing that if there exists a solution, namely an index sequence $i(k)$ with $k \in \mathbb{N}$, for a PCP instance, then there exists a trace set T satisfying the resulting formula ϕ_{reduc} and vice versa.

$\rightarrow$:
Assume there exists a solution i to the given PCP instance. We can construct a trace set T by building the trace $(i[0], i[0]) \dots (i[k], i[k])(\#, \#)^\omega$, denoted by t_0 and adding a dot to the symbol corresponding to the new stones start. We can infer the correct placement of the dots from the solution. Let the solution i start with index j, $|\alpha_j|$, denoting the length, be m_a, $|\beta_j|$ be m_b. We distinguish two cases. If the solution is only of length 1, we add $(\#, \#)^\omega$ to T and constructed successfully a trace set satisfying the formula. Otherwise we also add one of the following trace t_1 based on t_0 to T:

$$\begin{aligned}
&\text{if } m_a = m_b : (i[m_a], i[m_b]) \dots (i[k], i[k])(\#, \#)^\omega \\
&\text{if } m_a < m_b : (i[m_a], i[m_b]) \dots (i[k], i[k - m_b + m_a]) \dots (\#, i[k])(\#, \#)^\omega \\
&\text{if } m_a > m_b : (i[m_a], i[m_b]) \dots (i[k - m_a + m_b], i[k]) \dots (i[k], \#)(\#, \#)^\omega
\end{aligned}$$

We repeat adding traces t_n based on every newly added trace t_{n-1} until we terminate with $(\#, \#)^\omega$. Note that t_{n-1} might already end asynchronously. By construction this is exactly a trace set T satisfying ϕ_{reduc}.

$\leftarrow$:
Let the formula ϕ_{reduc} be satisfiable by a trace set T. Therefore, there exists a witness

t_1 for π_1, which starts with a dot, whose α and β components are the same at all positions and which ends **synchronously** with $(\#, \#)^\omega$. t_1 also needs to start with a valid stone, which is ensured by "StoneOrdEnd", since otherwise $t_1 \notin T$. By construction there exists a subset $T_{\text{min}} \subseteq T$ that satisfies ϕ_{reduc}, which contains t_1 and every trace constructed by deleting one stone after another, with the last trace being $(\#, \#)^\omega$. This is a valid solution since we delete valid stones only finitely often, because eventually t_1 needs to terminate synchronously with $(\#, \#)^\omega$, ensuring that the solution remains finite. We define a total order for the traces in T_{min} according to the number of dots or, equivalently, the number of stones. We also define a function s that maps traces to the index of their starting stone. Let $A = [t_1, t_2, \ldots, t_n]$ be the list of traces in T_{min} sorted in descending order. A possible solution for the PCP instance is the index sequence $s(t_1)\ s(t_2) \ldots s(t_n)$.

Therefore the minimal fragment of HyperLTL that is undecidable is $\exists^1\forall^1\exists^1$. ■

In this section we presented a reduction from Post's correspondence problem to a HyperLTL-SAT formula with at least two quantifier alternations starting with an existential quantifier. We exploited the $\forall$-$\exists$ structure of the formula to establish an inductive argument, by constructing the model step-by-step from the PCP solution, respectively deconstructing the trace set step-by-step to obtain the solution for the PCP instance. Interestingly the same argument holds for the HyperLTL-SAT$^+$ problem for $\forall^n\exists^m$ formulae, which we will discuss in the next section.

6.3 Undecdiability of HyperLTL-SAT$^+$ with One Quantifer Alternation

We postponed the HyperLTL-SAT$^+$ problem to this section, since we can now use the reduction above to show its undecidability.

Definition 6.7 (Undecidability of HyperLTL-SAT$^+$ for $\forall^1\exists^1\exists^1$) Let a HyperLTL formula ϕ be given. Finding a **non-empty** set of traces $T \in 2^{TR}$, s.t. $T \models \phi$ is undecidable in general. □

Proof (Reduction from Post's Correspondence Problem) We use a slightly modified version of the reduction in the previous section, which is shown in Figure 6.5. Only the quantifier prefix has changed. Since we now require that our model contains at least one trace, there exists a witness for π_1. With the argument of the proof in Section 6.2, the HyperLTL-SAT$^+$ problem is undecidable too. ■

It is important to note that the minimal fragment for which the HyperLTL-SAT$^+$ problem is undecidable is in fact $\forall^1\exists^1$. One can use Construction 4.4 to encode the complete reduction given above in this fragment by zipping the existentially quantified traces together.

$$
\begin{aligned}
\forall \pi \exists \pi_1 \exists \pi' .\ & (\bigvee_{i=1}^{n} (\dot{a}_i, \dot{a}_i)_{\pi_1}) \\
& \wedge \Box (\bigvee_{i=1}^{n} (\tilde{a_i}, \tilde{a_i})_{\pi_1}) \,\mathcal{U}\, \Box (\#, \#)_{\pi_1} \\
& \wedge \Diamond \Box (\#, \#)_{\pi} \\
& \wedge \Bigg(\Bigg(\bigvee_{i=1}^{k} \Big(\mathrm{StoneOrEnd}(\pi, m_{\alpha_i}, m_{\beta_i}, *) \\
& \qquad\qquad \vee \mathrm{StoneOrEnd}(\pi, m_{\alpha_i}, m_{\beta_i}, \#) \Big) \\
& \qquad \bigwedge_{x \,\in\, \Sigma \cup \{\#\}} \Box (\bigcirc^{m_{\alpha_i}} (\tilde{x}, *)_{\pi} \rightarrow (\tilde{x}, *)_{\pi'}) \\
& \qquad \bigwedge_{x \,\in\, \Sigma \cup \{\#\}} \Box (\bigcirc^{m_{\beta_i}} (*, \tilde{x})_{\pi} \rightarrow (*, \tilde{x})_{\pi'}) \Bigg) \\
& \qquad \vee \Box (\#, \#)_{\pi} \Bigg)
\end{aligned}
$$

Figure 6.5: Reduction Formula $\phi_{\text{reduc}^{+}}$

Chapter 7

Implementation & Experimental Results

We implemented the results presented in this thesis in the prototype **EAHyper** (Equivalence Checker of Universally Quantified HyperLTL Formulae). The tool can be downloaded here: EAHyper [1] [1].

The implementation is based on the satisfiability checker for LTL called **pltl** [37], which is used in the portfolio LTL-SAT checker **polsat** [27]. Pltl is tableau-based and works best for this approach according to the experiments presented in [36].

