Deciding Hyperproperties*

Bernd Finkbeiner[1] and Christopher Hahn[2]

1 **Saarland University**
Saarbrücken, Germany
finkbeiner@react.uni-saarland.de
2 **Saarland University**
Saarbrücken, Germany
hahn@react.uni-saarland.de

Abstract

Hyperproperties, like observational determinism or symmetry, cannot be expressed as properties of individual computation traces, because they describe a relation between multiple computation traces. HyperLTL is a temporal logic that captures such relations through trace variables, which are introduced through existential and universal trace quantifiers and can be used to refer to multiple computations at the same time. In this paper, we study the satisfiability problem of HyperLTL. We show that the problem is PSPACE-complete for alternation-free formulas (and, hence, no more expensive than LTL satisfiability), EXPSPACE-complete for $\exists^*\forall^*$ formulas, and undecidable for $\forall\exists$ formulas. Many practical hyperproperties can be expressed as alternation-free formulas. Our results show that both satisfiability and implication are decidable for such properties.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases temporal logics, satisfiability, hyperproperties, complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.13

1 Introduction

Hyperproperties [4] are system properties that relate multiple computation traces. For example, in the design of a system that handles sensitive information, we might specify that a certain secret is kept confidential by requiring that the system is *deterministic* in its legitimately observable inputs, i.e., that all computations with the same observable inputs must have the same observable outputs, independently of the secret [13, 16]. In the design of an access protocol for a shared resource, we might specify that the access to the resource is *symmetric* between multiple clients by requiring that for every computation and every permutation of the clients, there exists a computation where the access is granted in the permuted order [7].

To express hyperproperties in a temporal logic, linear-time temporal logic (LTL) has recently been extended with trace variables and trace quantifiers. In HyperLTL [3], observational determinism can, for example, be expressed as the formula $\forall\pi.\forall\pi'.\ \square(I_\pi = I_{\pi'}) \rightarrow \square(O_\pi = O_{\pi'})$, where I is the set of observable inputs and O is the set of observable outputs. The universal quantification of the trace variables π and π' indicates that the property must hold for all pairs of computation traces. It has been shown that many hyperproperties of interest can be expressed in HyperLTL [12].

* This work was partially supported by the German Research Foundation (DFG) in the Collaborative Research Center 1223 and by the Graduate School of Computer Science at Saarland University.

In this paper, we study the *satisfiability problem* of HyperLTL. Unlike the model checking problem, for which algorithms and tools exist [3, 7], the decidability and complexity of the satisfiability problem was, so far, left open. The practical demand for a decision procedure is strong. Often, one considers multiple formalizations of similar, but not necessarily equivalent hyperproperties. An alternative (and slightly stronger) version of observational determinism requires, for example, that differences in the observable output may only occur *after* differences in the observable input have occurred: $\forall\pi.\forall\pi'.\ (O_\pi = O_{\pi'})\ \mathcal{W}\ (I_\pi \neq I_{\pi'})$. A decision procedure for HyperLTL would allow us to automatically check whether such formalizations imply each other. Another important application is to check whether the functionality of a system, i.e., a standard trace property, is compatible with the desired hyperproperties, such as confidentiality. Since both types of properties can be expressed in HyperLTL, a decision procedure for HyperLTL would make it possible to identify inconsistent system requirements early on, before an attempt is made to implement the requirements.

The fundamental challenge in deciding hyperproperties is that hyperproperties are usually not ω-regular [1]. HyperLTL formulas thus cannot be translated into equivalent automata [6]. Intuitively, since hyperproperties relate multiple infinite traces, an automaton, which only considers one trace at a time, would have to memorize an infinite amount of information from one trace to the next. This means that the standard recipe for checking the satisfiability of a temporal logic, which is to translate the given formula into an equivalent Büchi automaton and then check if the language of the automaton is empty [15], cannot be applied to HyperLTL.

In model checking, this problem is sidestepped by verifying the *self-composition* [2] of the given system: instead of verifying a hyperproperty that refers to n traces, we verify a trace property that refers to a *single* trace of a new system that contains n copies of the original system. Since the satisfiability problem does not refer to a system, this idea cannot immediately be applied to obtain a decision procedure for HyperLTL. However, it would seem natural to define a similar self-composition, on the formula rather than the system, in order to determine satisfiability.

We organize our investigation according to the quantifier structure of the HyperLTL formulas. LTL, for which the satisfiablity problem is already solved [14], is the sublogic of HyperLTL where the formulas have a single universally quantified trace variable, which is usually left implicit. The next larger fragment consists of the alternation-free formulas, i.e., formulas with an arbitrary number of trace variables and a quantifier prefix that either consists of only universal or only existential quantifiers. Many hyperproperties of practical interest, such as observational determinism, belong to this fragment. It turns out that the satisfiability of alternation-free formulas can indeed be reduced to the satisfiability of LTL formulas by replicating the atomic propositions such that there is a separate copy for each trace variable. This construction is sound, because in an alternation-free formula, the values for the quantifiers can be chosen independently of each other. The size of the resulting LTL formula is the same as the given HyperLTL formula; as a result, the satisfiability problem of the alternation-free fragment has the same complexity, PSPACE-complete, as LTL satisfiability.

