Dissertation Summary: Logical Methods for the Hierarchy of Hyperlogics

Jana Hofmann
Azure Research, Microsoft

Abstract

The increasing prominence of digital systems in our daily lives requires rethinking the notion of correctness. Digital systems are often employed in sensitive domains; thus, a modern notion of correctness must encompass societal aspects such as data privacy and fairness concerns. Many of these properties are, in fact, hyperproperties, which relate multiple execution traces of a system. While non-relational trace properties have been extensively researched in recent decades, hyperproperties are a relatively young concept that is far from fully understood.

The presented dissertation effectively organizes the spectrum of hyperproperties through a hierarchy of hyperlogics. The resulting logical classes encompass a broad range of properties, including epistemic properties and ω-regular hyperproperties. Based on this hierarchy, we identify decidability boundaries (e.g., of the model checking problem) and propose new fragments and algorithms for the satisfiability problem. Finally, we turn to the synthesis problem and show how to circumvent its general undecidability in the example of synthesizing the temporal control flows of smart contracts from hyperproperties.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Hyperproperties, Hyperlogics, Satisfiability, Reactive Synthesis

Digital Object Identifier 10.4230/LIPIcs...0

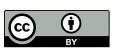
Related Version Full dissertation: `http://dx.doi.org/10.22028/D291-38889`

1 Introduction

Digital systems are playing an increasingly prominent role in our everyday lives while simultaneously becoming more and more complex. Machine-learned systems make personnel decisions, autonomous vehicles participate in public traffic, and highly sensitive health data is processed in the cloud. This development calls for a sophisticated notion of correctness: Nowadays, correctness cannot simply mean that a computer performs arithmetic calculations accurately. Instead, we must also consider aspects such as information flow security, robustness, and fairness.

Many of these properties belong to the class of *hyperproperties* [4], the class of properties that relate multiple execution traces. Hyperproperties, in comparison to trace properties, constitute a relatively recent research field that is significantly less well understood. This lack of understanding can also be attributed to the complex relational reasoning that is required for the specification of hyperproperties and the development of formal algorithms.

The dissertation presented here [16] systematically analyzes the landscape of hyperproperties through the lens of various logics (we name them *hyperlogics*). First, we investigate the expressiveness of several mechanisms for the construction of hyperlogics, resulting in a true hierarchy of logics that helps us understand the different classes of hyperproperties. We then build on this hierarchy to explore decidability boundaries and to develop algorithms for the satisfiability problem and the synthesis from hyperlogics.



1.1 Hyperproperties

The need for correctness properties beyond functional correctness expressed as trace properties became particularly evident after 2018's highly discussed Meltdown [22] and Spectre [19] attacks. These attacks revealed that the vast majority of modern CPUs is susceptible to data leaks through side channels: when running the same program on different secret data, the attacker can learn these secrets by observing, e.g., differences in the latency of cache accesses. Information flow properties such as noninterference express the absence of side channels. Noninterference states that for any pair of inputs that agree on the non-secret data, the possible observations of an attacker must be the same.

The crucial aspect of the noninterference property is the comparison of two program inputs and their corresponding executions. As a set-theoretical generalization of noninterference and similar information flow policies, Clarkson and Schneider proposed the concept of hyperproperties in 2008 [4]. Their definition is a straightforward generalization of trace properties: given an alphabet Σ, a *trace property* P is a set of traces over Σ, i.e., $P \in (2^\Sigma)^\omega$. A *hyperproperty* H, on the other hand, is a set of set of traces, i.e., $H \subseteq (2^\Sigma)^\omega$. Like that, hyperproperties define which combination of traces satisfy the property, not just which individual traces.

The general set-theoretic definition of hyperproperties was motivated by information flow properties. Since then, however, it became apparent that a variety of other properties from diverse areas of computer science also fall in the category of hyperproperties. Examples include fairness in automated decision processes, robustness of cyber-physical systems, and serializability in database queries. Fairness, for instance, can be formulated as "any two applicants with the same qualifications must have equal chances of being invited for an interview — regardless of their gender, sexual orientation, or social background".

