A Temporal Logic for Hyperproperties

Bernd Finkbeiner[1], Markus N. Rabe[1], and César Sánchez[2,3]

1 **Saarland University, Saarbrücken, Germany**
`{finkbeiner,rabe}@cs.uni-saarland.de`
2 **IMDEA Software Institute, Madrid, Spain**
`cesar.sanchez@imdea.org`
3 **Institute for Information Security, CSIC, Spain**

Abstract

Hyperproperties, as introduced by Clarkson and Schneider, characterize the correctness of a computer program as a condition on its *set* of computation paths. Standard temporal logics can only refer to a single path at a time, and therefore cannot express many hyperproperties of interest, including noninterference and other important properties in security and coding theory. In this paper, we investigate an extension of temporal logic with explicit path variables. We show that the quantification over paths naturally subsumes other extensions of temporal logic with operators for information flow and knowledge. The model checking problem for temporal logic with path quantification is decidable. For alternation depth 1, the complexity is PSPACE in the length of the formula and NLOGSPACE in the size of the system, as for linear-time temporal logic.

Submitted to CSL'13 on April 15, 2013

1 Introduction

The automatic verification of computer systems against specifications in temporal logic is one of the great success stories of logic in computer science. A controversial question of this area, with a continuous discussion in the literature since the early 1980s, is which temporal logic is most suitable to describe and verify the relevant properties of interest. Traditionally, this discussion has been framed as a choice between linear and branching time [36]. The linear time paradigm of logics like linear-time temporal logic (LTL) describes the correct behavior as a set of sequences [27]. This view allows us to express important temporal properties, like invariants and eventualities, but ignores possible dependencies between different executions of the system. The branching time paradigm of logics like computation tree logic (CTL) describes the correct behavior in terms of multiple possible futures and therefore allows to express of the existence of certain computations [10]. Emerson and Halpern's CTL* logic [13] unifies the linear and branching time view by combining path formulas with the path *quantifiers* E and A, which allows to continue on an existentially or universally chosen path.

An important limitation, common to all these temporal logics, is that they only refer to a *single* path at a time and therefore cannot express requirements that relate *multiple* paths. In particular, this limitation excludes *information flow* properties as they are studied in security and in coding theory. For example, *noninterference* [16], the property that certain secrets remain hidden from an observer, relates *all* paths that have (possibly) different secrets but otherwise identical input. Noninterference requires that all such paths are observationally equivalent to each other. In coding theory, a typical requirement is that the encoder of a communication protocol *preserves the entropy* of the input, which means that there does not exist a *pair of paths* with different input but identical encoding. Clarkson and Schneider coined the term *hyperproperties* [11], defined as sets of legal sets of behaviors, for this more general class of properties. That is, a correct system must produce exactly one of the legal combinations of behaviors described by the property.

In this paper we study how to extend temporal logics to hyperproperties. We generalize the path quantifiers E and A to $\exists\pi$ and $\forall\pi$, respectively, where π is a *path variable* that may be referred to in the subsequent path formula. As a result, we obtain a generalization of CTL*, which we call HyperCTL. HyperCTL provides a uniform logical framework for linear-time and branching-time hyperproperties. In this paper, we show that the quantification over paths naturally subsumes related extensions of temporal logic that have previously been studied in the literature, such as the *hide* operator of SecLTL [12], and the *knowledge* operator of temporal epistemic logic [14, 34]. Additionally, HyperCTL can express a rich set of properties that have previously been studied intensively, particularly in the security community, but as isolated properties. HyperCTL provides a common logical framework that allows to customize and compose these properties.

The paper is structured as follows. We introduce HyperCTL in Section 2. In Section 3, we relate HyperCTL to other logics and show that the quantification over paths subsumes operators for information flow and knowledge. The *model checking* and *satisfiability* problems of HyperCTL are discussed in Section 4. In Section 5, we show applications of HyperCTL in *security*, including noninterference, declassification, and quantitative information flow. We conclude with a discussion of open problems in Section 6.

2 HyperCTL

In this section, we introduce HyperCTL, our extension of the temporal logic CTL* for hyperproperties. We begin with a quick review of standard notation for Kripke structures.

2.1 Preliminaries: Kripke structures

A *Kripke structure* $K = (S, s_0, \delta, \mathsf{AP}, L)$ consists of 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 labeling function $L : S \to 2^{\mathsf{AP}}$. We require that all states $s \in S$ have at least one successor, $\delta(s) \neq \emptyset$. The *cardinality* or *size* of a Kripke structure is defined as the cardinality of its states and transitions $|K| = |S| \cdot |\mathsf{AP}| + |\delta|$, where δ is interpreted as a set of tuples. A *path* $\pi \in (S \times 2^{\mathsf{AP}})^\omega$ is an infinite sequence of labeled states (i.e., states equipped with a set of atomic propositions). For a path π and $i \in \mathbb{N}$, $\pi(i)$ is the i-th state of the sequence and $\pi[i]$ is the i-th set of atomic propositions. Furthermore, we use $\pi[0, i]$ to denote the prefix of π up to (including) position i, and $\pi[i, \infty]$ to denote the suffix starting at position i. We lift these notions to tuples of paths by applying the operations for each position: e.g. $\langle \pi_1, \dots, \pi_n \rangle [i, \infty] = \langle \pi_1[i, \infty], \dots, \pi_n[i, \infty] \rangle$. Every Kripke structure K gives rise to a set of paths from any state $s \in S$, denoted by $\mathsf{Paths}_{K,s}$. To allow the case of a specification referring to atomic propositions that are not contained in the atomic propositions AP of the Kripke structure, we assume that these atomic propositions are not restricted. Therefore, we define $\mathsf{Paths}^{\mathsf{AP}'}_{K,s} = \big\{(s_1, P_1 \cup P'_1).(s_2, P_2 \cup P'_2).\dots \mid (s_1, P_1).(s_2, P_2).\dots \in \mathsf{Paths}_{K,s},\ \forall i.\ P'_i \subseteq \mathsf{AP}' \setminus \mathsf{AP}\big\}$.

2.2 Syntax

We extend the syntax of CTL* with path variables. Path variables are introduced by path quantifiers. Since formulas may refer to multiple paths at the same time, atomic propositions are indexed with the path variable they refer to. The formulas of HyperCTL are defined by the following grammar, where $a \in \mathsf{AP}$ for some set AP of atomic propositions and $\pi \in \mathcal{V}$ for an infinite supply $\mathcal{V}$ of path variables:

$$\varphi \ ::= \quad a_\pi \quad | \quad \neg\varphi \quad | \quad \varphi \vee \varphi \quad | \quad \bigcirc\varphi \quad | \quad \varphi\,\mathcal{U}\,\varphi \quad | \quad \exists\pi.\,\varphi$$

A formula is *closed* if all occurrences of some path variable π are in the scope of a path quantifier $\exists\pi$. A HyperCTL *specification* is a Boolean combination of closed HyperCTL formulas each beginning with a quantifier (or its negation). A formula is in *negation normal form* (NNF) if it only contains negations in front of labels and in front of quantifiers. Every HyperCTL formula can be transformed into an equivalent formula in negation normal form, if we introduce the additional operators $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$ and $\varphi_1 \mathcal{R} \varphi_2 \equiv \neg(\neg\varphi_1 \mathcal{U} \neg\varphi_2)$.

We extend HyperCTL with *universal* path quantification $\forall\pi.\ \varphi \equiv \neg\exists\pi.\neg\varphi$ as well as with the usual temporal operators: $\mathsf{true} \equiv a_\pi \vee \neg a_\pi$, $\mathsf{false} \equiv \neg\mathsf{true}$, $\Diamond\varphi \equiv \mathsf{true}\,\mathcal{U}\,\varphi$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$, $\varphi_1 \mathcal{W} \varphi_2 \equiv \varphi_1\,\mathcal{U}\,\varphi_2 \vee \Box\varphi_1$. For convenience, we also introduce some abbreviations for comparing paths. Given a set $P \subseteq \mathsf{AP}$ of atomic propositions, we abbreviate $\pi[0] =_P \pi'[0] \ \equiv\ \bigwedge_{a \in P} a_\pi \leftrightarrow a_{\pi'}$, and $\pi =_P \pi' \ \equiv\ \Box(\pi[0] =_P \pi'[0])$, and analogously for $\neq$, e.g., $\pi \neq_P \pi' \ \equiv\ \neg(\pi =_P \pi')$. Finally, $[\varphi]_\pi$ indicates that the atomic propositions in φ are indexed with the path variable π, and $[\varphi]_{\pi\to\pi'}$ is the formula obtained by replacing index π by index π' on all (free) atomic propositions in φ.