If the formula contains a quantifier alternation, the values of the quantifiers can no longer be chosen independently of each other. However, if the quantifier structure is of the form $\exists^*\forall^*$, i.e., the formula begins with an existential quantifier and then has a single quantifier alternation, then it is still possible to reduce HyperLTL satisfiability to LTL satisfiability by explicitly considering all possible interactions between the existential and universal quantifiers. For example, $\exists\pi_0\exists\pi_1\forall\pi_2.\ (\bigcirc p_{\pi_0}) \wedge (\square p_{\pi_1}) \wedge (\diamondsuit p_{\pi_2})$ is equisatisfiable to $\exists\pi_0\exists\pi_1.\ (\bigcirc p_{\pi_0}) \wedge (\square p_{\pi_1}) \wedge (\diamondsuit p_{\pi_0}) \wedge (\diamondsuit p_{\pi_1})$, which is in turn equisatisfiable to the

Table 1 Complexity results for the satisfiability problem of HyperLTL.

$\exists^*$	$\forall^*$	$\exists^*\forall^*$	bounded $\exists^*\forall^*$	$\forall\exists$
PSPACE-complete	PSPACE-complete	EXPSPACE-complete	PSPACE-complete	undecidable

LTL formula $(\bigcirc p_0) \wedge (\Box p_1) \wedge (\Diamond p_0) \wedge (\Diamond p_1)$. In general, enumerating all combinations of existential and universal quantifiers causes an exponential blow-up and we show that the satisfiability problem for the $\exists^*\forall^*$-fragment is indeed EXPSPACE-complete. This high complexity is, however, relativized by the fact that practical hyperproperties rarely need a large number of quantifiers. If we bound the number of universal quantifiers by a constant, the complexity becomes PSPACE again.

Formulas where an existential quantifier occurs in the scope of a universal quantifier make the logic dramatically more powerful, because they can be used to enforce, inductively, models with an infinite number of traces. We show that a single pair of quantifers of the form $\forall\exists$ suffices to encode Post's correspondence problem. The complete picture is thus as summarized in Table 1: The largest decidable fragment of HyperLTL is the EXPSPACE-complete $\exists^*\forall^*$ fragment. Bounding the number of universal quantifiers and in particular restricting to alternation-free formulas reduces the complexity to PSPACE. Any fragment that contains the $\forall\exists$ formulas is undecidable.

From a theoretical point of view, the undecidability of the $\forall\exists$ fragment is a noteworthy result, because it confirms the intuition that hyperproperties are truly more powerful than trace properties. In practice, already the alternation-free fragment suffices for many important applications (cf. [7]). From a practical point of view, the key result of the paper is therefore that both satisfiability of alternation-free formulas and implication between alternation-free formulas, which can be expressed as unsatisfiability of an $\exists^*\forall^*$ formula, are decidable.

2 HyperLTL

Let AP be a set of *atomic propositions*. A *trace* t is an infinite sequence over subsets of the atomic propositions. We define the set of traces $TR := (2^{AP})^\omega$. A subset $T \subseteq TR$ is called a *trace property*. We use the following notation to manipulate traces: let $t \in TR$ be a trace and $i \in \mathbb{N}$ be 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 $t[i]\ t[i+1] \dots t[j-1]\ t[j]$. $t[i,\infty]$ denotes the infinite suffix of t starting at position i.

LTL Syntax. Linear-time temporal logic (LTL) [9] combines the usual boolean connectives with temporal modalities such as the *Next* operator $\bigcirc$ and the *Until* operator $\mathcal{U}$. The syntax of LTL is given by the following grammar:

$$\varphi \ ::= \ p \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \bigcirc\varphi \ \mid \ \varphi \,\mathcal{U}\, \varphi$$

where $p \in AP$ is an atomic proposition. $\bigcirc\varphi$ means that φ holds in the *next* position of a trace; $\varphi_1 \,\mathcal{U}\, \varphi_2$ means that φ_1 holds *until* φ_2 holds. There are several derived operators, such as $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$, and $\varphi_1\ W\varphi_2 \equiv (\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \Box\varphi_1$. $\Diamond\varphi$ states that φ will *eventually* hold in the future and $\Box\varphi$ states that φ holds *globally*; $\mathcal{W}$ is the *weak* version of the *until* operator.

LTL Semantics. Let $p \in AP$ and $t \in TR$. The semantics of an LTL formula is defined as the smallest relation $\models$ that satisfies the following conditions:

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

LTL-SAT is the problem of deciding whether there exists a trace $t \in TR$ such that $t \models \psi$.

▶ **Theorem 1.** *LTL-SAT is PSPACE-complete [14].*

HyperLTL Syntax. HyperLTL [3] extends LTL with trace variables and trace quantifiers. Let $\mathcal{V}$ be an infinite supply of trace variables. The syntax of HyperLTL is given by the following grammar:

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

where $a \in AP$ is an atomic proposition and $\pi \in \mathcal{V}$ is a trace variable. Note that atomic propositions are indexed by trace variables. The quantification over traces makes it possible to express properties like "on all traces ψ must hold", which is expressed by $\forall \pi.\ \psi$. Dually, one can express that "there exists a trace such that ψ holds", which is denoted by $\exists \pi.\ \psi$. The derived operators $\Diamond$, $\Box$, and $\mathcal{W}$ are defined as for LTL.

HyperLTL Semantics. A HyperLTL formula defines a *hyperproperty*, i.e., a set of sets of traces. A set T of traces satisfies the hyperproperty if it is an element of this set of sets. Formally, the semantics of HyperLTL formulas is given with respect to a *trace assignment* Π from $\mathcal{V}$ to TR, i.e., a partial function mapping trace variables to actual traces. $\Pi[\pi \mapsto t]$ denotes that π is mapped to t, with everything else mapped according to Π. $\Pi[i,\infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i,\infty]$ for all π.

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

HyperLTL-SAT is the problem of deciding whether there exists a *non-empty* set of traces T such that $\Pi \models_T \psi$, where Π is the empty trace assignment and $\models_T$ is the smallest relation satisfying the conditions above. If it is clear from the context, we omit Π and simply write $T \models \psi$. If $\models_T \psi$, we call T a model of ψ.