EAHyper is implemented in Python (Version 3.4.3) [43]. The corresponding python script can be found in Appendix A. According to the results presented in Sections 4.1, 4.2 and 5.2, the tool syntactically reduces a given $\exists^n$, $\forall^n$ or $\exists^n\forall^m$ formula to its corresponding, equisatisfiable LTL formula, using the constructions presented. For a $\forall^n$ formula, the HyperLTL-SAT$^+$ problem is solved. Pltl is invoked as the decision procedure for the satisfiability problem of the reduced LTL formula to solve the satisfiability problem of the original formula. EAHyper can, therefore, solve the satisfiability$^+$ problem for every decidable fragment of HyperLTL.

The equivalence check for universally quantified HyperLTL formulae, presented in Section 5.4, as well as an implication check is also implemented in EAHyper.

7.1 Experimental Results

We present our experimental results in this section. For the purpose of evaluating the first basic prototype implementation, we generated over 300.000 HyperLTL formulae. We ran the experiments on an Intel Core I5-3230M [3] machine running Arch Linux with 4GB RAM.

[1] An installation and usage guide, including the input syntax, can be found in the corresponding README file.

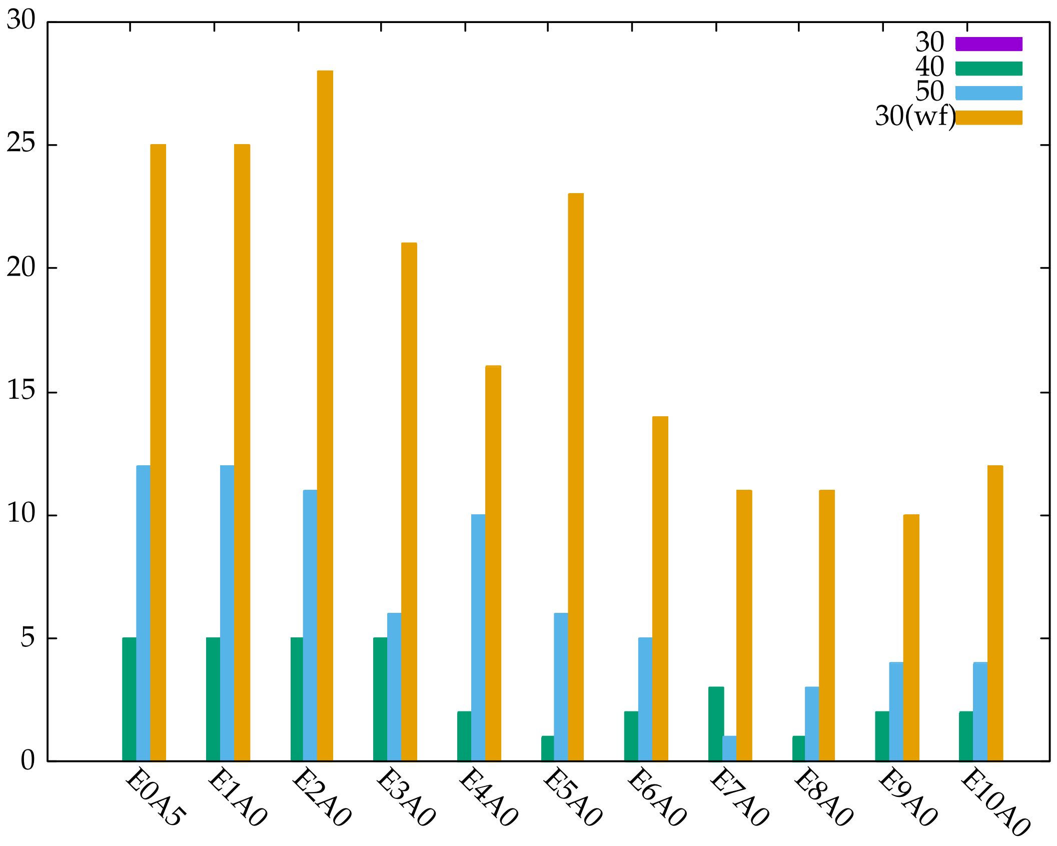

Figure 7.1: Number of Timeouts for Alternation-free Instances of 5000 Formulae

The experiments, including more test cases, the results and plots, can be downloaded here: EAHyper Experiments [2]. Each instance consists of 5000 formulae, classified by "formulae depth" of the underlying LTL formula generated using "randltl" [18], as well as quantifier shape. For example "E3A2" denotes a formula with a quantifier prefix of the form $\exists\pi\exists\pi'\exists\pi''\forall\tilde{\pi}\forall\tilde{\pi}'$.

We used the same seed for generating the underlying LTL formulae for every instance, but assigned the path variables pseudo randomly to the atomic propositions. Note that there is no guarantee that every path variable will be assigned.

We also added experiments considering the so-called **weak fairness** condition [40], denoted by wf. Weak fairness ensures that every atomic proposition a keeps occurring by adding the conjunct $\wedge\Box(\Diamond a)$, for all $a \in AP$. We fixed the number of atomic propositions that can occur in a formula to five and set a time out for a computation of a single formula to 10 minutes. To provide intuition on the formula sizes, representative formulae of some experiment instances are given in Appendix C.