2.3 Semantics

Let $K = (S, s_0, \delta, \mathsf{AP}, L)$ be a Kripke structure and φ a HyperCTL formula over atomic propositions AP'. We define $\Pi \models_K \varphi$, the satisfaction relation of φ with respect to a set of paths $\Pi = \langle \pi_1, \ldots, \pi_m \rangle$, as follows:

$\Pi \models_K a_{\pi_k}$	whenever	$a \in \pi_k[0]$
$\Pi \models_K \neg\psi$	whenever	$\Pi \not\models_K \psi$
$\Pi \models_K \psi_1 \vee \psi_2$	whenever	$\Pi \models_K \psi_1$ or $\Pi \models \psi_2$
$\Pi \models_K \bigcirc\psi$	whenever	$\Pi[1,\infty] \models_K \psi$
$\Pi \models_K \psi_1\,\mathcal{U}\,\psi_2$	whenever	$\exists i \geq 0.\ \Pi[i,\infty] \models_K \psi_2 \quad \text{and} \quad \forall 0 \leq j < i.\ \Pi[j,\infty] \models_K \psi_1$
$\Pi \models_K \exists\pi.\ \psi$	whenever	$\exists \pi \in \mathsf{Paths}^{\mathsf{AP}'}_{K,\pi_n[0]}.\ \Pi + \pi \models_K \psi$,

where $\Pi + \pi$ is short for $\langle \pi_1, \ldots, \pi_n, \pi \rangle$. For the special case that the set of paths is empty, $\Pi = \langle\rangle$, we set $\mathsf{Paths}_{K,\pi_n[0]} = \mathsf{Paths}_{K,s_0}$. We say that a Kripke structure K *satisfies* a HyperCTL formula φ, denoted as $K \models \varphi$, if $\langle\rangle \models_K \varphi$ holds true. The *model checking problem* for HyperCTL is to decide whether a given Kripke structure satisfies a given HyperCTL formula.

2.4 Examples

HyperCTL has a broad range of applications. In this subsection, we review a few motivating examples to illustrate the expressivity of HyperCTL. Due to space limitations we focus on applications in security and only sketch a few examples of other areas, such as the Hamming distance of messages in coding theory.

For simplicity, we consider a synchronous system model[1] for the remainder of this section. The system reads input variables $I \subseteq \mathsf{AP}$ and writes output variables $O \subseteq \mathsf{AP}$ at every step; for simplicity we assume that all variables are binary. Since we are not interested in the

[1] HyperCTL can also express meaningful properties for asynchronous systems.

internal behavior of systems, we specify our properties only in terms of their input-output behavior.

Security

Informally, most secrecy properties essentially require that secret inputs shall not influence the observable behavior of a system. We assume that the inputs are partitioned into subsets H and L, that indicate the secret and the public variables (often called *high* and *low* in the literature), and that the observer may only observe the output variables O. Noninterference attempts to captures this notion of secrecy, which can be written in HyperCTL, simply requiring that the output may only depend on the public inputs:

$$\forall \pi. \forall \pi'.\ (\pi[0] =_O \pi'[0])\ \mathcal{W}\ (\pi[0] \neq_I \pi'[0]).$$

This notion can be reinterpreted as a requirement of the system to produce the same output for all paths up to a point in which a difference in the public input excuses differences in the observable behavior. This reinterpretation enables the expression a basic form of *declassification* by extending the "excuse", using the following HyperCTL pattern:

$$\forall \pi. \forall \pi'.\ (\pi[0] =_O \pi'[0])\ \mathcal{W}\ (\pi[0] \neq_I \pi'[0]\ \vee\ \varphi),$$

where φ would be instantiated by a simple LTL formula (for example on π) requiring that the user is logged in. This pattern can express for example that the user may only see data if he is logged in, and if he is logged out, he may not receive any further updates. For a more formal discussion of security definitions we refer to Section 5.

Coding Theory

Following the seminal work of Shannon, many encoding schemes have been designed to provide error-resistant transmission of data. However, the formal verification of functional correctness of implementations has received little attention so far. HyperCTL can easily express important functional correctness criteria for encoders and decoders.

We first define the fundamental notion of Hamming distance. The Hamming distance is the number of positions of two sequences of the same length that differ. The formula Ham_P expresses that two sequences of states have a Hamming distance of at most d, considering only differences in the propositions $P \subseteq \mathsf{AP}$:

$$\begin{aligned} \mathrm{Ham}_P(0, \pi, \pi') &= \pi =_P \pi' , \\ \mathrm{Ham}_P(d, \pi, \pi') &= \pi[0] =_P \pi'[0]\ \mathcal{W}\ \big(\pi[0] \neq_P \pi'[0]\ \wedge\ \bigcirc(\mathrm{Ham}_P(d-1, \pi, \pi'))\big) \end{aligned}$$

The Hamming distance is often used as a metric for the error resistance of a code. If all paths of a code have a minimal Hamming distance of d, the decoder can retrieve the original word even if up to $\lfloor \frac{d-1}{2} \rfloor$ errors happened.

For the scenario in which the encoder transforms an input stream, represented by propositions I into codewords that are represented by propositions C, we can specify that an encoder produces only code words that have minimal Hamming distance d:

$$\forall \pi. \forall \pi'.\ \ \pi \neq_I \pi' \rightarrow \neg \mathrm{Ham}_C(d-1, \pi, \pi')\ .$$

Similarly, we could formulate a correctness condition for decoders. For every code word, and every message that differs from that code word in less than $\lfloor \frac{d-1}{2} \rfloor$ positions, the decoder must produce the same decoded message. Note that these formulations are completely independent of parameters like the block length and thus can be applied to compare different kinds of coding schemes.

Edit distance

Similarly to the Hamming distance, we can define other metrics based on the insertion or deletion of elements. The following formula fragment requires that paths π and π' differ only in that π' has an additional position:

$$\pi[0] =_P \pi'[0] \; \mathcal{W} \; \big(\pi[0] \neq_P \bigcirc \pi'[0]\big) \; .$$

We use this observation in Section 5 to encode security properties for non-synchronous systems.

3 Related logics

3.1 Linear- and Branching-Time Temporal Logics

HyperCTL is an extension of CTL*, and therefore subsumes the usual temporal logics LTL, CTL, and CTL* [13, 22].

▶ **Theorem 1.** *HyperCTL subsumes LTL, CTL, and CTL*.*

A more interesting case is quantified propositional temporal logic (QPTL) [31]. QPTL also extends LTL with quantification, but HyperCTL quantifies over paths and QPTL over propositions. In terms of expressiveness, we prove now that HyperCTL strictly subsumes QPTL. Informally, quantification over paths is more powerful than quantification over propositions, because the set of paths depends on the Kripke structure, while the truth values of the quantified propositions do not. We will also exploit in Section 4 the relation between HyperCTL and QPTL to obtain a model checking algorithm for HyperCTL. QPTL formulas are generated by the following grammar, where $p \in \mathsf{AP}$:

$$\psi \quad ::= \quad p \;\mid\; \neg\psi \;\mid\; \psi \vee \psi \;\mid\; \bigcirc\psi \;\mid\; \Diamond\psi \;\mid\; \exists p.\, \psi \; ,$$

QPTL formulas are interpreted over paths with all operators inheriting the LTL semantics except $\exists \pi . \psi$:

$$\pi \models \exists p.\, \psi \qquad \text{whenever} \qquad \exists \pi' \in (2^{\mathsf{AP}})^\omega .\; \pi =_{\mathsf{AP}\setminus p} \pi' \;\wedge\; \pi' \models \psi \; .$$

A Kripke structure satisfies a QPTL formula if all paths from the initial state satisfy the formula.