3 Alternation-free HyperLTL

We begin with the satisfiability problem for the alternation-free fragments of HyperLTL. We call a HyperLTL formula ψ (quantifier) *alternation-free* iff the quantifier prefix only consists of either only universal or only existential quantifiers. We denote the corresponding fragments as the $\forall^*$ and $\exists^*$ fragments, respectively. For both fragments, we show that every formula can be reduced, as discussed in the introduction, to an equisatisfiable LTL formula of the same size. As a result, we obtain that the satisfiability problem of alternation-free HyperLTL is PSPACE-complete, like the satisfiability problem of LTL. In the following, some proofs are omitted due to space constraints. The proofs can be found in the full version of this paper [5].

3.1 The $\forall^*$ Fragment

The $\forall^*$ fragment is particularly easy to decide, because we can restrict the models, without loss of generality, to singleton sets of traces: since all quantifiers are universal, every model with more than one trace could immediately be translated into another one where every trace except one is omitted. Hence, we can ignore the trace variables and interpret the HyperLTL formula as a plain LTL formula.

▶ **Example 2.** Consider the following HyperLTL formula with atomic propositions $\{a, b\}$:

$$\forall \pi_1 \forall \pi_2.\ \square b_{\pi_1} \wedge \square \neg b_{\pi_2}$$

Since the trace variables are universally quantified, we are reasoning about *every* pair of traces, and thus in particular about the pairs where both variables refer to the same trace. It is, therefore, sufficient to check the satisfiability of the LTL formula $\square b \wedge \square \neg b$, which turns out to be unsatisfiable.

The satisfiability of hyperproperties that can be expressed in the $\forall^*$ fragment, such as observational determinism, thus immediately reduces to LTL satisfiability.

▶ **Lemma 3.** *For every $\forall^*$ HyperLTL formula there exists an equisatisfiable LTL formula of the same size.*

3.2 The $\exists^*$ Fragment

A model of a formula in the $\exists^*$ fragment may, in general, have more than one trace. For example the models of $\exists \pi_1 \exists \pi_2.\ a_{\pi_1} \wedge \neg a_{\pi_2}$ have (at least) two traces. In order to reduce HyperLTL satisfiability again to LTL satisfiability, we *zip* such traces together. For this purpose, we introduce a fresh atomic proposition for every atomic proposition a and every path variable π that occur as an indexed proposition a_π in the formula. We obtain an equisatisfiable LTL formula by removing the quantifier prefix and replacing every occurrence of a_π with the new proposition.

▶ **Example 4.** Consider the following HyperLTL formula over the atomic propositions $\{a, b\}$:

$$\exists \pi_1 \exists \pi_2.\ a_{\pi_1} \wedge \square \neg b_{\pi_1} \wedge \square b_{\pi_2}$$

By discarding the quantifier prefix and replacing the indexed propositions with fresh propositions, we obtain the equisatisfiable LTL formula over the atomic propositions $\{a_1, b_1, b_2\}$:

$$a_1 \wedge \square \neg b_1 \wedge \square b_2$$

The LTL formula is satisfied by the trace $\tilde{p}$: $(\{a_1, b_2\})^\omega$. We can map the fresh propositions back to the original indexed propositions. In this way, we obtain witnesses for π_1 and π_2 by splitting $\tilde{p}$ into two traces $\{a\}^\omega$ and $\{b\}^\omega$, where for every position in these traces only those atomic propositions that were labelled with π_1 or π_2, respectively, hold. Hence, the trace set satisfying the HyperLTL formula is $\{\{a\}^\omega, \{b\}^\omega\}$.

▶ **Lemma 5.** *For every $\exists^*$ HyperLTL formula there exists an equisatisfiable LTL formula of the same size.*

Combining Lemma 3 and Lemma 5, we conclude that HyperLTL-SAT inherits the complexity of LTL-SAT for the alternation-free fragment.

▶ **Theorem 6.** *HyperLTL-SAT is PSPACE-complete for the alternation-free fragment.*

4 The $\exists^*\forall^*$ Fragment

Allowing quantifier alternation makes the satisfiability problem significantly more difficult, and even leads to undecidability, as we will see in the next section. In this section, we show that deciding formulas with a single quantifier alternation is still possible if the quantifiers start with an existential quantifier. A HyperLTL formula is in the $\exists^*\forall^*$ fragment iff it is of the form $\exists \pi_1 \dots \exists \pi_n \forall \pi'_1 \dots \forall \pi'_m.\ \psi$. This fragment is especially interesting, because it includes implications between alternation-free formulas. The idea of the decision procedure is to eliminate the universal quantifiers by explicitly enumerating all possible interactions between the universal and existential quantifiers. This leads to an exponentially larger, but equisatisfiable $\exists^*$ formula.

