Special Issue

Bernd Finkbeiner* and Markus N. Rabe

The linear-hyper-branching spectrum of temporal logics

Abstract: The family of temporal logics has recently been extended with logics for the specification of hyperproperties, such as noninterference or observational determinism. Hyperproperties relate multiple computation paths of a system by requiring that they satisfy a certain relationship, such as an identical valuation of the low-security outputs. Unlike classic temporal logics like LTL or CTL*, which refer to one computation path at a time, temporal logics for hyperproperties like HyperLTL and HyperCTL* can express such relationships by explicitly quantifying over multiple computation paths simultaneously. In this paper, we study the extended spectrum of temporal logics by relating the new logics to the linear-branching spectrum of process equivalences.

Keywords: Specification, temporal logics, hyperproperties, information flow.

ACM CCS: Theory of computation → Logic → Modal and temporal logics

DOI 10.1515/itit-2014-1067
Received June 17, 2014; revised October 10, 2014; accepted October 20, 2014

1 Introduction

A *hyperproperty* [5] is a property of the behavior of a system that relates multiple computation paths. In computer security, hyperproperties have gained prominence as a formal framework for the specification of confidentiality and integrity. Prominent examples of hyperproperties are information-flow security policies like *noninterference* [10], which requires that all pairs of computations that result from different secrets produce the same externally visible output as long as the inputs are the same. Noninterference thus does not forbid any particular computation, but instead characterizes combinations of computations that may be present in the same system.

The study of hyperproperties as a class of temporal properties is a fairly new research direction. The standard spectrum of the temporal specification logics ranges from linear-time temporal logic (LTL) [13] to the computation tree logic CTL* [8] and its sublogic CTL [4]. LTL is the standard logic to describe *linear-time* properties, or properties of individual paths, such as *invariants* (“$x > y$ is true forever”) or *response* properties (“event a is always followed by event b”). CTL* adds existential and universal quantification over paths and can therefore describe *branching-time* properties, such as the *existence* of paths (“there is a path where event a occurs”). Relationships between multiple paths can, however, neither be expressed in LTL, which implicitly quantifies over a *single* path, nor in CTL*, which explicitly quantifies over the paths in a system but can, again, only refer to a single path at a time.

HyperLTL and HyperCTL* [6] are two recent additions to the family of temporal logics that address this semantic limitation. HyperLTL is an extension of LTL where the propositions can stipulate relationships between multiple computation paths. A HyperLTL formula consists of a quantifier prefix, in which several path variables are introduced by existential or universal quantifiers, and an LTL formula, in which these path variables are used to indicate for each atomic proposition a specific path in which it is to be evaluated. Analogously, HyperCTL* is an extension of CTL* with path variables, where the path quantifiers may not only occur at the beginning of the formula, but also in the scope of temporal and boolean operators. Simultaneous quantification over multiple paths subsumes [6] other modalities, such as the knowledge modality of epistemic temporal logic [9] and the hide modality of SecLTL [7].

HyperLTL and HyperCTL* are well-suited as specification languages for model checking tools, i. e., tools that check automatically whether a finite-state system satisfies a given property. Like for the standard temporal logics, the model checking problem is decidable for HyperLTL and HyperCTL*; while the logics allow, in general, for the specification of properties with expensive, even non-

***Corresponding author: Bernd Finkbeiner,** Universität des Saarlandes, Saarbrücken, e-mail: finkbeiner@cs.uni-saarland.de
Markus N. Rabe: Universität des Saarlandes, Saarbrücken

elementary, model checking problems, many properties of interest can be verified much easier. For example, model checking noninterference is NLOGSPACE in the size of the system, and thus in the same complexity class as the verification of LTL properties [6].