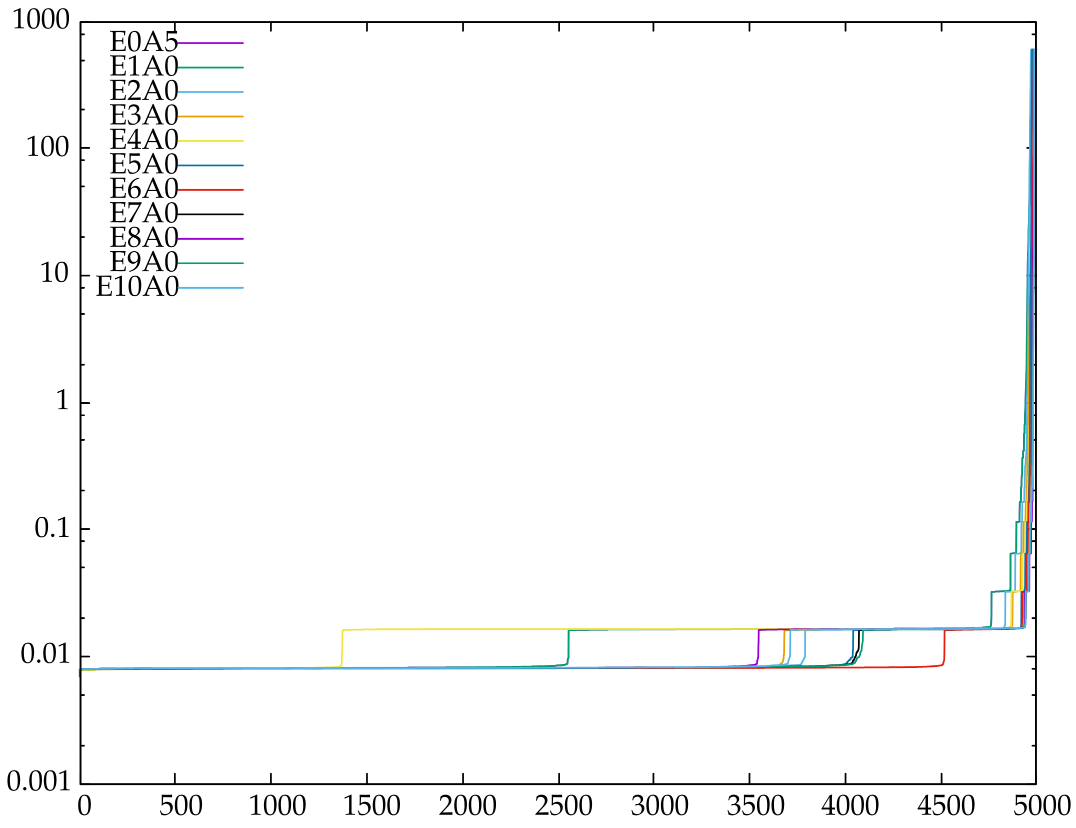

Figure 7.2: Sorted Runtime of the Alternation-free Fragment for Depth 30 (Including Weak Fairness) with a logscaled y-axis

7.1.1 Alternation-free Fragment

We first test the basic performance of the tool for alternation-free HyperLTL formulae shown in Figure 7.1. EAHyper reduces the HyperLTL-SAT problem to LTL-SAT by manipulating the input string of the formulae and invoking the above mentioned tool “pltl” as a decision procedure. All experiment instances are displayed as a bar, where one instance consists of 5000 HyperLTL formulae. The y-axis represents the number of formulae that could not be solved within the given time limit of ten minutes. The x-axis represents the different quantifier prefixes. The tool did not timeout for any formula of depth 30, solving every instance in about 50 seconds. In contrast, adding weak fairness conditions to the formula hardens the problem. Corresponding to the results presented in Section 4.2, the tool solves an instance only containing universally quantified HyperLTL formulae exactly as fast as the corresponding instance containing formulae, which are quantified by a singly existential quantifier. Note that the underlying LTL formulae are the same, which is

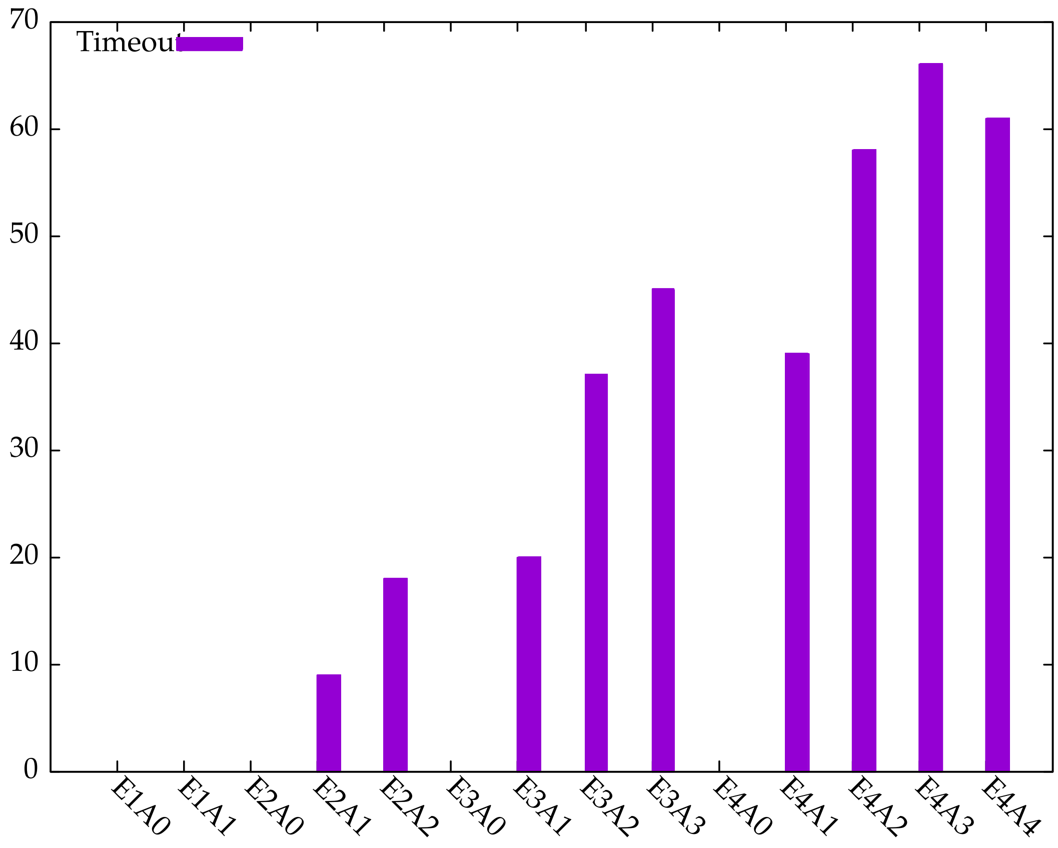

Figure 7.3: Number of Timeouts for One Alternation of Depth 30

ensured by using the same seed for its generation. Interestingly, the tool handled formulae with higher depth better when more existential quantifiers are added. By adding such quantifiers, the tool interprets every atomic proposition with its corresponding trace variable as a fresh one (see Section 4.1). Intuitively, by adding an existential quantifier, it provides a bigger "solution space" and, therefore, giving the tool more possibilities to find a solution faster.

Figure 7.2 shows the runs of the instances for depth 30, including weak fairness conditions, in detail. The tool solves the majority of them in under 0.01 seconds. Which shows that for the comparison of the alternation-free instances, the computation time for the solvable formulae is in fact negligible to the amount of time EAHyper needs if it time outs. Overall, the tool performs surprisingly well even on non trivial instances that respect weak fairness. The number of universal quantifiers makes no difference for the computation time and adding existential quantifiers (and therefore also adding atomic propositions) even softens the problem. We can therefore observe that the only dimension that hardens the problem, besides