▶ **Lemma 7.** *For every formula in the $\exists^*\forall^*$ fragment, there is an equisatisfiable formula in the $\exists^*$ fragment with exponential size.*

Proof. We define a function *sp* that takes a formula of the form $\exists \pi_1 \dots \exists \pi_n \forall \pi'_1 \dots \forall \pi'_m.\ \psi$ and yields an $\exists^*$ HyperLTL formula ψ' of size $\mathcal{O}(n^m)$ 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:

$$\exists \pi_1 \dots \exists \pi_n.\ \bigwedge_{j_1=1}^{n} \dots \bigwedge_{j_m=1}^{n}.\ \psi[\pi'_1 \backslash \pi_{j_1}] \dots \psi[\pi'_m \backslash \pi_{j_m}]$$

Let φ be an $\exists^*\forall^*$ HyperLTL formula satisfied by some model T. Hence, there exist traces $t_1, \dots, t_n \in T$ such that $\{t_1, \dots, t_n\}$ satisfies $sp(\varphi)$. Assume $sp(\varphi)$ is satisfied by some model T'. Since *sp* covers every possible combination of trace assignments for the universally quantified trace variables, $T' \models \varphi$. ◀

▶ **Example 8.** Consider the $\exists^*\forall^*$ formula $\exists \pi_1 \exists \pi_2 \forall \pi'_1 \forall \pi'_2.\ (\square a_{\pi'_1} \wedge \square b_{\pi'_2}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})$. Applying the construction from Lemma 7, we obtain the following $\exists^*$ formula:

$$\begin{aligned} &sp(\exists \pi_1 \exists \pi_2 \forall \pi'_1 \forall \pi'_2.\ (\square a_{\pi'_1} \wedge \square b_{\pi'_2}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \text{ yields :} \\ &\quad \exists \pi_1 \exists \pi_2.\ ((\square a_{\pi_1} \wedge \square b_{\pi_1}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \\ &\qquad \wedge ((\square a_{\pi_2} \wedge \square b_{\pi_1}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \\ &\qquad \wedge ((\square a_{\pi_1} \wedge \square b_{\pi_2}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \\ &\qquad \wedge ((\square a_{\pi_2} \wedge \square b_{\pi_2}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \end{aligned}$$

Combining the construction from Lemma 7 with the satisfiability check for $\exists^*$ formulas from Section 3, we obtain an exponential-space decision procedure for the $\exists^*\forall^*$ fragment.

▶ **Theorem 9.** $\exists^*\forall^*$ *HyperLTL-SAT is EXPSPACE-complete.*

Proof. Membership in EXPSPACE follows from Lemma 7 and Lemma 5. We show EXPSPACE-hardness via a reduction from the problem whether an exponential-space bounded deterministic Turing machine T accepts an input word x. Given T and x, we construct an $\exists^*\forall^*$ HyperLTL formula φ such that T accepts x iff φ is satisfiable.

Let $T = (\Sigma, Q, q_0, F, \rightarrow)$, where Σ is the alphabet, Q is the set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\rightarrow \subseteq Q \times \Sigma \times Q \times \Sigma \times \{L, R\}$ is the transition relation. We use $(q, \sigma) \rightarrow (q', \sigma', \Delta)$ to indicate that when T is in state q and it reads the input σ in the current tape cell, it changes its state to q', writes σ' in the current tape cell, and moves its head one cell to the left if $\Delta = L$ and one cell to the right if $\Delta = R$. Let $n \in \mathcal{O}(|x|)$ be such that the working tape of T has 2^n cells. We encode each letter of Σ as a valuation of a set $\vec{s} = \{s_1, \ldots, s_{k_\Sigma}\}$ of atomic propositions and each state Q as a valuation of another set $\vec{q} = \{q_1, \ldots, q_{k_Q}\}$ of atomic propositions, where k_Σ is logarithmic in $|\Sigma|$ and k_Q is logarithmic in $|Q|$. We furthermore use the valuations of a set $\vec{a} = \{a_1, \ldots, a_n\}$ to encode the position of a tape cell in a configuration of T, and the valuations of a set $\vec{h} = \{h_1, \ldots, h_n\}$ to encode the position of the head of the Turing machine. With these atomic propositions, we can represent configurations of the Turing machine as sequences of valuations of the atomic propositions. The state of the Turing machine is encoded as the valuation of $\vec{q}$ at the position indicated by $\vec{h}$. Computations of a Turing machine are sequences of configurations; we thus represent computations as traces.

We begin our encoding into HyperLTL with four quantifier-free formulas over a free trace variable π: $\varphi_{init}(\pi)$ encodes that the initial configuration represents x and q_0, and places the head in the first position of the sequence. $\varphi_{head}(\pi)$ ensures that the position of the head may only change when a new configuration begins and that the change of the position as well as the change of the state is as defined by $\rightarrow$. $\varphi_{count}(\pi)$ expresses that the addresses in $\vec{a}$ continuously count from 1 to 2^n. $\varphi_{halt}(\pi)$ expresses that the Turing machine halts eventually, i.e., the trace eventually visits a final state at the position of the head.

The more difficult part of the encoding now concerns the comparison of the tape content from one configuration to the next. We need to enforce that the tape content at the position represented by $\vec{h}$ changes as defined by $\rightarrow$, and that the content of all tape cells except for the position represented by $\vec{h}$ stays the same. For this purpose, we need to be able to memorize a position from one configuration to the next. We accomplish the "memorization" with the following trick: we introduce two existentially quantified trace variables π_{zero} and π_{one}. Let v be a new atomic proposition. We use a quantifier-free formula $\varphi_{zero/one}(\pi_{zero}, \pi_{one})$ to ensure that v is always *false* on π_{zero} and always *true* on π_{one}. We now introduce another set of n universally quantified trace variables $\pi_1, \pi_2, \ldots, \pi_n$ that will serve as memory: if one of these trace variables is bound to π_{zero} its "memory content" is 0, if it is bound to π_{one} its memory content is 1. We add a sufficient number of universally quantified variables to memorize the position of some cell and its content. Our complete encoding of the Turing machine as a HyperLTL formula then looks, so far, as follows:

$$\begin{aligned}&\exists \pi, \pi_{zero}, \pi_{one}.\ \forall \pi_1, \pi_2, \ldots, \pi_n, \pi'_1, \pi'_2, \ldots, \pi'_{k_\Sigma}.\\ &\quad \varphi_{init}(\pi) \wedge \varphi_{head}(\pi) \wedge \varphi_{count}(\pi) \wedge \varphi_{halt}(\pi) \wedge\ \varphi_{zero/one}(\pi_{zero}, \pi_{one})\\ &\quad \wedge \psi(\pi, \pi_1, \pi_2, \ldots, \pi_n, \pi'_1, \pi'_2, \ldots, \pi'_{k_\Sigma})\end{aligned}$$