In the past, hyperproperties have mainly been studied individually within their respective application domains, e.g., noninterference in information flow security. Logics as universal mathematical languages abstract away from the details of a specific domain and thereby enable the comparison of properties, their classification in terms of complexity, and the development of domain-independent algorithms. HyperLTL [3] was the first logic for hyperproperties and remains the most prominent one. It extends linear temporal logic (LTL) with prefixed quantifiers over the set of execution traces. Additionally, atomic propositions in the inner LTL formula are associated with one of the quantified traces. Noninterference can be expressed in HyperLTL as follows:

$$\forall\pi\forall\pi'.\, \mathit{publicIn}_\pi = \mathit{publicIn}_{\pi'} \rightarrow \square(\mathit{obs}_\pi = \mathit{obs}_{\pi'})$$

The formula expresses that for any two program executions π and π' that share the same public (non-secret) input *publicIn*, the observation *obs* of an attacker must be the same at all points in time (indicated by $\square$).

Formally, a HyperLTL formula φ is evaluated on a set of traces $T \subseteq (2^{AP})^\omega$, where AP is (as in LTL) the set of atomic propositions. The logic's syntax is defined as follows:

$$\varphi ::= \forall\pi.\, \varphi \mid \exists\pi.\, \varphi \mid \psi$$
$$\psi ::= a_\pi \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \psi\,\mathcal{U}\,\psi$$

where $a \in AP$ and $\pi \in V_\pi$, which is a set of trace variables. Above, $\bigcirc$ is the temporal "next" operator, and $\psi\,\mathcal{U}\,\psi'$ states that ψ has to hold "until" ψ' holds. Other temporal operators like $\square$ and "eventually" ($\diamondsuit$) can be derived from these operators. For the formal semantics of the logic, we refer to [16].

Similar to HyperLTL, HyperCTL* is based on CTL* and expresses branching-time hyperproperties [3]. While temporal hyperlogics are by far the most prominent hyperlogics, there also exist a handful of hyperlogics based on other types of base logics like first-order logics [13] and team semantics [20].

1.2 Formal Methods for Hyperproperties

While studying the expressiveness of logics is instrumental to develop a deep understanding of the properties they express, the ultimate goal is using these logics to facilitate the analysis and construction of provably correct systems.

The main algorithmic focus of this thesis is on the satisfiability problem and the reactive synthesis problem of hyperlogics. Both problems play a crucial role in the development of correct systems. The satisfiability problem can be leveraged to ensure a good specification quality: an unsatisfiable formula clearly constitutes a specification error, and if one formula implies another, then the latter one does not need to be included in the specification. The synthesis problem is the problem of automatically generating a system (for us, that is a Kripke structure) from a given specification. It dates back to Alonzo Church [2] is one of the most intriguing but also one of the hardest problems based on temporal logics.

With hyperlogics, the satisfiability and synthesis problem (and formal methods in general) pose a notoriously hard challenge: for HyperLTL, satisfiability checking is undecidable in general [8], and its synthesis problem is undecidable already for the fragment of universally quantified formulas [10]. Given the general undecidability of these problems, we are faced with two potential approaches towards a solution. On option is to define logical fragments (or entirely new logics) for which the problem becomes easier, but which still encompass many relevant properties. Alternatively, we can approximate the problem, preferably in a manner that maintains soundness to ensure we still obtain formal guarantees.

1.3 Contribution

This thesis is the first to thoroughly examine the expressive power of hyperlogics based on various logical mechanisms. Furthermore, it develops algorithms for the satisfiability and synthesis problems of hyperlogics. In the following, we briefly summarize the main contributions of the thesis and point to the corresponding publications. In the next sections, we discuss each of these contributions in more detail.