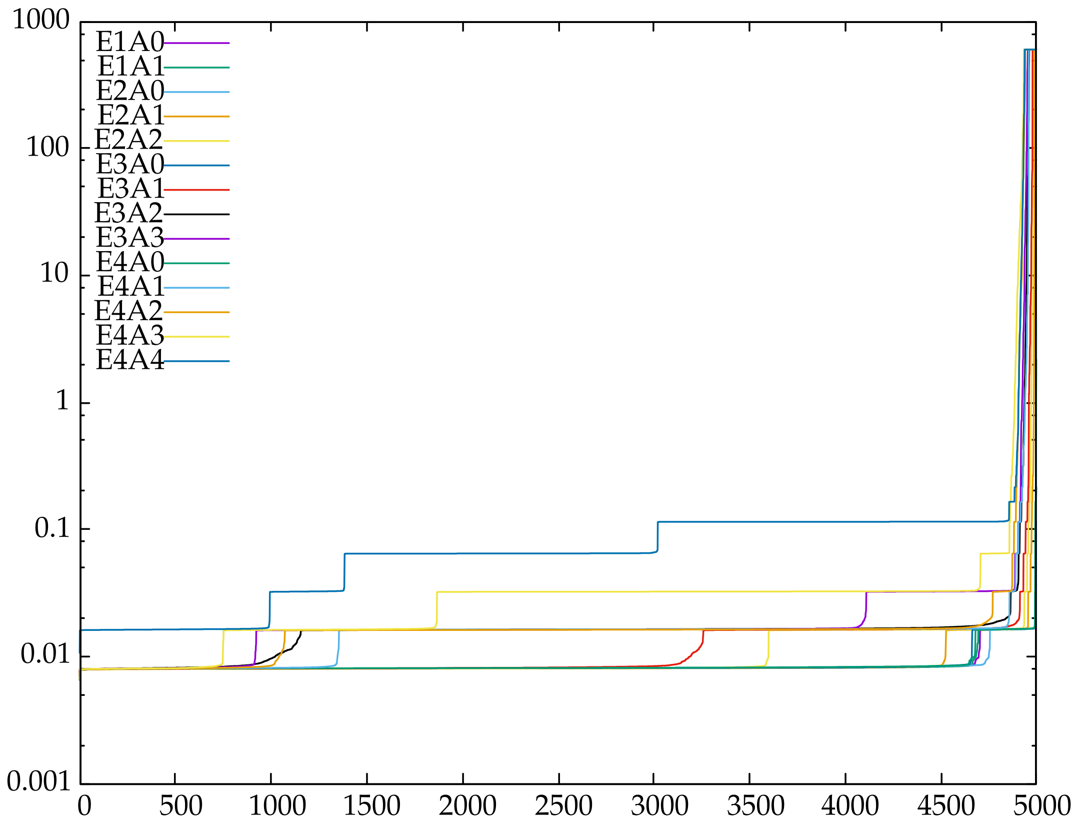

Figure 7.4: One Alternation for Depth 30 with a logscaled y-axis

adding weak fairness conditions, is in fact the size of the formula.

We will consider the fragment of HyperLTL with one quantifier alternation, starting with an existential quantifier, in the next subsection.

7.1.2 One Quantifier Alternation

As expected, a quantifier alternation has a huge impact on the computation time of the tool. We ran experiments on every decidable quantifier alternation up to four existential and four universal quantifiers. The timeouts are shown in Figure 7.3. There is a blow up, if we add a single universal quantifier to a plain existentially quantified formula. For example, the gap between "E4A0" and "E4A1". The experiment instance "E4A0" terminates in under 48 seconds, but "E4A1" terminates in 6 hours and 42 minutes while timing out 39 times. Interestingly, adding additional universal quantifiers does not result in a blow up. The computation of the instance "E4A2" took 10 hours and 14 minutes, timing out 58 times. This corresponds to the results presented in Section 5.2. Since for an additional universal quantifier, the tool has to compute every permutation of the existential quantifiers with a cardinality increased by one.

Figure 7.4 shows the runs of the considered instances in detail. Note that the tool solves experiment instances of depth 30 without an alternation in under one minute. As expected, the tool times out more often even for a smaller quantifier prefix, but is still able to solve the majority of the problems in a fast manner. Furthermore, as already discussed above, adding the weak fairness condition to the formula hardens the problem. We ran an experiment instance with depth 30, weak fairness, two existential quantifiers and one universal quantifier. Whereas the instance without the weak fairness condition took one hour and 32 minutes to compute, timing out 9 times, the corresponding instance with the weak fairness condition took 27 hours and 48 minutes, timing out 160 times. Note that in our experiment results, the trace variables were randomly assigned. Therefore the weak fairness condition might demand that the atomic proposition is reoccurring in every trace. See Appendix C for an intuition on the formula structure and size.

This follows the discussed trend in Subsection 7.1.1, which is not surprising, since the tool reduces a formula with one quantifier alternation to a plain LTL formula. As a further experiment, we considered an implication check of two universally quantified HyperLTL formulae of a non trivial size in the next subsection.

7.1.3 Implication Check for Quantitative Noninterference

Quantitative Noninterference [12] is a information flow policy that permits information leakage at restricted rates. Smith [39] presented a way to measure such information leakage of a system with the concept of **min-entropy**. This quantifies the amount of information that might be gained given the answer to a single guess about high security information. Clarkson, Finkbeiner et al. [16] present a way to formalize the so-called **bounding-problem** [44] for min-entropy in HyperLTL. This problem asks whether the amount a system might leak is bounded by a constant c. If we consider a deterministic system and assume the secret information is uniformly distributed throughout this system, the bounding problem reduces, accordingly to Smith [39] to the question if there are no $2^c + 1$ distinguishable traces for a **low-security** user. Let I be the low inputs and O be the low outputs of the system.

$$\neg\exists\pi_0 \dots \exists\pi_{2^c}. \, (\bigwedge_i \Box(\bigwedge_{in \in I} in_{\pi_i} \leftrightarrow in_{\pi_0})) \wedge \bigwedge_{i \neq j} \Box(\bigwedge_{out \in O} out_{\pi_i} \not\leftrightarrow out_{\pi_j})$$

Intuitively this states that there are not enough low distinguishable traces, such that an observer could possibly infer enough information. Note that this is not yet a proper HyperLTL formula, which can be fixed by pushing the negation in.

$$\forall \pi_0 \ldots \forall \pi_{2^c}.\ \neg((\bigwedge_{i} \Box(\bigwedge_{in \in I} in_{\pi_i} \leftrightarrow in_{\pi_0})) \wedge \bigwedge_{i \neq j} \Box(\bigwedge_{out \in O} out_{\pi_i} \not\leftrightarrow out_{\pi_j}))$$

We ran an experiment whether the corresponding HyperLTL formula, where c was chosen as 1 denoted by $\mathcal{B}_1$, implies the HyperLTL formula, where the constant was chosen as 2 denoted by $\mathcal{B}_2$. We have chosen I and O as singletons. Stating that if there are no three low distinguishable traces, then there are also no five low distinguishable traces. The tool reduces this question to an LTL-SAT problem, corresponding to the results presented in Section 5.4. We denote the resulting LTL formula by $\mathcal{I}$, which is 147 times bigger then $\mathcal{B}_2$. To provide an intuition on the formula size, the HyperLTL formulae $\mathcal{B}_1$ and $\mathcal{B}_2$ can be found in Appendix B, whereas $\mathcal{I}$ may be downloaded [2] because of its size.