The missing requirement about the correct contents of the tape cells is encoded in the last conjunct ψ. We first ensure that all the universally quantified traces have constant values in v, i.e., v is either always *true* or always *false*. To enforce that the tape content changes at the head position, we specify in ψ that whenever we are at the head position, i.e., whenever

$a_{i,\pi} = h_{i,\pi}$ for all $i = 1, \dots, n$, then when we visit the same position in the next configuration, the tape content must be as specified by $\rightarrow$: i.e., if $a_{i,\pi} = v_{\pi_i}$ for all $i = 1, \dots, n$, then when $a_{i,\pi} = v_{\pi_i}$ holds again for all $i = 1, \dots, n$ during the next configuration, the tape content as represented in $\vec{s}$ must be the one defined by $\rightarrow$. To enforce that the tape content is the same at every position except that encoded in $\vec{h}$, we specify that for all positions except the head position, i.e., whenever $a_{i,\pi} \neq h_{i,\pi}$ for some $i = 1, \dots, n$, then if $a_{i,\pi} = v_{\pi_i}$ for all $i = 1, \dots, n$, and $s_{i,\pi} = v_{\pi'_i}$ for all $i = 1, \dots, k_\Sigma$, then the following must hold: when, during the next configuration, we visit the same position again, i.e., when again $a_{i,\pi} = v_{\pi_i}$ for all $i = 1, \dots, n$, we must also find the same tape content again, i.e., $s_{i,\pi} = v_{\pi'_i}$ for all $i = 1, \dots, k_\Sigma$.

By induction on the length of the computation prefix, we obtain that any model of the HyperLTL formula represents in π a correct computation of the Turing machine T. Since this computation must reach a final state, the model exists iff T accepts the input word x. ◀

In practice, the number of quantifiers is usually small. Often it is sufficient to reason about pairs of traces, which can be done with just two quantifiers. To reflect this observation, we define a *bounded* version of the $\exists^*\forall^*$ fragment, where the number of universal quantifiers that may occur in the HyperLTL formula is bounded by some constant $b \in \mathbb{N}$. A bounded $\exists^*\forall^*$ formula of length n with bound b can be translated to an equisatisfiable LTL formulas of size $\mathcal{O}(n^b)$. The satisfiablility problem can thus be solved in polynomial space.

▶ **Corollary 10.** *Bounded $\exists^*\forall^*$ HyperLTL-SAT is PSPACE-complete.*

Another observation that is important for the practical application of our results is that implication between alternation-free formulas is decidable. As discussed in the introduction, it frequently occurs that multiple formalizations are proposed for the same hyperproperty, and one would like to determine whether the proposals are equivalent, or whether one version is stronger than the other. A HyperLTL formula ψ *implies* a HyperLTL formula φ iff every set T of traces that satisfies ψ also satisfies φ.

To determine whether ψ implies φ, we check the satisfiability of the negation $\neg(\psi \rightarrow \varphi)$. If one formula is in the $\forall^*$ fragment and the other in the $\exists^*$ fragment, implication checking is especially easy, because the formula we obtain is alternation-free.

▶ **Example 11.** To determine if the $\forall^*$ formula $\forall\pi_1 \dots \forall\pi_n.\ \psi$ implies the $\forall^*$ formula $\forall\pi'_1 \dots \forall\pi'_m.\ \varphi$, we check the $\exists^*\forall^*$ formula $\exists\pi_1 \dots \pi_n \forall\pi'_1 \dots \pi'_m.\ \psi \wedge \neg\varphi$ for unsatisfiability.

Analogously to Theorem 9, we obtain that checking implication between two alternation-free HyperLTL formulas is EXPSPACE-complete.

▶ **Theorem 12.** *Checking implication between alternation-free HyperLTL formulas is EXPSPACE-complete.*

Proof. The upper bound of Theorem 9 applies here as well. For the lower bound, we note that the encoding in the proof of Theorem 9 is of the form

$$\exists\pi, \pi_{zero}, \pi_{one}.\ \forall\vec{\pi}'.\ \varphi_1(\pi) \wedge \varphi_2(\pi_{zero}, \pi_{one}) \wedge \psi(\pi, \vec{\pi}'),$$

which is not an implication of alternation-free formulas. We can, however, transform this formula into an equisatisfiable formula by quantifying π universally:

$$\exists\pi_{zero}, \pi_{one}.\ \forall\pi, \vec{\pi}'.\ \varphi_1(\pi) \wedge \varphi_2(\pi_{zero}, \pi_{one}) \wedge \psi(\pi, \vec{\pi}')$$

In the models of the new formula, the accepting computation of the Turing machine is simply represented on *all* traces instead of on *some* trace. The formula is satisfiable iff the following

$$\forall\pi\exists\pi_s\exists\pi'.\ \Big(\big((\dot{a},\dot{a})_{\pi_s} \vee (\dot{b},\dot{b})_{\pi_s}\big) \tag{1}$$

$$\wedge\ ((\tilde{a},\tilde{a})_{\pi_s} \vee (\tilde{b},\tilde{b})_{\pi_s})\,\mathcal{U}\,\square(\#,\#)_{\pi_s}\Big) \tag{2}$$

$$\wedge\ \diamondsuit\square(\#,\#)_{\pi} \tag{3}$$

$$\wedge\ \Big(\bigvee_{i\in\{1,2,3\}} \mathit{StoneEncoding}_i \tag{4}$$

$$\vee\ \square(\#,\#)_{\pi}\Big) \tag{5}$$

Figure 1 Reduction to HyperLTL for the PCP instance from Example 13.