▶ **Theorem 2.** *HyperCTL subsumes QPTL.*

Proof. (*sketch*) In order to express a QPTL formula in HyperCTL, we rename all bound propositions with unique fresh names AP'. We use these propositions as free variables, which are unconstrained because they do not occur in the Kripke structure. Then, each propositional quantification $\exists p$ in the QPTL formula is replaced by a path quantification $\exists \pi_p$ in the HyperCTL formula, and each occurrence of p is replaced by p_{π_p}. ◀

▶ **Theorem 3.** *QPTL does not subsume HyperCTL.*

Proof. (*sketch*) Since QPTL formulas express path properties, which are checked on all paths, QPTL cannot express properties that express the *existence* of paths such as $\exists \pi . \bigcirc p_\pi$. ◀

EQCTL* and QCTL*

The logic EQCTL* [21,22] extends CTL* by the *existential* quantification over propositions. QCTL* [15] additionally allows for negated existential (i.e. universal) quantifiers. However, in contrast with our work, EQCTL* and QCTL* do not unify the quantification over paths and the quantification over propositions. Similar to the result on satisfiability of HyperCTL, the satisfiability of QCTL* is highly undecidable, which is shown using a different proof technique. To the best of the authors' knowledge, no model checking algorithm has been proposed for QCTL*. It is straightforward to derive one from the work in this paper. A restricted version of QCTL*, building on CTL instead of CTL*, allows for a comparably low worst-case complexity of the model checking problem [26]. It is open whether this result can be transferred to HyperCTL.

3.2 Epistemic Logics

Epistemic logics [14] reason about the *knowledge* of agents and have gained increasing popularity as specification languages for security over the past years [3,8,35]. Since the semantics of epistemic logics relates multiple possible worlds (paths in the setting of temporal properties), these logics define hyperproperties. We now study the relation between epistemic logics and HyperCTL.

Epistemic temporal logic with perfect recall semantics [3, 14, 34] extends LTL with the *knowledge* operator $\mathsf{K}_i\psi$, which specifies that an observer i may *know* ψ. This logic is defined on *interpreted systems*. An interpreted system $I = (R, L, Q, \mathsf{AP})$ for m agents (or processors) consists of a set of possible observations Q, a set of runs $R \subseteq (Q^m)^\omega$, and a labeling function $L : R \times \mathbb{N} \to 2^{\mathsf{AP}}$. In general, each agent can only observe *changes* to its current observation. Let $r|_n$ be the prefix of a run, up to and including step n, and let $r|_{n,i}$ be the prefix of a run projected to the states of an agent i. The knowledge of an agent then corresponds to $\mathit{trace}(r|_{n,i})$, which is defined for $n = 1$ as $\mathit{trace}(q_i) = q_i$, and otherwise as $\mathit{trace}(r|_{n-2,i}.q_i.q_i') = \mathit{trace}(r|_{n-2,i}.q_i)$ if $q_i = q_i'$ and $\mathit{trace}(r|_{n-2}.q_i.q_i') = \mathit{trace}(r|_{n-2,i}.q_i).q_i'$ if $q_i \neq q_i'$.

Under the *synchronous time assumption* [14], agents may observe the occurrence of every step. We capture this assumption by requiring that the observations of all agents include the clock, e.g. $Q = Q' \times \mathbb{N}$, and let the runs provide every agent with the correct step number. We then have that $\mathit{trace}(r|_{n,i}) = r|_{n,i}$.

For a given run $r \in R$ and a step number n, the satisfaction relation for the knowledge operator is defined as follows:

$$r, n \models \mathsf{K}_i\psi \quad \text{whenever} \quad \forall r' \in R.\ \mathit{trace}(r|_{n,i}) = \mathit{trace}(r'|_{n,i}) \ \rightarrow\ r', n \models \psi$$

An interpreted system $\mathcal{I} = (R, L, Q, \mathsf{AP})$ induces a unique Kripke structure $K(\mathcal{I}) = (S, s_0, \delta, \mathsf{AP}', L')$ with $S = Q^m \times \mathbb{N} \cup \{s_0\}$, where s_0 is a fresh initial state, and the runs R define the transition relation δ. For the initial state s_0, $\delta(s_0)$ leads to the all initial states of the interpreted system, which are defined by all prefixes of length 1 of some run in R. The new set of propositions $\mathsf{AP}' = \mathsf{AP} \times Q^m$ includes local observations for all agents, to later encode their knowledge in HyperCTL. The resulting labeling function is unique.

Note that for finite interpreted systems, that is if Q and AP are finite, L is only dependent on the current states and the runs R are generated by a relation on Q^m. Hence, the resulting Kripke structure is finite as well.

Similarly, we can define the semantics of the knowledge operator on Kripke structures.

We have that $\pi, i \models \mathsf{K}_P\varphi$ iff

$$\forall \pi' \in \mathsf{Paths}_K.\ \mathit{trace}(\pi[0,i], P) =_P \mathit{trace}(\pi'[0,i], P) \ \rightarrow\ \pi', i \models \varphi\ ,$$

where $\mathit{trace}(\pi, P)$ (the type being $\mathit{trace} : S^* \times 2^{\mathsf{AP}} \rightarrow S^*$) is the projection to the points in π where propositions in P change. For example, $\mathit{trace}(s_1.s_2.s_3.s_4.s_4) = s_1.s_3$, assuming $L(s_1) = \{a\}$, $L(s_2) = \{a, c\}$, $L(s_3) = \{a, b, c\}$, $L(s_4) = \{a, b\}$, and $P = \{a, b\}$.

The following theorem shows that, under the synchronous time assumption, HyperCTL subsumes epistemic temporal logic.

▶ **Theorem 4.** *For every epistemic temporal logic formula ψ with synchronous time assumption and interpreted system $\mathcal{I}$, there is a HyperCTL formula φ such that $\mathcal{I} \models \psi$ iff $K(\mathcal{I}) \models \varphi$.*

Proof. We start by considering extension HyperCTL with the *knowledge* operator with the semantics above, and prepend a universal path quantifier. Then we apply a step-wise transformation to eliminate all knowledge operators.

Let φ be a HyperCTL formula in NNF and in prenex normal form (note that epistemic logic formulas prepended with a universal path quantifier are in prenex normal form). For brevity we summarize the leading quantifiers of φ with $\mathbf{Q}$, such that $\varphi = \mathbf{Q}.\varphi'$ where φ' is quantifier free. Let t and u be propositions that φ does not refer to.

For the case that a knowledge operator $\mathsf{K}_P\psi$ occurs with positive polarity, we translate φ into the following HyperCTL formula:

$$\begin{aligned}&\mathbf{Q}.\exists\pi.\ \varphi'|_{\mathsf{K}_P\psi \rightarrow u_\pi}\ \wedge\ \big(\forall\pi'.\ (t_{\pi'}\,\mathcal{U}\Box\neg t_{\pi'})\ \rightarrow \\ &\quad \forall\pi''.\ \Box(t_{\pi'} \rightarrow (\pi_n[0] =_P \pi''[0]))\ \rightarrow\ \Box(t_{\pi'} \wedge u_\pi \rightarrow [\psi]_{\pi''})\big)\ ,\end{aligned}$$

and, if the knowledge operator occurs negatively,

$$\begin{aligned}&\mathbf{Q}.\exists\pi.\ \varphi'|_{\neg\mathsf{K}_P\psi \rightarrow u_\pi}\ \wedge\ \big(\forall\pi'.\ (t_{\pi'}\,\mathcal{U}\Box\neg t_{\pi'}) \rightarrow \\ &\quad \exists\pi''.\ \Box(t_{\pi'} \rightarrow (\pi_n[0] =_P \pi''[0]))\ \wedge\ \Diamond(t_{\pi'} \wedge u_\pi \wedge \neg[\psi]_{\pi''})\big)\ ,\end{aligned}$$

where $\varphi'|_{\mathsf{K}_P\psi \rightarrow u_\pi}$ denotes that in φ' one positive or one negative occurrence, respectively, of the knowledge operator $\mathsf{K}_P\psi$ gets replaced by proposition u. We repeat this transformation until no knowledge operators remain, using the fact that every HyperCTL formula can be transformed into prenex normal form.