The effort of solving this question grows exponentially with our approach. Whereas the satisfiability$^+$ problem for $\mathcal{B}_1$ and $\mathcal{B}_2$ are solved in 0.2 and 0.6 milliseconds respectively, solving the satisfiability problem of $\mathcal{I}$ took 10 hours and 18 minutes, returning that $\mathcal{B}_1$ implies $\mathcal{B}_2$. This is, considering the size of the LTL formula $\mathcal{I}$, still a reasonable amount of computation time for this non trivial problem instance.

In this section, we provided experimental results for the first prototype implementation of a satisfiability checker for HyperLTL. Since the instances of the alternation-free fragment are nearly instantly reduced to a plain LTL-SAT instance, the tool performs as expected. We unrolled the universal quantifiers in our approach to decide the fragment with one quantifier alternation. Because of this, we observed that the effort of solving those problem instances grew enormously. Small problem instances are still solved in a fast manner. We also considered an implication of two universally quantified HyperLTL formula of a bigger size, showing that the presented approach is capable of solving problems in practice.

Chapter 8

Conclusion

We will briefly sum up the content of this thesis and discuss open questions that are promising research topics.

8.1 Summary

In this thesis, we completely solved the satisfiability and satisfiability$^+$ problem for a newly-introduced logic, called HyperLTL. The results are shown in Table 8.1. We presented the necessary preliminaries on computability and complexity theory in Chapter 2. In Chapter 3, we introduced LTL as a logic to express traceproperties before introducing HyperLTL as a powerful logic that not only subsumes LTL, but is also able to express hyperproperties. We stated the difference between a trace property and a hyperproperty. To this end we provided the example of observational determinism.

We considered the alternation-free fragment of HyperLTL in Chapter 4 and proved its reducability to plain LTL. Interestingly, the satisfiability and respectively the satisfiability$^+$ problem has the same complexity as the satisfiability problem for plain LTL. In Section 4.1 and 4.2 we showed that the question of satisfiability for the alternation-free fragments is decidable and is **PSpace-complete**.

	$\exists^n$	$\forall^m$	$\forall^m\exists^n$	$\exists^n\forall^m$	b-Bounded $\exists^n\forall^m$	$\exists^n\forall^m\exists^k$
SAT	**PSpace-complete**	trivial	trivial	**EXPSpace PSpace-hard**	**PSpace-complete**	undec.
SAT$^+$	**PSpace-complete**	**PSpace-complete**	undec.	**EXPSpace PSpace-hard**	**PSpace-complete**	undec.

Table 8.1: Complexity of the Presented Satisfiability Results

$\forall^n\forall^m$ Equivalence Checking	b-Bounded $\forall^n\forall^m$ Equivalence Checking
EXPSpace **PSpace-hard**	**PSpace-complete**

Table 8.2: Complexity of Equivalence Checking for Universally Quantified HyperLTL Formulae

We tackled the satisfiability problem for the HyperLTL fragment with one quantifier alternation in Chapter 5. We solved the problem by unrolling all universal quantifiers and proved its **PSpace-hardness** in Section 5.2. To restrict the exponential blow up involved in this approach, we introduced a bounded version of the problem in Section 5.3 that restricts the number of universal quantifiers in an $\exists^n\forall^m$ formula and proved its **PSpace-completeness**. As an application of this result, we provided an algorithm for deciding whether two universally quantified HyperLTL formulae are equivalent in Section 5.4, which lies in **EXPSpace** and is **PSpace-hard**. This allows checking formulated specifications for redundancy, which is a desirable application, since a lot of verification algorithm scale in the size of the formalized specification. In Section 5.5, we defined a bounded version of the problem and proved its **PSpace-completeness**. The results are shown in Figure 8.2.

While the satisfiability problem of the $\forall^m\exists^n$ fragment has a trivial solution, we had to postpone the satisfiability$^+$ problem to Chapter 6, where we identified the minimal fragments for which the satisfiability as well as the satisfiability$^+$ problem is undecidable, which is $\exists^1\forall^1\exists^1$ and $\forall^1\exists^1$ respectively. We achieved this via a reduction from Post's correspondence problem.

For reasoning on a reactive system (e.g. by model checking) or constructing a correct reactive system (e.g by synthesis), writing consistent and compact, yet expressive, properties is highly desirable. HyperLTL manages to express those specifications in an intuitive manner. With bigger systems and bigger specifications that a system must satisfy, it is important to sanity check the formalized specifications. For this purpose we implemented the results presented in this thesis in the satisfiability checker "EAHyper", which was presented in Chapter 7. Experimental results were provided in Section 7.1. We observed that the tool is capable of checking satisfiability of a formula in a consistent and fast manner. Also checking implications of HyperLTL formulae of a non trivial size could be done in a reasonable amount of time. This establishes the possibility of not only sanity checking a formalized security policy but also finding minimized formulae in practice.

8.2 Future Work

The results of this thesis suggest future work on different topics. It remains open if $\exists^n\forall^m$ HyperLTL-SAT is either **EXPSpace-hard** or in **PSpace**, which actually involves the question whether an $\exists^n\forall^m$ HyperLTL formula might be reduced in polynomial space to an $\exists^n\forall^2$ HyperLTL formula. On the other hand an **EXPSpace-hardness** proof via a reduction from an **EXPSpace Turing machine** is reasonable as well, since it also includes the desired $\exists$-$\forall$ structure. Further research on this topic may close this complexity gap.

Since every Kripke structure for a given **finite** alphabet is recursively enumerable, HyperLTL-SAT might in fact be semi decidable. Based on the concept of "Bounded Synthesis" by Schewe and Finkbeiner [35], one might be able to find a feasible solution for a b-Bounded version of HyperLTL-SAT. Instead of modelling the system as its set of execution traces, one can use a Kripke structure and preserve decidability by bounding its state space by a natural number b. With its relation to synthesis, solving b-Bounded HyperLTL-SAT (respectively HyperCTL*-SAT) would enable the possibility of synthesizing reactive systems that in fact are robust against information leakage.

Promising future work lies in the field of epistemic logics [20], which are also able to express interesting information flow policies [7]. The relation between HyperLTL and such logics, which are based on a "Knowledge"-Operator, is not yet fully investigated.

Future work might also be invested in the prototype implementation **EAHyper**. Until now, the tool only returns either "Sat" or "Unsat" ("Equivalent" or "Not Equivalent" respectively), whereas returning the model is a desirable extension. Running experiments with different LTL-SAT solvers than "pltl" and comparing their results could lead to a better performance. Based on a possible solution for b-Bounded HyperLTL-SAT, the tool might also be extended to solve the bounded satisfiability problem for the full logic.