The Hierarchy of Hyperlogics [5, 9, 24]. In the first part of this thesis, we analyze and compare the expressiveness of hyperlogics based on different base logics. We demonstrate that quantifier-based temporal hyperlogics (like HyperLTL) and first-order/second-order hyperlogics can be strictly ordered according to their expressive power. This holds for both linear-time and branching-time hyperlogics. The resulting hierarchies enable us to draw the decidability boundaries, e.g., of the model checking problem. Even more importantly, by analyzing the classes of hyperproperties expressible in the different logics, we gain a profound understanding of how expressively complex these classes are. As a radically different approach to hyperlogics, we examine temporal logics with team semantics. This semantics enables the expression of hyperproperties without trace quantification. Not surprisingly, team-based hyperlogics do not fit into the hierarchy, but we identify fragments whose expressiveness falls into that of quantifier-based logics.

The HyperLTL Satisfiability Problem [1]. As a new approach to the highly undecidable satisfiability problem of HyperLTL, we define what we call the "temporal safety" and "temporal liveness" fragments. In contrast to the traditional definition of hypersafety,

temporal safety specifications may contain quantifier alternations. We demonstrate that the problem becomes co-semi-decidable, which is a significant reduction from the Σ^1_1-completeness of the general problem [14]. Interestingly, we demonstrate that, on the other hand, Σ^1_1-hardness already holds for very simple temporal liveness formulas. As a second contribution to the satisfiability problem, we develop a sound but necessarily incomplete algorithm for finding largest satisfying models for specifications in the $\forall\exists^*$-fragment of HyperLTL.

Smart Contract Synthesis [6, 11]. As a concrete application target, we develop logics and synthesis algorithms for hyperproperties (and trace properties) of smart contracts. Smart contracts are digital contracts implemented on top of a blockchain. The blockchain eliminates the need for a trusted third party that enforces a correct order of transactions. For all parties to trust this process, it is strictly necessary that the implicit transition system underlying the contract is correct. Unfortunately, smart contracts have been prone to errors in the past, an issue that can be addressed by developing smart contracts with the help of formal methods.

As a first step, we define logics based on temporal stream logic (TSL) [12] that are capable of expressing both trace properties and hyperproperties of smart contracts. For these logics, we develop synthesis algorithms that automatically generate correct-by-design implementations of smart contract transition systems. For trace properties, we describe an algorithm that represents the infinite-state system of a smart contract in a finite manner and, to efficiently implement the system in Solidity, divides it into a hierarchical structure of distributed systems. For hyperproperties, we propose a multi-stage approach that first tests whether the property can be expressed as a simpler trace property. If not, we design a repair mechanism that corrects an over-approximation of the system with respect to the given hyperproperty.

2 The Hierarchy of Hyperlogics

In the first part of this section, we present our results on the comparison of quantifier-based hyperlogics (i.e., temporal hyperlogics like HyperLTL versus first-order (FO) and second-order (SO) hyperlogics). In the second part, we discuss how hyperlogics based on team semantics fit into the picture.

2.1 Quantifier-based Hyperlogics

First-order Hyperlogics. Many temporal logics are expressively equivalent to a first-order or second-order logic. The most well-known of these results is Kamp's theorem [17, 15], which states that LTL is equivalent to FO[$<$]. This is first-order monadic logic of order, i.e., a FO logic with just monadic predicates except a single binary predicate $<$, which defines a strict order on the domain and thus enforces the trace-shaped model. Based on Kamp's theorem, the most likely logic for a similar result for HyperLTL would be FO[$<, E$] [13], which extends FO[$<$] with another binary predicate E. This predicate defines an "equal-level" relation between points residing on possibly different traces but on the same temporal level. The syntax of FO[$<, E$] is defined as follows.

$$\tau ::= P_a(x) \mid x < y \mid x = y \mid E(x, y)$$
$$\varphi ::= \tau \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x.\, \varphi$$

Similarly to HyperLTL, FO[$<, E$] formulas have a quantifier prefix followed by formulas from the base logic (here FO[$<$] instead of LTL). FO[$<, E$] formulas are also evaluated over a set of traces T. The difference is, however, that variables x, y range over points