implication between $\exists^*$ formulas does *not* hold:

$$\exists\pi_{zero},\pi_{one}.\ \varphi_2(\pi_{zero},\pi_{one}) \quad \text{implies} \quad \exists\pi,\vec{\pi}'.\ \neg(\varphi_1(\pi)\wedge\psi(\pi,\vec{\pi}'))$$

Hence, we have reduced the problem whether an exponential-space bounded deterministic Turing machine accepts a certain input word to the implication problem between two $\exists^*$ HyperLTL formulas. ◀

With the results of this section, we have reached the borderline of the decidable HyperLTL fragments. We will see in the next section that HyperLTL-SAT immediately becomes undecidable if the formulas contain a quantifier alternation that starts with a universal quantifier.

5 The Full Logic

We now show that any extension beyond the already considered fragments makes the satisfiability problem undecidable. We prove this with a many-one-reduction from *Post's correspondence problem* (PCP) [11] to the satisfiability of a $\forall\exists$ HyperLTL formula. In PCP, we are given two lists α and β consisting of finite words from some alphabet Σ. For example, α, with $\alpha_1 = a$, $\alpha_2 = ab$ and $\alpha_3 = bba$ and β, with $\beta_1 = baa$, $\beta_2 = aa$ and $\beta_3 = bb$, where α_i denotes the ith element of the list, and α_{i_j} denotes the jth symbol of the ith element. In this example, α_{3_1} corresponds to b. PCP is the problem 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 $\alpha_{i_1}\dots\alpha_{i_K} = \beta_{i_1}\dots\beta_{i_K}$. We denote the finite words of a PCP solution with i_α and i_β, respectively.

It is a useful intuition to think of the PCP instance as a set of n domino stones. The first stone of our example is $\frac{a}{baa}$, the second is $\frac{ab}{aa}$ and the third, and last, is $\frac{bba}{bb}$. Those stones must be arranged (where copying is allowed) to construct the same word with the α- and β-concatenations. A possible solution for this PCP instance would be $(3,2,3,1)$, since the stone sequence $\frac{bba}{bb}\ \frac{ab}{aa}\ \frac{bba}{bb}\ \frac{a}{baa}$ produces the same word, i.e., $bbaabbbaa = i_\alpha = i_\beta$. For modelling the necessary correspondence between the α and β components, we will use pairs of the PCP instance alphabet as atomic propositions, e.g., (a,b). We represent a stone as a sequence of such pairs, where the first position of the pair contains a symbol of the α component and the second position a symbol of the β component. For example, the first stone $\frac{a}{baa}$ will be represented as $(a,b),(\#,b)(\#,a)$. We will use $\#$ as a termination symbol. Since

$$\mathit{StoneEncoding}_3 = \tag{6}$$

$$\Big(\big((\dot{b},\dot{b})_\pi \wedge \bigcirc(b,b)_\pi \wedge \bigcirc\bigcirc(a,\dot{*})_\pi \wedge \bigcirc\bigcirc\bigcirc(\dot{*},\tilde{*})_\pi\big) \tag{7}$$

$$\vee\, ((\dot{b},\dot{b})_\pi \wedge \bigcirc(b,b)_\pi \wedge \bigcirc\bigcirc(a,\#)_\pi \wedge \bigcirc\bigcirc\bigcirc(\#,\#)_\pi)\Big) \tag{8}$$

$$\wedge\, \square(\bigcirc\bigcirc\bigcirc(\tilde{a},*)_\pi \rightarrow (\tilde{a},*)_{\pi'}) \tag{9}$$

$$\wedge\, \square(\bigcirc\bigcirc\bigcirc(\tilde{b},*)_\pi \rightarrow (\tilde{b},*)_{\pi'}) \tag{10}$$

$$\wedge\, \square(\bigcirc\bigcirc\bigcirc(\#,*)_\pi \rightarrow (\#,*)_{\pi'}) \tag{11}$$

$$\wedge\, \square(\bigcirc\bigcirc(*,\tilde{a})_\pi \rightarrow (*,\tilde{a})_{\pi'}) \tag{12}$$

$$\wedge\, \square(\bigcirc\bigcirc(*,\tilde{b})_\pi \rightarrow (*,\tilde{b})_{\pi'}) \tag{13}$$

$$\wedge\, \square(\bigcirc\bigcirc(*,\#)_\pi \rightarrow (*,\#)_{\pi'})\Big) \tag{14}$$

Figure 2 Formula in the reduction of the PCP instance from Example 13, encoding that a trace may start with a valid stone 3 and that there must also exist a trace where stone 3 is deleted.

the α and β component of a stone may differ in its length, a sequence of stone representations might "overlap". Therefore, we indicate the start of a new stone with a dotted symbol. For example, we can string the first stone two times together: $(\dot{a},\dot{b}),(\dot{a},b)(\#,a)(\#,\dot{b})(\#,b)(\#,a)$. In the following, we write $\tilde{a}$ if we do not care if this symbol is an a or $\dot{a}$ and use $*$ as syntactic sugar for an arbitrary symbol of the alphabet. We assume that only singletons are allowed as elements of the trace, which could be achieved by adding for every atomic proposition (y_1,y_2) the conjunction $\bigwedge_{(y_1,y_2)\neq(y,y')} \square(\neg((y_1,y_2)\wedge(y,y')))$, for all (y,y').