The intuition of this transformation is to label the path (more precisely the tuple of paths Π that is determined by the quantifiers $\mathbf{Q}$) with binary signals t and u that indicate when to check that ϕ has to hold on all (or some) alternative paths. By replacing the knowledge operator by the existentially quantified proposition u, we select a satisfying interpretation (if there is any) among the disjunctions concerning the application of the knowledge operator. Hence, proposition u_π at position i indicates whether to check that all paths that are equal to π_n up to position i must satisfy $\Pi, i \models_K \psi$. We express that by universally selecting a position, indicated by the point at which $t_{\pi'}$ switches from true to false (up to that point the paths must be equal) and then requiring that when we reach that position and the flag u_π is set, we also have to check ψ on path φ.

The negated case follows the same intuition on the signals u and t, except that the same (universal) quantifier cannot be reused to select the alternative path and the signal t. ◀

Without the synchronous time assumption, epistemic temporal logic is still subsumed by HyperCTL, if we add *stutter steps* to the modified Kripke structure; let $K'(\mathcal{I})$ denote the adapted Kripke structure. We give details on the modified transformation and the encoding of the formula in Appendix A.

▶ **Theorem 5.** *For every epistemic temporal logic formula ψ and interpreted system $\mathcal{I}$, there is a HyperCTL formula φ such that $\mathcal{I} \models \psi$ iff $K'(\mathcal{I}) \models \varphi$.*

3.3 SecLTL

SecLTL [12] extends temporal logic with the *hide* modality $\mathcal{H}$, which allows us to express information flow properties such as noninterference [16]. The semantics of SecLTL is defined in terms of labeled transition system, where the edges are labeled with evaluations of the set of variables. The formula $\mathcal{H}_{H,O}\varphi$ specifies that the current evaluations of a subset H of the input variables I are kept secret from an attacker who may observe variables in O until the release condition φ becomes true. The semantics is formalized in terms of a set of *alternative paths* to which the *main path* is compared:

$$\mathsf{AltPaths}(\pi, H) = \{\pi' \in \mathsf{Paths}_{K,\pi[0]} \mid \pi[1] =_{I\setminus H} \pi'[1] \wedge \pi[2,\infty] =_I \pi'[2,\infty]\}$$

A path π satisfies the SecLTL formula $\mathcal{H}_{H,O}\varphi$, denoted by $\pi \models \mathcal{H}_{H,O}\varphi$, iff

$$\forall \pi' \in \mathsf{AltPaths}(\pi, H).\ \big(\pi =_O \pi' \ \vee\ \exists i \geq 0.\ (\pi[i,\infty] \models_K \varphi) \ \wedge\ \pi[1, i-1] =_O \pi'[1, i-1]\big)\ .$$

A transition system M satisfies a SecLTL formula ψ, denoted by $M \models \psi$, if every path π starting in the initial state satisfies ψ. To encode the hide modality in HyperCTL, we first translate M into a Kripke structure $K(M)$ whose states are labeled with the evaluation of the variables on the edge leading into the state. The initial state is labeled with the empty set. In the modified system, $L(\pi[1])$ corresponds to the current labels. We encode $\mathcal{H}_{H,O}\varphi$ as the following HyperCTL formula:

$$\forall \pi'.\ \pi[1] =_{I\setminus H} \pi'[1] \wedge \bigcirc\big(\pi[1] =_O \pi'[1]\ \mathcal{W}\ (\pi[1] \neq_I \pi'[1] \vee \varphi)\big)$$

▶ **Theorem 6.** *For every SecLTL formula ψ and transition system M, there is a HyperCTL formula φ such that $M \models \psi$ iff $K(M) \models \varphi$.*

The model checking problem for SecLTL is PSPACE-hard in the size of the Kripke structure [12]. The encoding of SecLTL specifications in HyperCTL implies that the model checking problem for HyperCTL is also PSPACE-hard (for a fixed specification of alternation depth ≥ 2), as claimed in Theorem 9.

4 Model Checking and Satisfiability

4.1 Model Checking

In this section we exploit the connection between HyperCTL and QPTL to obtain a model checking algorithm for HyperCTL. We reduce the model checking problem for HyperCTL to the satisfiability problem for QPTL. First, we encode the Kripke structure as a QPTL formula. Then we qualify all quantifiers to refer to the interpretations of the propositions that occur on the corresponding path in the Kripke structure. The satisfiability problem for the resulting QPTL formula is nonelementary in the quantifier alternation depth.

▶ **Definition 7** (Alternation Depth). A HyperCTL or QPTL formula φ in NNF has alternation depth 1 plus the highest number of alternations from existential to universal and universal to existential quantifiers along any of the paths of the formula's syntax tree. Occurrences of $\mathcal{U}$ and $\mathcal{R}$ count as an additional alternation.

▶ **Theorem 8.** *The model checking problem for HyperCTL specifications φ with alternation depth k is complete for NSPACE($g(k-1, |\varphi|)$).*

Proof. The satisfiability problem for QPTL formulas φ in prenex normal form and with alternation depth k is complete for NSPACE($g(k-1, |\varphi|)$) [31].

For the upper bound on the HyperCTL model checking complexity, we encode Kripke structures as QPTL formulas (see e.g. [19, 23]) and translate until operators $\psi \,\mathcal{U}\, \psi'$ as

$$\exists t.\ t \wedge \Box(t \rightarrow \psi' \vee (\psi \wedge \bigcirc t)) \wedge \neg \Box\, t\ .$$

HyperCTL path quantifiers are encoded by extending the set of atomic propositions, introducing for each quantifier an additional copy of AP. That is, assuming that all paths have a unique name, we translate $\exists \pi_k.\ \psi$ to $\exists \mathsf{AP}_k.[\psi]_{\pi_k \rightarrow \mathsf{AP}_k}$, where $[\psi]_{\pi_k \rightarrow \mathsf{AP}_k}$ means that we let the atomic propositions a_{π_k} in ψ refer to the proposition a in the k-th copy of AP.

For the lower bound, we reduce the satisfiability problem for a given QPTL formula φ in prenex normal form to a model checking problem $K \models \varphi'$ for some Kripke structure K and some HyperCTL formula φ'. We assume, without loss of generality, that φ is closed (if a free proposition occurs in φ, we bind it with an existential quantifier) and each quantifier in φ introduces a different proposition.

The Kripke structure $K = (S, s_0, \delta, \mathsf{AP}, L)$ consists of two states $S = \{s_0, s_1\}$, is fully connected $\delta : s \mapsto S$ for $s \in S$, and is labeled with the truth values of a single atomic proposition $\mathsf{AP} = \{p\}$, $L : s_0 \mapsto \emptyset, s_1 \mapsto \{p\}$.

The QPTL quantifiers choose a valuation of the quantified propositions already in the first position of a sequence while the path quantifiers of HyperCTL can only pick a new state in the *next* position. Hence, we shift all atomic propositions in φ by one position, by replacing every atomic proposition q in φ by $\bigcirc q$, resulting in the new QPTL formula ψ. The final HyperCTL formula $\varphi' = T_1(\psi)$ is constructed recursively from ψ as follows:

$$\mathcal{T}_m(\psi) = \begin{cases} \exists \pi_m.\ \mathcal{T}_{m+1}(\psi'[t \mapsto p_{\pi_m}]) & \text{if } \psi = \exists t.\psi' \\ \neg\exists \pi_m.\ \mathcal{T}_{m+1}(\psi'[t \mapsto p_{\pi_m}]) & \text{if } \psi = \neg\exists t.\psi' \\ \psi & \text{otherwise} \end{cases}$$

◀

An NLOGSPACE lower bound in the size of the Kripke structure for fixed specifications with alternation depth 1 follows from the non-emptiness problem of non-deterministic Büchi automata. For alternation depth 2 and more we can derive PSPACE hardness in the size of the Kripke structure from the encoding of the logic SecLTL into HyperCTL (see Subsection 3.3).