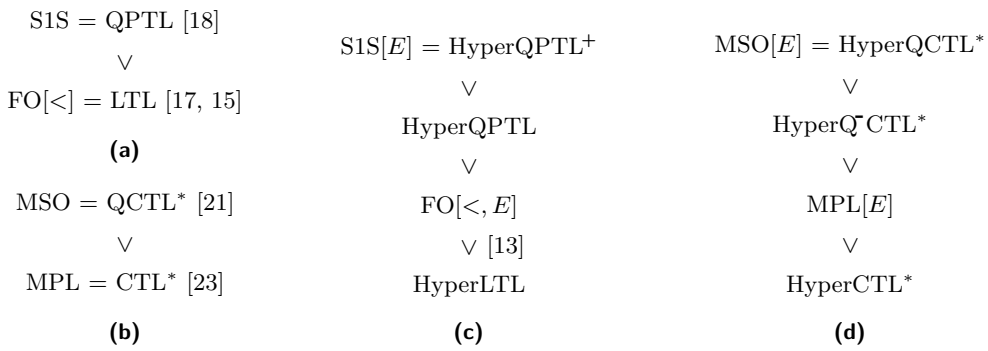


Figure 1 The linear-time hierarchies of standard logics (a) and hyperlogics (c), and the branching-time hierarchies of standard logics (b) and hyperlogics (d).

on traces, i.e., over $T \times \mathbb{N}$. The simplest example that highlights the differences between HyperLTL and FO$[<, E]$ is the formula stating that "all traces agree at all times on the value of proposition a". In HyperLTL, this would be formulated as $\forall\pi\forall\pi'.\square(a_\pi \leftrightarrow a_{\pi'})$. In FO$[<, E]$, the $\square$-operator is replaced by the explicit quantification of the variables, resulting in $\forall x\forall y.\, E(x, y) \rightarrow P_a(x) \leftrightarrow P_a(y)$.

Kamp's Theorem for Hyperlogics? Surprisingly, FO$[<, E]$ is strictly more expressive than HyperLTL [13]. Inspired by this result, we initiate a systematic study of the two mechanisms for defining hyperlogics. On the one hand, we augment temporal logics with prefixed trace quantifiers (similar to how HyperLTL is based on LTL); on the other hand, we extend their equivalent FO or SO logics with the equal-level predicate (as done for the definition of FO$[<, E]$).

Our investigation is guided by a set of known equivalences for both linear-time (depicted in Figure 1a) and branching-time logics (depicted in Figure 1b). Besides the pair LTL and FO$[<]$, we also investigate QPTL, the extension of LTL with propositional quantification. QPTL can express the entirety of ω-regular languages [18] and is equivalent to S1S (monadic second-order logic of one successor). For branching-time logics, it is known that CTL* is equivalent to monadic path logic, whereas the extension of CTL* with propositional quantification is equivalent to full monadic second-order logic.

A Hierarchy of Quantifier-based Logics. Extending these logics to hyperlogics is not always straight forward. For QPTL's propositional quantifiers $\exists q/\forall q$, for example, there are two possible semantics. One option is that the quantifiers produce a single q-sequence $s \in (2^{\{q\}})^\omega$ that is used to evaluate q over time. The other option is that they completely re-assign the proposition q in the model. For QPTL, the model is a single trace and therefore these two interpretations are equivalent; for HyperQPTL, the model is a set of traces and they yield two different logics, HyperQPTL and HyperQPTL$^+$. The same issue also applies to the definition of HyperQCTL*.

The results of our expressiveness analysis are depicted in Figure 1c for linear-time hyperlogics and in Figure 1d for branching-time hyperlogics. In both cases, the logics can be arranged in a strictly ordered hierarchy. We observe a repeating pattern: given a temporal logic and its FO/SO equivalent, the equal-level predicate E adds more expressiveness to the FO/SO logic than prefixed trace/path quantifiers add to the temporal logic. Only for the more expressive interpretations of propositional quantification, we obtain equivalent logics.