The linear-branching spectrum of temporal logics has been studied extensively since the 1980s [4, 12, 13]. The spectrum classifies temporal logics by the *proccess equivalences* [16] they induce. The induced process equivalence of a given logic is the equivalence that distinguishes two processes exactly if there exists a formula in the logic that is satisfied by one process but not by the other. *Trace equivalence* is the standard process equivalence of the linear-time view and *bisimulation* is the standard process equivalence of the branching-time view. Two processes are trace equivalent if their sets of traces are the same; they are bisimilar if they additionally have the same branching structure, i. e., if there exists a relation between their states that ensures that for every pair of related states, every transition from one state can be simulated by a transition from the other state such that the successors are related again. The induced process equivalences provide justification for the classification of LTL and CTL/CTL* as linear and branching-time logics, because the equivalence induced by LTL is trace equivalence, and, likewise, the process equivalence induced by CTL and CTL* is bisimulation.

The recent extension of the temporal logics to hyperproperties compels us to revisit this classification. Is HyperLTL still a linear-time logic, despite its ability to relate multiple paths? How do the two extensions to the expressiveness of LTL, to branching-time properties in CTL* and to hyperproperties in HyperLTL, relate to each other? What is gained by the combination of the two types of properties in HyperCTL*? To develop some intuition about these questions, consider the following example system consisting of three processes, where a boolean input is stored by process 1, while processes 2 and 3 write a constant output. Let us assume that input is a secret that must not be observable in the output.

Process 1: `x := input(); // boolean secret`
Process 2: `output(0);`
Process 3: `output(1);`

We analyze the system under two different scheduler models. Suppose first that the scheduler chooses in each step which process is to be executed next. This situation is depicted on the left in Figure 1. The resulting system is insecure, because, in cases where process 1 goes first, the subsequent decisions of the scheduler may *depend* on the secret. For example, it may schedule process 2 before process 3 if the secret is "0", and the other way around if the secret is "1". This situation can be detected by a HyperLTL formula that requires that all paths have the same output. This property is violated, because there is a path with output "0-1" and an observably different path with output "1-0". Suppose now that we fix the problem with a scheduler that must commit in advance to the order in which the processes are to be executed. This situation is depicted on the right in Figure 1. The secret is safe now, because the scheduling decision happens prior to the execution of process 1 and is thus *independent* of the input received by process 1. The HyperLTL formula we used to detect the problem with the previous scheduler model, however, still (incorrectly) classifies the execution tree as insecure. This is not just a problem with the specific formula we chose for this example. In this article, we show that the process equivalence induced by HyperLTL is trace equivalence, like for LTL. Since the two execution trees have the same set of traces, this implies that *no* HyperLTL formula can distinguish the two execution trees. The requirement

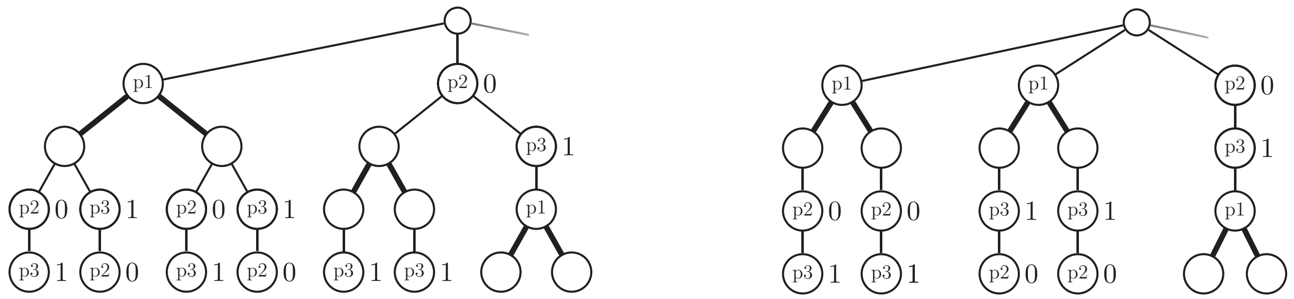


Figure 1: Execution trees of the example program under two different scheduler models. The node labels indicate the process to be executed; the labels next to nodes indicate the output. Branchings marked with thick edges represent the secret input. Other branchings represent scheduling choices. Fading edges represent scheduling choices omitted in this diagram. The left computation tree results from a scheduler model that allows the scheduler to pick in every step the process to execute next. The right tree results from a scheduler model where the scheduler must commit in advance to an ordering of the processes.