Overall, exploiting this very powerful formalism of HyperLTL provided interesting results. The recent work on model checking [21] as well as this thesis on satisfiability show that this approach of quantifying over execution traces is up-and-coming. Providing feasible solutions for practice will be a challenging, but nevertheless a promising task for further research.

Appendices

Appendix A

Code Listing

```
import os
import sys

#splits the quantifiers and the formula body
def readExistsForall(ins):
        return (ins.split("."))

#checks for consistency of the input
#raises error if forall exists prefix
#returns the number of existential quantifier
def checkConsistency (li):
        a = False
        numberE = 0
        for i in range(len(li)-1):
                if a==False and li[i][0]=="e":
                        numberE = numberE + 1
                        continue
                if a==False and li[i][0]=="f":
                        numberE = i
                        a = True
                        continue
                if a==True and li[i][0]=="e":
                        raise NameError('Only
                        exists forall allowed')
        return (numberE)
```

```
#gets the variables of the quantifier prefix
def getExistsAndForallLists (sp):
        #checks for consistency
        numberExists = checkConsistency(sp)
        ql = len(sp) −1
        prefix = sp[:ql]
        varlist = [i.split(' ')[1] for i in prefix]
        existslist = varlist[:numberExists]
        foralllist = varlist[numberExists:]
        return(existslist, foralllist)

#get every selection
def xselections(items, n):
        if n==0: yield []
        else:
                for i in range(len(items)):
                        for ss in xselections(items,
                         n−1):
                                yield [items[i]]+ss

#replace the universal quantified trace variables
#with existential quantifiers
def repl (s, exlist, alist):
        if len(alist) == 0:
                return s
        for i in range(len(exlist)):
                re = s.replace(alist[0], exlist[0])
                tailexlist = exlist[1:]
                tailalist = alist[1:]
                return repl (re, tailexlist,
                 tailalist)

#calls "pltl" as a decision procedure
#returns unsat or sat respectively
def callDecProc (originstring):
        sp = readExistsForall(originstring)
        ql = len(sp) −1
        ea = getExistsAndForallLists (sp)
        numberExists = len(ea[0])
        numberForall = len(ea[1])
```

```
if (numberForall == 0):
        f = open('formulae_temp_exists.txt','w')
        f.write(sp[q1])
        f.close()
        p = os.popen("cat formulae_temp_exists.txt
        | ./pltl tree beverbose","r")
        while 1:
                line = p.readline()
                if not line: break
                print(line)
        return
else:
        if(numberExists == 0):
                newf = sp[ql]
                for i in ea[1]:
                        newf = newf.replace(i,
                        "")
                        newf = newf.replace("_",
                        "")
                f = open('formulae_temp_forall.txt','w')
                f.write(newf)
                f.close()
                p = os.popen("cat formulae_temp_forall.txt
                | ./pltl tree beverbose","r")
                while 1:
                        line = p.readline()
                        if not line: break
                        print(line)
                return
#compute the every possibilty
#for the replacement of the
#universally quantified path variables
perm = xselections(ea[0], numberForall)
unsat = False
first = True
finalFormula = ""
for i in (perm):
        if first:
                finalFormula = finalFormula+"("+
                repl(sp[ql],i,ea[1])+")"
```

```
                        first = False
                else:
                        finalFormula = finalFormula+" & "+"("+
                        repl(sp[ql],i,ea[1])+")"
        f = open('formulae_temp_exists_forall.txt','w')
        f.write(finalFormula)
        f.close()
        p = os.popen("cat formulae_temp_exists_forall.txt
        | ./pltl tree beverbose","r")
        while 1:
                line = p.readline()
                if "not" in line:
                        unsat = True
                if not line: break
                print (line)
        if unsat:
                print("Overall: unsatisfiable")
                return unsat
        print("Overall: satisfiable")
        return unsat


#negates a formula
def negateFormula (formula):
        ret = "~("+formula+")"
        return ret

#conjunct two formulae
def conjunctTwoFormulae (formula1, formula2):
        ret = "("+formula1+")"+"&"+"("+formula2+")"
        return ret

#reduces the equivalence problem to LTL-SAT
def reduceEquivToLTL (formula1, formula2, flag):
        sp1 = readExistsForall (formula1)
        sp2 = readExistsForall (formula2)
        ql1 = len(sp1) - 1
        ql2 = len(sp2) - 1
        ea1 = getExistsAndForallLists (sp1)
        ea2 = getExistsAndForallLists (sp2)
```

```
numberExists1 = len(ea1[0])
numberForall1 = len(ea1[1])
numberExists2 = len(ea2[0])
numberForall2 = len(ea2[1])

if (numberExists1 > 0) | (numberExists2 > 0):
        raise NameError("Only universal
                quantification allowed")

newformulaOne = conjunctTwoFormulae(sp1[ql1],
                negateFormula(sp2[ql2]))
newformulaTwo = conjunctTwoFormulae(sp2[ql2],
                negateFormula(sp1[ql1]))

permOne = xselections(ea2[1], numberForall1)
permTwo = xselections(ea1[1], numberForall2)
unsat1 = False
first1 = True
first2 = True
finalFormula1 = ""
finalFormula2 = ""
for i in (permOne):
        if first1:
                finalFormula1 = finalFormula1 +
                "("+ repl(newformulaOne,i,
                ea1[1])+")"
                first1 = False
        else:
                finalFormula1 = finalFormula1 +" & " +
                "(" + repl(newformulaOne,i,ea1[1]) +")"
f = open('formulae_temp.txt','w')
f.write(finalFormula1)
f.close()
p = os.popen("cat formulae_temp.txt
        | ./pltl tree beverbose","r")
while 1:
        line = p.readline()
        if "not" in line:
                unsat1 = True
        if not line: break
        print (line)
if unsat1:
        print("Overall first
        disjunct: unsatisfiable")
        if(flag):
                print("The first formula
                does imply the second")
                return True
if not unsat1:
        print("The first formula
        does not imply the second.")
        return False

unsat2 = False
for i in (permTwo):
        if first2:
                finalFormula2 = finalFormula2 +
                "(" + repl(newformulaTwo,i,
                ea2[1])+ ")"
                first2 = False
        else:
                finalFormula2 = finalFormula2 +" & " +
                "("+repl(newformulaTwo,i,ea2[1])+")"
f = open('formulae_temp2.txt','w')
f.write(finalFormula2)
f.close()
p = os.popen("cat formulae_temp2.txt
| ./pltl tree beverbose","r")
while 1:
        line = p.readline()
        if "not" in line:
                unsat2 = True
        if not line: break
        print (line)
if unsat2:
        print("The formulae are equivalent.")
if not unsat2:
        print("The formulae are
        not equivalent.")
        return False
```

```python
if len(sys.argv) == 2:
        callDecProc(sys.argv[1])
if len(sys.argv) == 3:
        reduceEquivToLTL(sys.argv[1],sys.argv[2],False)
if len(sys.argv) == 4:
        reduceEquivToLTL(sys.argv[1],sys.argv[2],True)
if len(sys.argv) > 5:
        raise NameError("To many arguments!")
```

Appendix B

Bounding Problem for Quantitative Noninterference

We formalized the bounding problem for quantitative noninterference from Subsection 7.1.3. Where $in \in I$ is the only low security input and $out \in O$ is the only low security output. The corresponding reduced LTL formula $\mathcal{I}$ may be downloaded [2], because of its size.