Classifying Hyperproperties through Logics. Besides the fact that this hierarchy of logics draws a clear picture of the expressiveness of different logical mechanisms, it also helps

us understand and categorize the hyperproperties they can express. Epistemic properties, for example, are a well-studied class of properties that express the knowledge of different agents in distributed systems [7]. $\text{LTL}_\mathcal{K}$ is an epistemic logic that extends LTL with the *knowledge operator*; it's expressiveness is captured by $\text{FO}[<, E]$. ω-regular hyperproperties, which lift the concept of ω-regularity to sets of traces, need the expressiveness of HyperQPTL. Another benefit of such a hierarchy are the clear decidability boundaries we obtain, for example, for the model checking problem.

▶ **Theorem 1.** *Of the logics depicted in Figure 1c, HyperQPTL is the most expressive logic that still has a decidable model checking problem.*

2.2 Hyperlogics Based on Team Semantics

As a third mechanism for the construction of linear-time hyperlogics, we investigate LTL with *team semantics* [20]. Unlike the logics presented so far, TeamLTL does not rely on some sort of trace quantification. Instead, it uses the $\vee$-operator to split the current set of traces (called a *team*) between the two subformulas. Team logics are traditionally studied in combination with various additional atomic statements and operators that gradually extend the expressiveness of a logic. As an example, the following TeamLTL uses the $\mathbin{⫽}$-operator, which, opposed to $\vee$, does not split the team but requires that either the left subformula holds on all traces or the right one does. The following formula expresses that an unknown input determines the behavior of the system, such that one half of the traces always agrees on a and the other half on b:

$$\square(a \mathbin{⫽} \neg a) \vee \square(b \mathbin{⫽} \neg b).$$

To express the same property in HyperLTL, we would need three trace quantifiers:

$$\exists \pi_1, \pi_2. \forall \pi. \square(a_{\pi_1} \leftrightarrow a_\pi) \vee \square(b_{\pi_2} \leftrightarrow b_\pi).$$

It is known that HyperLTL and TeamLTL are of incomparable expressiveness [20]. This inspires us to investigate how team-logic-specific operators like $\mathbin{⫽}$ extend the expressiveness of TeamLTL along the hierarchy presented above. We choose two representative logics for this investigation, $\text{TeamLTL}(\mathbin{⫽}, \mathsf{A}^1, \sim\!\bot)$ and $\text{TeamLTL}(\mathbin{⫽}, \mathsf{A}^1)$. The operator A^1 evaluates subformulas on all traces individually (as in the LTL semantics), whereas $\sim\!\bot$ states that a team is non-empty. What makes these logics good representatives is the fact that they are able to express all (resp., all downward-closed) Boolean relations over teams.

▶ **Theorem 2.** *$\text{TeamLTL}(\mathbin{⫽}, \mathsf{A}^1, \sim\!\bot)$ is strictly less expressive than HyperQPTL$^+$, while $\text{TeamLTL}(\mathbin{⫽}, \mathsf{A}^1)$ is strictly less expressive than HyperQPTL.*

3 The HyperLTL Satisfiability Problem

The satisfiability problem of HyperLTL, the least expressive of the logics discussed so far, is already highly undecidable, namely Σ^1_1-complete [14]. The undecidability is caused by quantifier alternations, especially $\forall$-quantifiers followed by $\exists$-quantifiers. However, the $\forall^*\exists^*$-fragment contains important properties such as generalized noninterference or program refinement. We propose two techniques to simplify the problem: fragments based on a new definition of safety and liveness and an approximating algorithm for finding largest models.

Temporal Safety and Temporal Liveness. Safety properties have a long tradition of simplifying problems like model checking or monitoring. The variant for hyperproperties,

Algorithm 1 Algorithm that searches for the largest model of a $\forall\exists^n$-property. $\mathcal{A}^\forall$ and $\mathcal{A}^{\exists i}$ existentially project $\mathcal{A}$ on the first (and $i+1$th) component and remove all trace variable annotations, resulting in automata over AP. $\mathcal{A}^\forall_{\pi_i}$ is $\mathcal{A}^\forall$ with each a in the alphabet changed to a_{π_i}.

```
1: procedure FINDMODEL(𝒜)
2:     if ℒ(𝒜^∀) = ∅ then
3:         return UNSAT;
4:     if ℒ(𝒜^{∃i}) ⊆ ℒ(𝒜^∀) for all 1 ≤ i ≤ n then
5:         return SAT, model: ℒ(𝒜^∀);
6:     𝒜_new := 𝒜 ∩ 𝒜^∀_{π_1} ∩ ... ∩ 𝒜^∀_{π_n};
7:     FINDMODEL(𝒜_new);
```

hypersafety [4], cannot serve as a simplification for the satisfiability problem: recognizing hypersafety properties is Π^1_1-complete and therefore not easier than the actual satisfiability problem. Instead, we define two new HyperLTL fragments, which we call *temporal safety* and *temporal liveness*.