▶ **Example 13.** Consider, again, the following PCP instance with $\Sigma = \{a,b\}$ and two lists α, with $\alpha_1 = a$, $\alpha_2 = ab$ and $\alpha_3 = bba$ and β, with $\beta_1 = baa$, $\beta_2 = aa$ and $\beta_3 = bb$. We can reduce this PCP instance to the question whether the HyperLTL formula shown in Figure 1 is satisfiable. Let $AP \coloneqq (\{a,b,\dot{a},\dot{b}\}\cup\{\#\})^2$. The stone encoding is sketched with the example of stone 3 in Figure 2.

The subformula (1) expresses 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 stone whose α and β component start with the same symbol. Subformula (2) requires that there exists a "solution" trace π_s. It ensures that the trace ends synchronously with $(\#,\#)^\omega$. Combined, this guarantees that the word constructed from the α components is equal to the word constructed from the β components, i.e., $i_\alpha = i_\beta$ for a PCP solution $i(k)$. Subformula (3) ensures that every trace eventually ends with the termination symbol $\#$. It is important to notice here that all traces besides π_s are allowed to end asynchronously.

It remains to ensure that trace π_s only consists of valid stones. This is where the $\forall\exists$ structure of the quantifier prefix comes into play. The key idea is to use a $\forall\exists$ formula to specify that for every trace with at least one stone there is another trace *with the first stone removed*. Since we check that *every* trace begins with a valid stone, this implies that all stones are valid. The encoding of stone 3 is exemplarily shown in Figure 2. The first *three* α components and the first *two* β components of the new trace are deleted. The example set shown in Figure 3 shows this behavior for π_s, which starts with stone 3. By deleting stone 3 from π_s and shifting every position accordingly, we obtain π_s'. Since π_s' starts with a valid stone, namely stone 2, it satisfies subformula (4) for $i = 2$. This requires that there exists

another trace where stone 2 is deleted analogously. This argument is repeated until the trace is reduced to $(\#,\#)^\omega$, which is the only possibility for "termination" in the sense that π_s ends synchronously with $(\#,\#)^\omega$.

Corresponding to this example, we can give a generalized reduction, establishing the undecidability of $\forall\exists$ formulas.

▶ **Theorem 14.** $\forall\exists$ *HyperLTL-SAT is undecidable.*

Proof. Let a PCP instance with $\Sigma = \{a_1, a_2, ..., a_n\}$ and two lists α and β be given. We choose our alphabet as follows: $\Sigma' = (\Sigma \cup \{\dot{a}_1, \dot{a}_2, ..., \dot{a}_n\} \cup \#)^2$, where we use the dot symbol to encode that a stone starts at this position of the trace. Again, we write $\tilde{a}$ if we do not care if this symbol is an a or $\dot{a}$ and use $*$ as syntactic sugar for an arbitrary symbol of the alphabet. We encode the idea from Example 13 in the following formula.

$$\varphi_{\text{reduc}} \coloneqq \forall\pi\exists\pi_s\exists\pi'.\ \varphi_{\text{sol}}(\pi_s) \wedge \varphi_{\text{validStone}}(\pi) \wedge \varphi_{\text{delete}}(\pi,\pi') \wedge \Diamond\Box(\#,\#)_\pi$$

- $\varphi_{\text{sol}}(\pi_s) \coloneqq (\bigvee_{i=1}^{n}(\dot{a}_i,\dot{a}_i)_{\pi_s}) \wedge (\bigvee_{i=1}^{n}(\tilde{a}_i,\tilde{a}_i)_{\pi_s})\,\mathcal{U}\,\Box(\#,\#)_{\pi_s}$
  We ensure that there exists a "solution" trace π_s, which starts pointed, i.e., where the α and β components are the same. Accordingly to PCP, we require *synchronous* "termination".
- $\varphi_{validStone}(\pi)$. This is ensured by a generalization of lines (7) and (8) of the stone encoding sketched in Figure 2.
  Every trace in the trace set starts with a valid stone. Note that we do *not* require synchronous termination in any other trace than the "solution" trace.
- $\varphi_{\text{delete}}(\pi,\pi')$. This is ensured by a generalization of lines (9) to (14) of the stone encoding sketched in Figure 2.
  By exploiting the $\forall\exists$ structure of the formula, we encode that for every trace π there exists another trace π' which is *nearly* an exact copy of π but with its first stone *removed*.

Correctness. We prove correctness of the reduction by showing that if there exists a solution, namely an index sequence $i(l)$ with $l \in \mathbb{N}$, for a PCP instance, then there exists a trace set T satisfying the resulting formula φ_{reduc} and vice versa. For the sake of readability, again, we omit the set braces around atomic propositions, since we can assume that only singletons occur.

- Assume there exists a solution i to the given PCP instance with $|i_\alpha| = |i_\beta| = k$. We can construct a trace set T by building the trace $(i_\alpha[0], i_\beta[0]) \dots (i_\alpha[k], i_\beta[k])(\#,\#)^\omega$, 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. We distinguish two cases. If the solution is only of length 1, we add $(\#,\#)^\omega$ to T and successfully constructed a trace set satisfying the formula. Otherwise, let t_0 start with stone j. We also add one of the following traces t_1 based on t_0 to T:

  $$\text{if } |\alpha_j| = |\beta_j| : (i[|\alpha_j|], i[|\beta_j|]) \dots (i[k], i[k])(\#,\#)^\omega$$
  $$\text{if } |\alpha_j| < |\beta_j| : (i[|\alpha_j|], i[|\beta_j|]) \dots (i[k], i[k - |\beta_j| + |\alpha_j|]) \dots (\#, i[k])(\#,\#)^\omega$$
  $$\text{if } |\alpha_j| > |\beta_j| : (i[|\alpha_j|], i[|\beta_j|]) \dots (i[k - |\alpha_j| + |\beta_j|], i[k]) \dots (i[k], \#)(\#,\#)^\omega$$

  We repeat adding traces t_n based on the starting stone of 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}.