▶ **Theorem 9.** *For HyperCTL formulas the model checking problem is hard for PSPACE in the size of the system.*

A Remark on Efficiency

The use of the standard encoding of the until operator in QPTL with an additional quantifier is, in certain cases, wasteful. The satisfiability of QPTL formulas can be checked with an automata-theoretic construction, where we first transform the formula into prenex normal form, then generate a nondeterministic Büchi automaton for the quantifier-free part of the formula, and finally apply projection and complementation to handle the existential and universal quantifiers. In this way, each quantifier alternation, including the alternation introduced by the encoding of the until operators, causes an exponential blow-up. However, if an until operator occurs in the quantifier-free part, the standard transformation of LTL formulas to nondeterministic Büchi automata could handle this operator without quantifier elimination, which would result in an exponential speedup.

Using this insight, the model checking complexity for many of the formulas presented above and in Section 5 could be reduced by one exponent. Additionally, the complexity with respect to the size of the system would reduce to NLOGSPACE for HyperCTL formulas where the leading quantifiers are all of the same type and are followed by some quantifier-free formula which may contain until operators without restriction.

4.2 Satisfiability

Unfortunately, the positive results regarding the decidability of the model checking problem for HyperCTL do not carry over to the satisfiability problem. The *finite-state satisfiability problem* consists of the existence of a finite model, while the *general satisfiability problem* asks for the existence of a possibly infinite model. The following theorem shows that both versions are undecidable.

▶ **Theorem 10.** *For HyperCTL, finite-state satisfiability is hard for Σ_1^0 and general satisfiability is hard for Σ_1^1.*

Proof. We give a reduction from the synthesis problem for LTL specifications in a distributed architecture consisting of two processes with disjoint sets of variables. The synthesis problem consists on deciding whether there exist transition systems for the two processes with input variables I_1 and I_2, respectively, and output variables O_1 and O_2, respectively, such that the synchronous product of the two transition systems satisfies a given LTL formula φ. This problem is hard for Σ_1^0 if the transition systems are required to be finite, and hard for Σ_1^1 if infinite transition systems are allowed (Theorems 5.1.8 and 5.1.11 in [28]).

To reduce the synthesis problem to HyperCTL satisfiability, we construct a HyperCTL formula ψ as a conjunction $\psi = \psi_1 \wedge \psi_2 \wedge \psi_3$. The first conjunct ensures that φ holds on all paths: $\psi_1 = \forall\pi.[\varphi]_\pi$. The second conjunct ensures that every state of the model has a successor for every possible input: $\forall\pi.\square \bigwedge_{I\subseteq I_1\cup I_2} \exists\pi' \bigcirc \bigwedge_{i\in I} i \ \bigwedge_{i\notin I} \neg i$. The third conjunct ensures that the output in O_1 does not depend on I_2 and the output in O_2 does not depend on I_1: $\psi_3 = \forall\pi.\forall\pi'.\ \big(\pi =_{I_1} \pi' \rightarrow \pi =_{O_1} \pi'\big) \wedge \big(\pi =_{I_2} \pi' \rightarrow \pi =_{O_2} \pi'\big)$.

The distributed synthesis problem has a (finite) solution iff the HyperCTL formula ψ has a (finite) model. ◀

5 Applications in Security

Information flow is a much studied field in security [1–5, 11, 16, 18, 20, 24, 25, 30, 33, 38, 40]. The question of interest is whether a system reveals information about a cryptographic key or personal data, generally called *the secret*, to an observer or attacker. This problem goes beyond classical data flow analysis, in the sense that not only forbids the system from revealing the secret as one of its outputs, but it also requires that the system's observable behavior is independent from the secret.

For example, consider the case of a hash set that stores both public and secret elements. If an attacker observes the list of public elements in the set he may learn partial information about the presence of secret elements and their values by observing the ordering of the low elements. Even though in this case a security attack is not immediate and it might only work under additional assumptions, secrets in the data structure may be compromised. Practical attacks based on such partial information have been implemented. For example, the well known attack on SSL [7] uses subtle timing differences in the answers from a server.

The notion of *noninterference* by Goguen and Meseguer is the starting point of most of the bast body of work on information flow. This notion states the secrecy requirement in terms of two groups of users, which represent the secret (their behavior must be secret) and the attacker, respectively. The group of attackers may not observe a different behavior of the system, independent of whether the secret users perform actions or not.

▶ **Theorem 11.** *There is a HyperCTL formula $\varphi_{NI}(G_H, G_L)$ and an encoding $K(M)$ of state machines into Kripke structures such that for every state machine M, and groups of users G_H and G_L, G_H does not interfere with G_L in M iff $K(M) \models \varphi_{NI}(G_H, G_L)$ holds.*

The encoding of state machines into Kripke structures and the proof are straightforward and can be found in Appendix B.

A multitude of variations of the notion of secrecy have been proposed [1,2,4,20,24,25, 30,40]. Even though hyperproperties [11] provide a useful semantical framework to capture a broad range of these information flow properties, [11] does not provide any algorithmic approaches. The technique of self-composition [5] can be used to cast the verification of these information flow properties in terms of safety (and sometimes in terms of liveness) problems [33,38] by *modifying the system.* However, these encodings can be rather involved (cf. [17]) and hence, for many of the information flow variations, no algorithms exist yet that allow for their precise verification.

In contrast, HyperCTL allows us to easily derive such algorithms for many of the proposed notions of information flow without the need to modify the system. In the rest of this section, we illustrate the expressiveness of HyperCTL by showing how to define noninterference and also more modern definitions of security[2]. From the results of Section 4.1 it follows that the formulas presented in this section yield efficient model checking algorithms, particularly considering the remark on efficiency. These examples suggest that our common logical framework of HyperCTL may also lead to advances for the practical verification of security policies.

5.1 Security Definitions for Nondeterministic Systems

A common concern of many pieces of work following the seminal work of Goguen and Meseguer is the extension of noninterference to nondeterministic systems (e.g. [24,25,39,40]). Different approaches differ in their assumptions on the source of nondeterminism and discuss what requirements would make a nondeterministic system intuitively secure.

Observational determinism [25, 40] is a conservative approach to nondeterminism that assumes that the attacker does no only know the model, but can choose an arbitrary implementation (refinement) thereof. Essentially, the property requires that for a given input all possible executions of the system look identical for an observer independently of the choice of the secret and of how the nondeterminism is resolved. Huisman et al. [17] encoded this information flow property as a CTL* formula after transforming the system via self-composition [5]. HyperCTL can naturally express properties over multiple paths and thus the property can be expressed without the need for modifications to the system. See Appendix C for details.

[2] The purpose of this section is not to argue for or against particular information flow policies.

▶ **Theorem 12.** *There is an encoding of programs as defined in [17] into Kripke structures, such that HyperCTL can express observational determinism.*

The proof shows that self-composition can be replaced by quantifying over two paths.

5.2 Security Lattices

It is common in security to allow for multiple hierarchically ordered security levels that are (statically) assigned to users and/or data. HyperCTL can express these concerns by a conjunction of multiple secrecy statements that separate the security levels. The formulation of security lattices is independent of the precise notion of secrecy chosen.

5.3 Declassification

Many security critical systems reveal secret information partially or under certain conditions as their normal mode of operation. For example, a piece of software that checks passwords must reveal whether the entered password is correct or not. Noninterference, though, categorically rules out such behavior because it leaks some information to an attacker. Thus, weaker notions of security have been proposed [2, 4, 12, 29] (see [30] for more literature). There is a variety of patterns to integrate exception into information flow policies, which are referred to as the *dimensions* of declassification [30].

The example of the password checking could be generalized to the pattern (or dimension) that a predefined fact about the secret may be released, e.g. [29]. Other patterns are that after a certain condition has been fulfilled information may be released (for example, after access has been granted) [6, 12], or that some input must be considered secret only under certain conditions (for example, the keyboard input is only secret while the cursor is in a password field) [12].