▶ **Definition 3.** *A HyperLTL formula $\forall/\exists\pi_1 \dots \forall/\exists\pi_n.\, \psi$ is a* temporal safety *formula if and only if ψ describes a safety property.*

The above definition requires that the inner LTL formula describes a safety formula (in the traditional sense). Unlike hypersafety, temporal safety does not restrict the quantifier prefix. While temporal safety does not yield decidability, it significantly simplifies the problem: from Σ^1_1 to coRE.

▶ **Theorem 4.** *The satisfiability problem of the temporal safety fragment of HyperLTL is coRE-complete.*

To prove membership in coRE, we show that the problem can be reduced to the satisfiability problem of first-order logic. This constructive proof makes well-studied first-order techniques like tableau and resolution applicable to HyperLTL as well. coRE-hardness already holds for very simple formulas with only one $\square$-operator followed by a few nested $\bigcirc$-operators.

Temporal liveness can be defined analogously to temporal safety. Contrary to the temporal safety fragment, the temporal liveness fragment (surprisingly) does not simplify the satisfiability problem.

▶ **Theorem 5.** *The satisfiability problem of the temporal liveness fragment of HyperLTL is Σ^1_1-complete.*

Finding Largest Models. To complement the results above, we propose a sound but incomplete algorithm to prove satisfiability and unsatisfiability of $\forall\exists^*$-HyperLTL formulas. The insight of this algorithm is that $\forall\exists^*$-formulas are closed under union, therefore, a formula φ is satisfiable iff there is a (unique) *largest* model satisfying φ. To find the largest model of a formula, we iteratively eliminate choices for the $\exists^*$-quantifiers for which there are no witness traces when chosen as $\forall$-trace. The resulting algorithm is depicted in Algorithm 1. Let $\varphi = \forall\pi.\, \exists\pi_1, \dots \exists\pi_n.\, \psi$ and let $\mathcal{A}$ be the Büchi automaton for ψ. The automaton ranges over $AP_\pi \times AP_{\pi_1} \times \dots \times AP_{\pi_n}$, where AP_{π_i} denotes the set $\{a_{\pi_i} \mid a \in AP\}$. First, we check if the requirements of the formula on the universally quantified trace are satisfiable (if not, the formula is unsatisfiable). If this is the case, we check if the requirements on the existentially quantified traces are implied by those on the universal trace (if yes, we found a model). If this is not the case, we remove all runs of $\mathcal{A}$ with an $\exists$-component that does not satisfy the universal requirements and repeat.

▶ **Theorem 6.** *Given a HyperLTL formula* $\varphi = \forall\pi.\,\exists\pi_1,\ldots,\pi_n.\,\psi$*, if Algorithm 1 terminates with UNSAT, the formula is unsatisfiable. If it terminates with SAT and model* $\mathcal{L}(\mathcal{A}^\forall)$*, then* $\mathcal{L}(\mathcal{A}^\forall)$ *is the unique largest model of* φ*.*

4 Smart Contracts Synthesis

Smart contracts are programs that operate decentralized on a blockchain and implement contracts between multiple parties. They realize monetary transactions such as wallets, crowdfunding, auctions, and even elections. Although smart contracts are often comparably small pieces of code, they have proven to be extremely prone to errors.

The synthesis problem offers a way to increase the trust in smart contracts. Our goal is to automatically generate the underlying transition system of a contract, which describes the correct order of method calls, as well as access rights and the data flow of the contract's fields. For this purpose, we develop two suitable logics: one for trace properties and one for hyperproperties. Based on these logics, we present approximating synthesis algorithms.