Start

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

Delete stone 3

π'_s : $(\dot{a},\dot{a})(b,a)(\dot{b},\dot{b})(b,b)(a,\dot{b})(\dot{a},a)(\#,a)(\#,\#)(\#,\#)\dots$

Delete stone 2

π''_s : $(\dot{b},\dot{b})(b,b)(a,\dot{b})(\dot{a},a)(\#,a)(\#,\#)(\#,\#)\dots$

Delete stone 3

π'''_s : $(\dot{a},\dot{b})(\#,a)(\#,a)(\#,\#)(\#,\#)\dots$

Delete stone 1

π''''_s : $(\#,\#)(\#,\#)\dots$

End

Figure 3 Trace set satisfying the formula from Example 13, with omitted set braces around the atomic propositions.

Let the formula φ_{reduc} be satisfiable by a trace set T. Therefore, there exists a witness t_0 for π_s, which starts with a dot, whose α and β components are the same at all positions, and which ends *synchronously* with $(\#,\#)^\omega$. t_0 also needs to start with a valid stone, which is ensured by the stone encoding, since otherwise $t_0 \notin T$. By construction there exists a subset $T_{\text{min}} \subseteq T$ that satisfies φ_{reduc}, which contains t_0 and every trace constructed by deleting one stone after another, with the last trace being $(\#,\#)^\omega$. Because t_0 eventually terminates synchronously with $(\#,\#)$, 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 $A = [t_0, t_1, \dots, t_n]$ be the list of traces in T_{min} sorted in descending order. A possible solution for the PCP instance is the index sequence $s(t_0)\ s(t_1)\dots s(t_n)$.

Since we can use the construction from Subsection 3.2, the minimal undecidable fragment of HyperLTL is, in fact, $\forall\exists$. ◀

6 Conclusion

We have analyzed the decidability and complexity of the satisfiability problem for various fragments of HyperLTL. The largest decidable fragment of HyperLTL is the EXPSPACE-complete $\exists^*\forall^*$ fragment; the alternation-free $\exists^*$ and $\forall^*$ formulas are PSPACE-complete; any fragment that contains the $\forall\exists$ formulas is undecidable. Despite the general undecidability, our results provide a strong motivation to develop a practical SAT checker for HyperLTL. The key result is the PSPACE-completeness for the alternation-free fragment and the bounded $\exists^*\forall^*$ fragment, which means that for the important class of hyperproperties that can be expressed as a HyperLTL formula with a bounded number of exclusively universal or exclusively existential quantifiers, satisfiability and implication can be decided within the same complexity class as LTL.

There are several directions for future work. An important open question concerns the extension to branching time. HyperLTL is a sublogic of the branching-time temporal logic HyperCTL* [3]. While the undecidability of HyperLTL implies that HyperCTL* is also, in

general, undecidable (this was already established in [3]), the obvious question is whether it is possible to establish decidable fragments in a similar fashion as for HyperLTL.

Another intriguing, and still unexplored, direction is the synthesis problem for HyperLTL (and HyperCTL*) specifications. In synthesis, we ask for the existence of an implementation, which is usually understood as an infinite tree that branches according to the possible inputs to a system and whose nodes are labeled with the outputs of the system. Since HyperLTL can express partial observability, the synthesis problem for HyperLTL naturally generalizes the well-studied synthesis under incomplete information [8] and the synthesis of distributed systems [10].

Finally, it will be interesting to develop a practical implementation of the constructions presented in this paper and to use this implementation to analyze the relationships between various hyperproperties studied in the literature.

References

1 Rajeev Alur, Pavol Cerný, and Steve Zdancewic. Preserving secrecy under refinement. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2006. doi:10.1007/11787006_10.

2 Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *17th IEEE Computer Security Foundations Workshop, CSFW-17 2004, 28-30 June 2004, Pacific Grove, CA, USA*, pages 100–114. IEEE Computer Society, 2004. doi:10.1109/CSFW.2004.17.

3 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

4 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

5 Bernd Finkbeiner and Christopher Hahn. Deciding Hyperproperties. *CoRR*, abs/1606.07047, 2016. URL: http://arxiv.org/abs/1606.07047.

6 Bernd Finkbeiner and Markus N. Rabe. The linear-hyper-branching spectrum of temporal logics. *it - Information Technology*, 56(6):273–279, 2014. URL: http://www.degruyter.com/view/j/itit.2014.56.issue-6/itit-2014-1067/itit-2014-1067.xml.

7 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2015. doi:10.1007/978-3-319-21690-4_3.

8 Orna Kupferman and Moshe Vardi. Synthesis with incomplete informatio. In *Advances in Temporal Logic*, pages 109–127. Springer, 2000.

9 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

10 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA,*

October 22-24, 1990, Volume II, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.

11 Emil L Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

12 Markus N. Rabe. *A Temporal Logic Approach to Information-flow Control*. PhD thesis, Saarland University, 2016.

13 A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*, pages 114–127. IEEE Computer Society, 1995. doi:10.1109/SECPRI.1995.398927.

14 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.

15 Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi:10.1006/inco.1994.1092.

16 Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *16th IEEE Computer Security Foundations Workshop, CSFW-16 2003, 30 June - 2 July 2003, Pacific Grove, CA, USA*, page 29. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.