▶ **Theorem 13.** *HyperCTL subsumes the declassification properties encoded in [3].*

The result follows from the encoding [3] of declassification properties into epistemic temporal logic [14] and the fact that HyperCTL subsumes epistemic temporal logic (see Section 3). Appendix D contains a discussion of the example of a password checker above to show that declassification properties can indeed be expressed elegantly in HyperCTL.

5.4 Quantitative Information Flow

In cases where leaks cannot be prevented absolutely, we might want to limit the *quantity* of the lost information [9, 18, 20, 32, 37]. First of all, this quantification requires to measure information by some definition of *entropy*. It turns out that for different practical scenarios different notions of this measure are appropriate. To define meaningful measures of entropy of a secret, secrets are generally assumed to be chosen randomly and independently and the system under consideration is assumed to be deterministic (not nondeterministic, that is). The essence of quantitative information flow, though, can be formulated in a uniform way for all measures of entropy. All we have to assume is a fixed measure of entropy $\mathcal{H}(X)$ for a random variable X and a notion of conditional entropy $\mathcal{H}(X|Y = y)$ indicating the entropy of X under the assumption that the result of a second (possibly correlated) random experiment Y is known to be y.

Leaks can then be measured in terms of *lost entropy*, that is, the difference between the initial entropy of the secret H, and the remaining entropy when observing the output O: $\mathcal{H}(H) - \mathcal{H}(H|O)$. Formally the *bounding problem* of quantitative information flow is then defined as to determine whether this difference is bounded by a given constant [32, 37].

Min Entropy

The min entropy measure concerns the highest probability that an attacker correctly guesses the secret in a single try and it is considered to be one of the most appropriate indications of how useful the observations are for the attacker [32]. The min entropy of a random variable H is defined as $-\log \max_h p(H = h)$, where p indicates the probability measure. The conditional min entropy is defined as $-\log \sum_o p(O = o)(\max_h p(H = h|O = o))$, which, by Bayes' theorem, becomes $-\log \sum_o \max_h p(O = o|H = h)p(H = h)$.

Under the assumption that the system under consideration is deterministic $p(O = o|H = h)$ is either 0 or 1 and further $p(H = h)$ is a constant c_H if we assume the secret to be uniformly distributed. The bounding problem of quantitative noninterference for the min entropy measure and for a given bound n hence simplifies to the following question:

$$\mathrm{n} \;\geq\; -\log c_H \;+\; \log\Big(\; c_H \textstyle\sum_{o,\ \exists h.p(O=o|H=h)=1} 1 \;\Big),$$

and amounts to specifying that there is no tuple of $2^n + 1$ distinguishable paths. (see [32] for more detailed discussions.) In a synchronous system setting (cf. Section 2.4), where $I \subseteq \mathsf{AP}$ and $O \subseteq \mathsf{AP}$ refer to the atomic propositions indicating the input and output, respectively, we can, thus formulate this condition as follows:

$$\varphi_{QIF} = \neg\exists\pi_0. \;\ldots\; . \;\exists\pi_{2^n}. \;\Big(\bigwedge_i \pi_i =_I \pi_0\Big) \wedge \bigwedge_{i\neq j} \pi_i \neq_O \pi_j$$

Yasuoka and Terauchi proved that 2^{n+1} is the optimal number of paths to formulate this property [37].

6 Conclusions and Open Problems

We have introduced the temporal logic HyperCTL, which extends CTL* with path variables. HyperCTL provides a uniform logical framework for temporal hyperproperties. The quantification over paths in HyperCTL subsumes related extensions of temporal logic such as the *hide* operator of SecLTL and the *knowledge* operator of temporal epistemic logic. Beyond the properties expressible in these temporal logics, HyperCTL expresses a rich set of properties from security and coding theory, such as noninterference and the preservation of entropy. The encoding of these properties in HyperCTL is natural and efficient: for many security properties the complexity of the HyperCTL model checking algorithm matches the lower bounds known for the individual properties. HyperCTL thus not only provides a common semantic foundation for properties from a diverse class of applications but also a uniform algorithmic solution.

An important open question concerns the identification of efficient fragments of HyperCTL, motivated in particular by the significant gap between the PSPACE-complete model checking problem for CTL* and the non-elementary model checking problem of general HyperCTL formulas.

From a semantic perspective, it is worth noting that the expressiveness of HyperCTL exceeds the original definition of hyperproperties [11]. While Clarkson and Schneider's hyperproperties are sets of sets of sequences, and thus have their semantic foundation in the linear-time paradigm of sequences, HyperCTL integrates, like CTL*, the linear-time view with the branching-time view. When path quantifiers occur in the scope of temporal operators, the HyperCTL formula expresses that the new paths branch off in the same state of the Kripke structure. A full investigation of the linear vs branching time spectrum of hyperproperties and the implications on the complexity, compositionality, and usability of logics for hyperproperties are intriguing challenges for future research.

References

1 T. Amtoft and A. Banerjee. Information flow analysis in logical form. In *Proc. of SAS'04*, pages 100–115, 2004.

2 A. Askarov and A. Myers. A semantic framework for declassification and endorsement. In *Proc. ESOP'10*, pages 64–84. Springer, 2010.

3 M. Balliu, M. Dam, and G. Le Guernic. Epistemic temporal logic for information flow security. In *Proc. of PLAS'11*, 2011.

4 A. Banerjee, D. A. Naumann, and S. Rosenberg. Expressive declassification policies and modular static enforcement. In *IEEE Symp. on Security and Privacy*, pages 339–353, 2008.

5 G. Barthe, P.R. D'Argenio, and T. Rezk. Secure information flow by self-composition. In *Proc. of CSFW'04*, pages 100–114. IEEE Computer Society Press, 2004.

6 N. Broberg and D. Sands. Flow locks: Towards a core calculus for dynamic flow policies. In *Proc. ESOP'06*, volume 3924 of *LNCS*. Springer, 2006.

7 D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proc. of USENIX Security Symp.*, volume 12, pages 1–1. USENIX, 2003.

8 R. Chadha, S. Delaune, and S. Kremer. Epistemic Logic for the Applied Pi Calculus. In *Proc. of FMOODS/FORTE'09*, pages 182–197. Springer, 2009.

9 D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. In *Proc. QAPL'01*, volume 59 of *ENTCS*, pages 238–251. ENTCS, 2002.

10 E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, pages 52–71. Springer, 1982.

11 M. R. Clarkson and F. B. Schneider. Hyperproperties. *J. of Computer Security*, 18(6):1157–1210, 2010.

12 R. Dimitrova, B. Finkbeiner, M. Kovács, M. N. Rabe, and H. Seidl. Model checking information flow in reactive systems. In *Proc. of VMCAI'12*, pages 169–185. Springer, 2012.

13 E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *JACM*, 33:151–178, 1986.

14 R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge.* MIT Press, 1995.

15 T. French. Decidability of quantifed propositional branching time logics. In *Proceedings of AI*, pages 165–176. Springer, 2001.

16 J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symp. on Security and Privacy*, pages 11–20, 1982.

17 M. Huisman, P. Worah, and K. Sunesen. A temporal logic characterisation of observational determinism. In *Proc. CSFW'06.* IEEE Computer Society, 2006.

18 J. W. Gray III. Toward a mathematical foundation for information flow security. In *Proc. of the IEEE Symp. on Security and Privacy*, pages 210–34, may 1991.

19 Y. Kesten and A. Pnueli. Complete proof system for QPTL. *J. Log. Comput.*, 12(5):701–745, 2002.

20 B. Köpf and D. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proc. of CCS'07*, pages 286–296. ACM, 2007.

21 O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. In *Proc. of CAV'95*, pages 325–338. Springer, 1995.

22 O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *JACM*, 47(2):312–360, 2000.

23 Z. Manna and A. Pnueli. *The Temporal Logic of Reactive Systems: Specification.* The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, 1992.

24 H. Mantel. Possibilistic definitions of security - an assembly kit. In *Proc. of CSFW'00*, pages 185–199, 2000.

25 J. McLean. Proving noninterference and functional correctness using traces. *J. of Computer Security*, 1:37–58, 1992.

26 A.C. Patthak, I. Bhattacharya, A. Dasgupta, P. Dasgupta, and P.P. Chakrabarti. Quantified computation tree logic. *Information Processing Letters*, 82(3):123 – 129, 2002.