Parameterized TSL. TSL [12] is a temporal logic with a cell mechanism to store values from an infinite domain, combined with uninterpreted functions and predicates. We extend TSL with universally quantified parameters to distinguish between method calls with different arguments. This enables us, for example, to express that in an election with two candidates A and B, each voter `m` is allowed to vote only once, and that a vote for candidate A should increment the corresponding counter:

$$\forall \mathtt{m}.\,\mathtt{vote}(\mathtt{m},\mathtt{A}) \rightarrow [\![\mathtt{votes}(\mathtt{A}) \leftarrowtail \mathtt{votes}(\mathtt{A}) + 1]\!] \;\wedge\; \bigcirc\Box\neg(\mathtt{vote}(\mathtt{m},\mathtt{A}) \vee \mathtt{vote}(\mathtt{m},\mathtt{B})).$$

Above, `vote` is a predicate describing a method call to the voting method, and `votes` is a field of the contract. The above formula, like all our smart contract specifications, describes a safety property. Unfortunately, the synthesis problem of safety TSL is undecidable even without quantified parameters.

▶ **Theorem 7.** *The synthesis problem of the safety fragment of TSL is undecidable.*

Synthesis from Parameterized TSL. Since the synthesis problem of already simple fragments of the logic is undecidable, we soundly approximate the problem. The challenge lies in the universal parameters, which range over an infinite domain. The solution, however, must be finitely representable in Solidity (the language for Ethereum-based smart contracts).

For a formula $\forall\mathtt{m}_1,\ldots,\mathtt{m}_n.\,\psi$, we first synthesize a finite system for ψ by using a sound reduction to the safety fragment of LTL [12]. We then define conditions under which we can divide this system into a set of smaller systems organized in a hierarchical structure. Each of these systems handles method calls with the same subset of parameters. By sharing their knowledge about the current state of the global system, these systems implement the correct infinite-state system. After translation to Solidity, the distributed representation ensures that during the execution of contract, at most one transition needs to be performed after each method call, minimizing the costs in form of gas consumption.

HyperTSL. Typical hyperproperties of a smart contract include fairness (*"No candidate is favored by the contract."*) or determinism (*"The winner of an election depends only on the received votes."*). We define HyperTSL as a logic for software-based hyperproperties. Since HyperTSL is based on TSL, it inherits its ability to describe the data flow in the fields of a contract. Fairness in an election with two candidates can be expressed in HyperTSL as a

symmetry property as follows:

$$\forall\pi\forall\pi'.\big((\llbracket \texttt{winner} \leftarrow \texttt{A} \rrbracket_\pi \leftrightarrow \llbracket \texttt{winner} \leftarrow \texttt{B} \rrbracket_{\pi'}) \ \wedge \ (\llbracket \texttt{winner} \leftarrow \texttt{B} \rrbracket_\pi \leftrightarrow \llbracket \texttt{winner} \leftarrow \texttt{A} \rrbracket_{\pi'})\big) \\ \mathcal{W}\ \big(\texttt{vote}(\texttt{A})_\pi \nleftrightarrow \texttt{vote}(\texttt{B})_{\pi'}\big).$$

The formula states that for two execution traces whose votes for A and B are swapped, the winner must also be swapped.

Synthesis from HyperTSL. Since the synthesis problem of safety TSL is already undecidable, the same holds true for HyperTSL. To approximate the problem, we take advantage of the fact that a system specification typically encompasses not only hyperproperties but also functional properties. We define two approximate mechanisms based on this idea. First, we test whether a $\forall^*$-HyperTSL specification describes a *pseudo hyperproperty*. These are properties that are equivalent to a simpler trace property:

▶ **Definition 8.** *A HyperTSL formula φ describes a pseudo hyperproperty if there exists an equivalent TSL formula ψ.*

We prove that if φ describes a pseudo-hyperproperty, computing the corresponding TSL formula ψ is straight-forward. Though we show that the equivalence check is undecidable, we can soundly reduce the problem to the corresponding check for HyperLTL, which is decidable.