$\mathcal{B}_1$

$\forall q_0 \forall q_1 \forall q_2 . \neg(((\Box(in_{q_0} \leftrightarrow in_{q_0}))$
$\wedge (\Box(in_{q_1} \leftrightarrow in_{q_0})) \wedge (\Box(in_{q_2} \leftrightarrow in_{q_0})))$
$\wedge (\Box(\neg(out_{q_0} \leftrightarrow out_{q_1}))) \wedge (\Box(\neg(out_{q_0} \leftrightarrow out_{q_2}))) \wedge (\Box(\neg(out_{q_1} \leftrightarrow out_{q_2}))))$

$\mathcal{B}_2$

$\forall p_0 \forall p_1 \forall p_2 \forall p_3 \forall p_4 .$
$\neg(((\Box(in_{p_1} \leftrightarrow in_{p_0})) \wedge (\Box(in_{p_2} \leftrightarrow in_{p_0})) \wedge (\Box(in_{p_3} \leftrightarrow in_{p_0})) \wedge (\Box(in_{p_4}$
$\leftrightarrow in_{p_0}))) \wedge (\Box(\neg(out_{p_0} \leftrightarrow out_{p_1}))) \wedge (\Box(\neg(out_{p_0} \leftrightarrow out_{p_2}))) \wedge (\Box(\neg(out_{p_0}$
$\leftrightarrow out_{p_3}))) \wedge (\Box(\neg(out_{p_0} \leftrightarrow out_{p_4}))) \wedge (\Box(\neg(out_{p_1} \leftrightarrow out_{p_0}))) \wedge (\Box(\neg(out_{p_1}$
$\leftrightarrow out_{p_2}))) \wedge (\Box(\neg(out_{p_1} \leftrightarrow out_{p_3}))) \wedge (\Box(\neg(out_{p_1} \leftrightarrow out_{p_4}))) \wedge (\Box(\neg(out_{p_2}$
$\leftrightarrow out_{p_0}))) \wedge (\Box(\neg(out_{p_2} \leftrightarrow out_p1))) \wedge (\Box(\neg(out_{p_2} \leftrightarrow out_{p_3}))) \wedge (\Box(\neg(out_{p_2}$
$\leftrightarrow out_{p_4}))) \wedge (\Box(\neg(out_{p_3} \leftrightarrow out_{p_0}))) \wedge (\Box(\neg(out_{p_3} \leftrightarrow out_{p_1}))) \wedge (\Box(\neg(out_{p_3}$
$\leftrightarrow out_{p_2}))) \wedge (\Box(\neg(out_{p_3} \leftrightarrow out_{p_4}))) \wedge (\Box(\neg(out_{p_4} \leftrightarrow out_{p_0}))) \wedge (\Box(\neg(out_{p_4}$
$\leftrightarrow out_{p_1}))) \wedge (\Box(\neg(out_{p_4} \leftrightarrow out_{p_2}))) \wedge (\Box(\neg(out_{p_4} \leftrightarrow out_{p_3}))))$

Appendix C

Representative Formulae of Experiment Instances

HyperLTL formula of depth 30 with three existential quantifier and two universal quantifier.

$$\begin{aligned}\exists\pi_0\exists\pi_1\exists\pi_2\forall\pi'_0\forall\pi'_1.\neg(\Diamond((\Diamond(\Box(a_{\pi'_1})))\,\mathcal{U}\\ &(((\bigcirc(d_\pi))\,\mathcal{U}(\neg((c_{\pi_0}) \wedge (d_{\pi_0})))) \wedge (\Diamond(((c_{\pi'_0}) \leftrightarrow\\ &(\Diamond(c_{\pi_2}))) \rightarrow (\neg(\bigcirc(\bigcirc(a_{\pi_1})))))))))\end{aligned}$$

HyperLTL formula of depth 50 with six existential quantifier and no universal quantifier.

$$\begin{aligned}\exists q\exists w\exists e\exists r\exists t\exists z.\neg(\Diamond((a_r)\,\mathcal{U}(((True)\,\mathcal{U}\\ &(\neg((d_r) \wedge ((d_t) \rightarrow (\neg(e_t)))))) \wedge ((\bigcirc(\bigcirc((d_w)\,\mathcal{U}\\ &((\Box(a_z)) \rightarrow (\Diamond(\neg(\Box(b_e)))))))) \leftrightarrow\\ &((b_q) \wedge ((e_z) \leftrightarrow (\bigcirc(c_r)))))))\end{aligned}$$

HyperLTL formula of depth 30 with weak fairness, two existential quantifier and two universal quantifier.

$$\begin{aligned}\exists\pi\exists\pi'\forall\pi''.(\bigcirc(\bigcirc(\Diamond((\neg((\Diamond((True)\\ &\mathcal{U}(b_{\pi''})))\,\mathcal{U}(\neg(c_{\pi''})))) \rightarrow (\Diamond(e_{\pi''})))))) \wedge ((c_\pi)|(\Box(b_{\pi'})))\\ &\wedge (\Box(\Diamond(a_{\pi'}))) \wedge (\Box(\Diamond(b_{\pi''}))) \wedge (\Box(\Diamond(c_{\pi''}))) \wedge (\Box(\Diamond(d_{\pi''}))) \wedge (\Box(\Diamond(e_{\pi''})))\end{aligned}$$

The reduced LTL formula which is provided as an input to the LTL-SAT decision procedure, where every labelled atomic proposition x_π is interpreted as a fresh atomic proposition. Note that π'' does not occur anymore and the universal quantification is unrolled by replacing π'' with either π or π'.

$$(\bigcirc(\bigcirc(\Diamond((\neg((\Diamond((True)$$
$$\mathcal{U}(b_{\pi})))\mathcal{U}(\neg(c_{\pi})))) \rightarrow (\Diamond(e_{\pi})))))) \wedge ((c_{\pi})|(\Box(b_{\pi'})))$$
$$\wedge (\Box(\Diamond(a_{\pi'}))) \wedge (\Box(\Diamond(b_{\pi}))) \wedge (\Box(\Diamond(c_{\pi}))) \wedge (\Box(\Diamond(d_{\pi}))) \wedge (\Box(\Diamond(e_{\pi})))$$
$$\wedge$$
$$(\bigcirc(\bigcirc(\Diamond((\neg((\Diamond((True)$$
$$\mathcal{U}(b_{\pi'})))\mathcal{U}(\neg(c_{\pi'})))) \rightarrow (\Diamond(e_{\pi'})))))) \wedge ((c_{\pi})|(\Box(b_{\pi'})))$$
$$\wedge (\Box(\Diamond(a_{\pi'}))) \wedge (\Box(\Diamond(b_{\pi'}))) \wedge (\Box(\Diamond(c_{\pi'}))) \wedge (\Box(\Diamond(d_{\pi'}))) \wedge (\Box(\Diamond(e_{\pi'})))$$

Bibliography

[1] Eahyper-Tool. `http://christopherhahn.de/eahyper.html`. Accessed: 2016-02-07.