27 A. Pnueli. The temporal logic of programs. In *Proc. of FOCS'77*, pages 46–57, 1977.

28 R. Rosner. *Modular synthesis of reactive systems.* PhD thesis, Weizmann Institute, 1992.

29 A. Sabelfeld and A. C. Myers. A model for delimited information release. In *Proc. of ISSS*, pages 174–191. Springer, 2004.

30 A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proc. of CSFW'05*, pages 255–269. IEEE Computer Society, 2005.

31 A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with appplications to temporal logic. *TCS*, 49:217–237, 1987.

32 G. Smith. On the foundations of quantitative information flow. In *Proc. of FOSSACS'09*, pages 288–302. Springer, 2009.

33 T. Terauchi and A. Aiken. Secure information flow as a safety problem. In *Proc. SAS'05*, pages 352–367, 2005.

34 R. van der Meyden. Axioms for knowledge and time in distributed systems with perfect recall. In *Proc. LICS'93*, pages 448–457, 1993.

35 R. van der Meyden and T. Wilke. Preservation of epistemic properties in security protocol implementations. In *Proc. of TARK'07*, pages 212–221. ACM, 2007.

36 M. Y. Vardi. Branching vs. linear time: Semantical perspective. In *Proc. CSL'11*, page 3, 2011.

37 H. Yasuoka and T. Terauchi. On bounding problems of quantitative information flow. In *Proc. of ESORICS'10*, pages 357–372. Springer, 2010.

38 H. Yasuoka and T. Terauchi. Quantitative information flow as safety and liveness hyperproperties. In *Proc. of QAPL'12*, pages 77–91, 2012.

39 A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Proc. of the IEEE Symp. on Security and Privacy*, pages 94–102. IEEE Computer Society Press, 1997.

40 S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *Proc. of CSFW'03*, 2003.

A Epistemic Logics Without the Synchronous Time Assumption

We now discuss the adaptations necessary to extend the encoding from the proof of Theorem 4 to non-synchronous systems. The following proof of Theorem 5 is also a good example for the general treatment of asynchronous executions in HyperCTL.

▶ **Theorem 5.** *For every epistemic temporal logic formula ψ and interpreted system $\mathcal{I}$, there is a HyperCTL formula φ such that $\mathcal{I} \models \psi$ iff $K'(\mathcal{I}) \models \varphi$.*

Proof. First, we define the model transformation $K'(\mathcal{I})$, which differs from $K(\mathcal{I})$ in that it adds stuttering steps. In the resulting Kripke structure, we indicate for each transition whether it is due to a step of the interpreted system or if it is a newly added stuttering step.

For a given interpreted system $\mathcal{I}$, the stuttering Kripke structure $K'(\mathcal{I}) = (S, s_0, \delta, \mathsf{AP}, L)$ is defined via the non-stuttering Kripke structure $K(\mathcal{I}) = (S', s'_0, \delta', \mathsf{AP}', L')$, where $S = S' \times \{\text{stutter}, \text{move}\}$, $s_0 = (s_0, \text{stutter})$, $\delta(\langle s, x\rangle) = \delta'(s) \times \{\text{move}\} \cup \{\langle s, \text{stutter}\rangle\}$, $\mathsf{AP} = \mathsf{AP} \cup \{\text{stutter}, \text{move}\}$, and $L(\langle s, x\rangle) = L'(s) \cup \{x\}$.

In $K'(\mathcal{I})$, the path quantifiers range over a larger set of paths that we have to restrict to those that do not stutter forever ($\mathsf{progress}(\pi) = \Box\Diamond \text{move}_\pi$) and to those pairs of paths that synchronize correctly. The correct synchronization of two paths π and π' means that the observations regarding some set of propositions P changes in one path iff it changes also in the other path: $\mathsf{synch}(\pi, \pi') = \Box\Big(\pi[0] \neq_P \bigcirc\pi[0] \quad\Leftrightarrow\quad \pi'[0] \neq_P \bigcirc\pi'[0]\Big)$.

The encoding from the proof of Theorem 4 is now modified as follows for positively occurring knowledge operators:

$$\begin{aligned}&\mathbf{Q}.\exists\pi.\ \varphi'|_{\mathsf{K}_P\psi\to u_\pi}\ \wedge\ \big(\forall\pi'.\ (t_{\pi'}\,\mathcal{U}\Box\neg t_{\pi'})\ \rightarrow\ \forall\pi''.\ \mathsf{progress}(\pi'')\ \wedge\ \mathsf{synch}(\pi,\pi'')\ \wedge\\ &\quad \Box(t_{\pi'} \rightarrow (\pi_n[0]{=_P}\,\pi''[0]))\ \rightarrow\ \Box(t_{\pi'} \wedge u_\pi \rightarrow [\psi]_{\pi''})\big)\ ,\end{aligned}$$

and, for negatively occurring knowledge operators:

$$\begin{aligned}&\mathbf{Q}.\exists\pi.\ \varphi'|_{\mathsf{K}_P\psi\to u_\pi}\ \wedge\ \big(\forall\pi'.\ (t_{\pi'}\,\mathcal{U}\Box\neg t_{\pi'}) \rightarrow \exists\pi''.\ \mathsf{progress}(\pi'')\ \wedge\ \mathsf{synch}(\pi,\pi'')\ \wedge\\ &\quad \Box(t_{\pi'} \rightarrow (\pi_n[0]{=_P}\,\pi''[0]))\ \wedge\ \Diamond(t_{\pi'} \wedge u_\pi \wedge \neg[\psi]_{\pi''})\big)\ ,\end{aligned}$$

For a positively occurring knowledge operator $\mathcal{K}_P\psi$, we prove that given two executions π_1 and π_2, if the sub-formula ψ must hold on π_2 at some position i (by the semantics of the knowledge operator), then the formula above requires that on a stuttered version of π_1 the sub-formula is applied at state $\pi_2[i]$. We consider the prefixes of the two executions, $\pi_1[0,i]$ and $\pi_2[0,j]$ and assume they have the same traces with respect to $P \subseteq \mathsf{AP}$. For these, there are two stuttered versions π and π'' (named to match the path variables in the formula above) that synchronize the positions at which they change their observations with respect to P and pad the prefixes to the same length k (without changing their final states, that is $\pi_2[i] = \pi''[k]$). There is also a labeling with $t_{\pi'}$ that ensures that the knowledge operator is evaluated until position k. Hence, ψ is applied on π'' at state $\pi_2[i]$.

The other direction is straightforward, and the case of a negatively occurring knowledge operator follows similarly. ◀

B Goguen and Meseguer's Noninterference

B.1 Noninterference

A point of reference for most of the literature on information flow is the definition of *noninterference* that was introduced by Goguen and Meseguer in 1982 [16]. In this subsection, we show how to express noninterference in a simple HyperCTL formula.

The system model used in [16], which we will refer to as *deterministic state machines*, operates on commands $c \in C$ that are issued by different users $u \in U$. The evolution of a deterministic state machine is governed by the transition function $\mathsf{do} : S \times U \times C \to S$ and there is a separate observation function $\mathsf{out} : S \times U \to \mathit{Out}$ that for each user indicates what he can observe.

We define standard notions on sequences of users and events. For $w \in (U \times C)^*$ and $G \subseteq U$ let $|w|_G$ denote the projection of w to the commands issued by the users in G. Further, we extend the transition function do to sequences, $\mathsf{do}(s, (u, c).w) = \mathsf{do}(\mathsf{do}(s, u, c), w)$, where the dot indicates concatenation. Finally, we extend the observation function out to sequences w, indicating the observation *after* w: $\mathsf{out}(w, G) = \mathsf{out}(\mathsf{do}(s_0, w), G)$. *Noninterference* is then defined as a property on systems M. A set of users $G_H \subseteq U$ does not interfere with a second group of users $G_L \subseteq U$, if:

$$\forall w \in (U \times C)^*.\ \mathsf{out}(w, G_L) = \mathsf{out}(|w|_{G_H}, G_L)\ .$$

That is, we ask whether the same output would be produced by the system, if all actions issued by any user in G_H were removed.