For genuine $\forall^*$-HyperTSL properties, we propose a repair mechanism: we synthesize the winning region of the trace properties described in TSL (these must be safety properties). This system may include nondeterminism, which we resolve to satisfy the HyperTSL property.

5 Conclusion

The presented dissertation is the first work that systematically maps out the landscape of hyperproperties by comparing the expressiveness of hyperlogics based on various base logics. The resulting hierarchy of hyperlogics reveals decidability boundaries (e.g., for the model checking problem) and enables comparing and categorizing the complexity of different hyperproperties. Spanning a range of logical mechanisms, the hierarchy also lays the foundation for future expressiveness analyses, for example, of asynchronous, probabilistic, or fixpoint hyperproperties.

Algorithmically, this work demonstrates how the general undecidability of many hyperproperty problems may be overcome by restrictions to suitable fragments and sound approximations. Two approaches are particularly interesting. First, as for trace properties, a restriction to safety properties also simplifies problems in the realm of hyperproperties (given an appropriate definition of safety). Second, we show that treating hyperproperties together with non-relational trace properties narrows down the search space for potential solutions.

References

1 Raven Beutner, David Carral, Bernd Finkbeiner, Jana Hofmann, and Markus Krötzsch. Deciding hyperproperties combined with functional specifications. In *37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2022.

2 Alonzo Church. Applications of Recursive Arithmetic to the Problem of Circuit Synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, 1957.

3 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Third International Conference on Principles of Security and Trust (POST)*, 2014.

4 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *21st IEEE Computer Security Foundations Symposium (CSF)*, 2008.

5 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2019.

6 Norine Coenen, Bernd Finkbeiner, Jana Hofmann, and Julia Tillman. Smart contract synthesis modulo hyperproperties, 2022. To appear at the 36th IEEE Computer Security Foundations Symposium (CSF). URL: `https://arxiv.org/abs/2208.07180`.

7 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

8 Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *27th International Conference on Concurrency Theory (CONCUR)*, 2016.

9 Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing omega-regular hyperproperties. In *32nd International Conference on Computer Aided Verification (CAV)*. Springer, 2020.

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

11 Bernd Finkbeiner, Jana Hofmann, Florian Kohn, and Noemi Passing. Reactive synthesis of smart contract control flows, 2023. To appear at the 21st International Symposium on Automated Technology for Verification and Analysis (ATVA). URL: `https://arxiv.org/abs/2205.06039`.

12 Bernd Finkbeiner, Felix Klein, Ruzica Piskac, and Mark Santolucito. Temporal stream logic: Synthesis beyond the bools. In *31st International Conference on Computer-Aided Verification (CAV)*. Springer, 2019.

13 Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. In *34th Symposium on Theoretical Aspects of Computer Science, (STACS)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

14 Marie Fortin, Louwe B. Kuijer, Patrick Totzke, and Martin Zimmermann. Hyperltl satisfiability is Σ^1_1-complete, hyperctl* satisfiability is Σ^2_1-complete. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

15 Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 1980.

16 Jana Hofmann. *Logical Methods for the Hierarchy of Hyperlogics*. PhD thesis, Saarland University, Germany, 2022.

17 Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Computer Science Department, University of California at Los Angeles, USA, 1968.

18 Yonit Kesten and Amir Pnueli. A complete proof systems for QPTL. In *10th Annual IEEE Symposium on Logic in Computer Science (LICS)*, 1995.

19 Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (S&P)*, 2019.

20 Andreas Krebs, Arne Meier, Jonni Virtema, and Martin Zimmermann. Team semantics for the specification and verification of hyperproperties. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

21 François Laroussinie and Nicolas Markey. Quantified CTL: expressiveness and complexity. *Logical Methods in Computer Science*, 10(4), 2014.

22 Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg.

Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium*, 2018.

23 Faron Moller and Alexander Moshe Rabinovich. On the expressive power of CTL. In *14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, 1999.

24 Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. Linear-time temporal logic with team semantics: Expressivity and complexity. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2021.