[2] Eahyper-Experiments. `http://christopherhahn.de/eahyper_experiments.tar.gz`. Accessed: 2016-02-07.

[3] Intel i5-3230M Processor. `http://ark.intel.com/products/72164/Intel-Core-i5-3230M-Processor-3M-Cache-up-to-3_20-GHz-rPGA`. Accessed: 2016-02-07.

[4] Methicillin Resistant Staphylococcus Aureus Screening. `https://labtestsonline.org/understanding/analytes/mrsa/tab/test/`. Accessed: 2016-29-01.

[5] Henrik Reif Andersen. A Polyadic Modal μ-calculus. 1994.

[6] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[7] Musard Balliu, Mads Dam, and Gurvan Le Guernic. Epistemic Temporal Logic for Information Flow Security. In *Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security*, page 6. ACM, 2011.

[8] Gilles Barthe, Pedro R D'Argenio, and Tamara Rezk. Secure Information Flow by Self-composition. In *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*, pages 100–114. IEEE, 2004.

[9] Ashok K Chandra and Larry J Stockmeyer. Alternation. In *Foundations of Computer Science, 1976., 17th Annual Symposium on*, pages 98–108. IEEE, 1976.

[10] Alonzo Church. A Set of Postulates for the Foundation of Logic. *Annals of mathematics*, pages 346–366, 1932.

[11] Alonzo Church. Application of Recursive Arithmetic to the Problem of Circuit Synthesis. 1963.

[12] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified Interference for a While Language. *Electronic Notes in Theoretical Computer Science*, 112:149–166, 2005.

[13] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic Verification of Finite-state Concurrent Systems using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[14] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT press, 1999.

[15] Michael R Clarkson and Fred B Schneider. Hyperproperties. In *Computer Security Foundations Symposium, 2008. CSF'08. IEEE 21st*, pages 51–65. IEEE, 2008.

[16] Michael R Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K Micinski, Markus N Rabe, and César Sánchez. Temporal Logics for Hyperproperties. In *Principles of Security and Trust*, pages 265–284. Springer, 2014.

[17] Rayna Dimitrova, Bernd Finkbeiner, Máté Kovács, Markus N Rabe, and Helmut Seidl. Model Checking Information Flow in Reactive Systems. In *Verification, Model Checking, and Abstract Interpretation*, pages 169–185. Springer, 2012.

[18] Alexandre Duret-Lutz. Manipulating LTL formulas using Spot 1.0. In *Automated Technology for Verification and Analysis*, pages 442–445. Springer, 2013.

[19] E Allen Emerson and Joseph Y Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time Temporal Logic. *Journal of the ACM (JACM)*, 33(1):151–178, 1986.

[20] Ronald Fagin, Yoram Moses, Moshe Y Vardi, and Joseph Y Halpern. *Reasoning About Knowledge*. MIT press, 2003.

[21] Bernd Finkbeiner, Markus N Rabe, and César Sánchez. Algorithms for Model Checking HyperLTL and HyperCTL^*. In *Computer Aided Verification*, pages 30–48. Springer, 2015.

[22] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

[23] Joseph A Goguen and José Meseguer. *Security Policies and Security Models*. IEEE, 1982.

[24] Richard M Karp. *Reducibility among Combinatorial Problems*. Springer, 1972.

[25] Dexter Kozen. Results on the Propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[26] Dexter Kozen. *Automata and Computability*. Springer Science & Business Media, 2012.

[27] Jianwen Li, Geguang Pu, Lijun Zhang, Yinbo Yao, Moshe Y Vardi, et al. Polsat: A Portfolio LTL Satisfiability Solver. *arXiv preprint arXiv:1311.1602*, 2013.

[28] John McLean. Proving Noninterference and Functional Correctness using Traces. Technical report, DTIC Document, 1992.

[29] Amir Pnueli. The Temporal Logic of Programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[30] Emil L Post. A Variant of a Recursively Unsolvable Problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

[31] Emil L Post et al. *Recursively Enummerable Sets of Postive Integers and their Decision Problems*. World Scientific, 1944.

[32] A William Roscoe. CSP and Determinism in Security Modelling. In *Security and Privacy, 1995. Proceedings., 1995 IEEE Symposium on*, pages 114–127. IEEE, 1995.

[33] Kristin Y Rozier and Moshe Y Vardi. LTL Satisfiability Checking. In *Model Checking Software*, pages 149–167. Springer, 2007.

[34] Walter J Savitch. Relationships between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[35] Sven Schewe and Bernd Finkbeiner. Bounded Synthesis. In *Automated Technology for Verification and Analysis*, pages 474–488. Springer, 2007.

[36] Viktor Schuppan and Luthfi Darmawan. Evaluating LTL Satisfiability Solvers. In *Automated Technology for Verification and Analysis*, pages 397–413. Springer, 2011.

[37] Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 277–291. Springer, 1998.

[38] A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, July 1985. ISSN 0004-5411. doi: 10.1145/3828.3837. URL `http://doi.acm.org/10.1145/3828.3837`.

[39] Geoffrey Smith. On the Foundations of Quantitative Information Flow. In *Foundations of Software Science and Computational Structures*, pages 288–302. Springer, 2009.

[40] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards Flexible Verification nder Fairness. In *Computer Aided Verification*, pages 709–714. Springer, 2009.

[41] Tachio Terauchi and Alex Aiken. *Secure Information Flow as a Safety Problem*. Springer, 2005.

[42] Alan Mathison Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[43] Guido Van Rossum and Fred L Drake. *Python Language Reference Manual*. Network Theory, 2003.

[44] Hirotoshi Yasuoka and Tachio Terauchi. On Bounding Problems of Quantitative Information Flow. In *Computer Security–ESORICS 2010*, pages 357–372. Springer, 2010.