B.2 Encoding GM's System Model

First, we need to map their system model into Kripke structures as used for the formulation of HyperCTL (which, obviously, must not solve problem by itself). We choose an intuitive encoding of state machines in Kripke structures that indicates in every state (via atomic propositions) which observations can be made for the different users, and also which action was issued last, including the responsible user.

We choose a simple translation that maps a state machine $M = (S, U, C, \mathit{Out}, \mathsf{out}, \mathsf{do}, s_0)$ to the Kripke structure $K = (S', s'_0, \delta, \mathsf{AP}, L)$, where $S' = \{s_0\} \cup S \times U \times C$, $s'_0 = s_0$, $\mathsf{AP} = U \times C\ \cup\ U \times \mathit{Out}$, and the labeling function is defined as $L(s_0) = \{(u, \mathsf{out}(s_0, u)) \mid u \in U\}$ for the initial state and $L((s, u, c)) = \{(u, c)\} \cup \{(u', \mathsf{out}(s, u')) \mid u' \in U\}$ for all other states.

The transition function is defined as

$$\delta(s, u, c) = \{(s', u', c') \mid \mathsf{do}(s, u', c') = s',\ u' \in U,\ c' \in C\}$$

Each state (except for the initial state) has labels indicating the command that was issued last, and the user that issued the command. The remaining labels denote the observations that the individual users can make in this state.

To access these two separate pieces of information, we introduce functions $\mathsf{in} : S \to U \times C$, which is not defined for s_0, and $\mathsf{out} : S \times U \to \mathit{Out}$ (by abusing notation slightly), with their obvious meanings.

In this system model let $s =_{U \setminus G_H} s'$ mean that if we entered one of the states s and s' with a user not in G_H the users and commands must be identical. If both states are entered by users in G_H then the commands may be different. The output equality on states $s =_{O,G_L} s'$, shall refer to the observations the users G_L can make at this state. Then, noninterference can be expressed as follows:

$$\begin{aligned}\varphi_{NI}(G_H, G_L) \ =\ & \forall \pi. \forall \pi'.\ \pi[0] =_{U \setminus G_H} \pi'[0]\ \ \mathcal{W}\ \ \Big(\pi[0] \neq_{U \setminus G_H} \pi'[0]\ \wedge \\ & \qquad (\ \pi[0] \in H\ \wedge\ \Box(\bigcirc\pi[0] =_{U \setminus G_H} \pi'[0])\ \implies\ \Box(\bigcirc\pi[0] =_{O,G_L} \pi'[0])\ )\Big)\ ,\end{aligned}$$

▶ **Theorem 14.** *There is a HyperCTL formula $\varphi_{NI}(G_H, G_L)$ and an encoding $K(M)$ of state machines into Kripke structures such that for every state machine M, and groups of users G_H and G_L it holds $K(M) \models \varphi_{NI}(G_H, G_L)$ iff G_H does not interfere with G_L in M.*

Proof. The formula pattern $\pi[0] =_{U\setminus G_H} \pi'[0] \, \mathcal{W} \, \big(\pi[0] \neq_{U\setminus G_H} \pi'[0] \ \wedge \ \varphi\big)$ implies that φ is applied exactly at the first position at which π and π' differ in their input (except, possibly, on the input of users in G_H). As, for a fixed path π, we quantify over all paths π' the subformula φ is hence applied to every position of every path π.

The subformula φ then requires that, if from that position on π's input differs from the input of π' only in that it has an additional (secret) action by some user in G_H, both paths must look equivalent from the view point of the users in G_L. The interesting part here is the use of the next operator, as it enables the comparison of different positions of the traces. Thus, we compare path π to all other paths that have one secret action less than itself. Hence, by transitivity of equivalence, we compare all paths π to a version of itself that is stripped of all secret actions. ◀

C Observational Determinism

▶ **Theorem 12.** *There is an encoding of programs as defined in [17] into Kripke structures, such that HyperCTL can express observational determinism.*

Proof. Huisman et al. [17] define observational determinism over programs in a simple while language, which is very similar to the execution model of [40]. First, they define a special version of self-composition on their programs that allows both sides to move independently by introduces stuttering steps. Then, they encode programs into Kripke structures that do not only maintain the current state, but also remember the last state. The CTL* formula is then proved to precisely express observational determinism. Since HyperCTL subsumes CTL*, we can make use of the same encoding of programs into Kripke structures, but we leave out the self-composition operation on programs. In order to re-enable the correct synchronization of paths, however, we have to introduce stutter steps for single, not self-composed that is, programs.

We prepend the CTL* formula with two path quantifiers over paths π_1 and π_2 and we let those propositions of the formula that referred to the two copies of the program now refer to the two paths, respectively. ◀

D An Example of a Declassification Property

Although we have shown that declassification properties can be encoded in principle, the two-step encoding via epistemic logics, will not necessarily lead to a practical verification approach. Further, the encoding via epistemic logics [3] assumes a non-reactive system model (it has no inputs) and thus its flavor of noninterference on which they build the other encodings does not subsume Goguen and Meseguer's definition. In this appendix we show with the help of an example that declassification properties can be encoded in an elegant way leading to efficient algorithmic approaches.

Let us continue to work in the setting of deterministic state machines as we used already for noninterference [16]. We consider the case of a system that shall allow a user A to set a secret password for her account names "Alice", which a user B wants to access. In order to get access to the account B would have to guess the password correctly, which we *assume* impossible without additional knowledge. We would like to specify the *external behavior* of the system in terms of the commands that allow to change passwords and access accounts. We assume that passwords are stored as hash values of length 256.

Let $C \supseteq \{\mathit{changePwd}\} \times \mathit{Accounts} \times \mathbb{B}^{256} \ \cup \ \{\mathit{login}\} \times \mathit{Accounts} \times \mathbb{B}^{256}$ and $U \supseteq \{A, B\}$, where *Accounts* refers to the domain of account names. We assume that passwords can only be changed while logged in and that user A is signed in from the beginning. Further, let $i \in \mathbb{B}^{256}$ be the initial password. Then the formulation of noninterference could be refined to compare only those paths on which the current password was not guessed correctly. First, we define the LTL property on individual paths that the initial password is not guessed until it is replaced, and that the same holds for every password that is set during the run of the system:

$$\begin{aligned}\psi = \ & \textstyle\bigwedge_{x\in\mathbb{B}^{256}}(\neg\langle B,\langle \mathit{login}, \mathit{Alice}, i\rangle\rangle\,\mathcal{U}\,\langle A,\langle \mathit{changePwd}, \mathit{Alice}, x\rangle\rangle)\ \wedge\\ & \qquad\qquad\qquad \square\big(\textstyle\bigwedge_{x,y\in\mathbb{B}^{256}}\langle A,\langle \mathit{changePwd}, \mathit{Alice}, x\rangle\rangle \implies\\ & \qquad\qquad\qquad\qquad \neg\langle B,\langle \mathit{login}, \mathit{Alice}, x\rangle\rangle\,\mathcal{U}\,\langle A,\langle \mathit{changePwd}, \mathit{Alice}, y\rangle\rangle\big)\ ,\end{aligned}$$

and then embed ψ into φ_{NI}:

$$\begin{aligned}&\forall\pi.\forall\pi'.\ [\psi]_\pi \wedge [\psi]_{\pi'} \implies \pi[0] =_{U\setminus G_H} \pi'[0]\ \ \mathcal{W}\ \Big(\pi[0] \neq_{U\setminus G_H} \pi'[0] \wedge\\ &\qquad\qquad \big(\ \pi'[0] \in H\ \wedge\ \square(\pi[0] =_{U\setminus G_H} \bigcirc\pi'[0]) \implies \square(\pi[0] =_{O,G_L} \bigcirc\pi'[0])\ \big)\Big)\ .\end{aligned}$$

Adding the assumption that the password is not simply guessed correctly essentially did not change the structure of the property. The advantage of the logical approach is that such assumptions could also be added to other definitions of secrecy, like that of observational determinism, without the need to reengineer the algorithm for their verification completely.