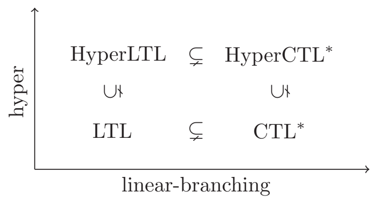


Figure 2: Linear-hyper-branching spectrum of temporal logics.

that the outputs may depend on the scheduling decisions, but not on the secret, is a branching-time property: it is only the paths of the subtrees that are rooted in some state where process 1 is about to be executed, not all paths in the execution tree, that must have the same outputs. While this property cannot be expressed in HyperLTL, it can be expressed in HyperCTL*. The process equivalence induced by HyperCTL* is bisimulation, like for CTL*.

Since the differences in expressiveness between LTL and HyperLTL, and likewise between CTL* and HyperCTL*, are not reflected in the linear-branching spectrum, the extended spectrum of temporal logics consists of two dimensions, as depicted in Figure 2. The linear-branching dimension organizes the logics according to the induced process equivalence and therefore puts LTL and HyperLTL, and CTL* and HyperCTL* into the same group. The hyper dimension classifies the logics according to their expressiveness with respect to properties that relate multiple paths, which separates HyperLTL and HyperCTL* from LTL and CTL*.

The remainder of the paper is structured as follows. After a review of basic definitions in Section 2, we traverse the spectrum of temporal logics from linear time in Section 3 to branching time in Section 4. In Section 3, we prove that HyperLTL is strictly more expressive than LTL (LTL cannot express observational determinism), but induces, like LTL, trace equivalence. In Section 4 we prove the corresponding result that HyperCTL* is strictly more expressive than CTL*, but induces, like CTL*, bisimulation. We discuss other temporal logics in Section 5.

2 System model, paths, traces

A *Kripke structure* is a tuple $K = (S, s_0, \delta, \mathsf{AP}, L)$ consisting of a set of states S, an initial state s_0, 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 each state has a successor, that is $\delta(s) \neq \emptyset$, to ensure that every execution of a Kripke structure can always be continued to infinity.

A *path* of a Kripke structure is an infinite sequence $s_0 s_1 \ldots \in S^\omega$ such that s_0 is the initial state of K and $s_{i+1} \in \delta(s_i)$ for all $i \in \mathbb{N}$. By $Paths(K, s)$ and $Paths^*(K, s)$ we denote the set of all paths of K starting in state $s \in S$ and the set of their suffixes, respectively.

A *trace* of a path $\sigma = s_0 s_1 \ldots$ is a sequence of labels $l_0 l_1 \ldots$ with $l_i = L(s_i)$ for all $i \in \mathbb{N}$. $Tr(K, s)$ (and $Tr^*(K, s)$) is the set of all (suffixes of) traces of paths of a Kripke structure K starting in state s.

3 Linear-time temporal logics

We begin our discussion of the temporal logics on the linear-time end of the spectrum.

LTL

LTL specifies properties of single traces such as “event a is always followed by event b”, which would be $\square(a \Rightarrow \Diamond b)$ in LTL syntax. LTL is generated by the following grammar:

$$\varphi \ ::= \ a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \,\mathrm{U}\, \varphi$$

where a is an *atomic proposition*, $\neg$ and $\wedge$ have the usual meaning, $\bigcirc$ denotes *next*, and U is the until operator. We also consider the usual derived Boolean operators and the derived temporal operators *eventually* $\Diamond\varphi \equiv \mathtt{true}\,\mathrm{U}\,\varphi$ and *globally* $\square\varphi \equiv \neg\Diamond\neg\varphi$.

LTL is interpreted over traces: $t \vDash a$ iff $a \in t(0)$; and $t \vDash \bigcirc\varphi$ iff $t[1, \infty] \vDash \varphi$; and $t \vDash \varphi_1 \mathrm{U} \varphi_2$ iff $\exists i \geq 0 : t[i, \infty] \vDash \varphi_2 \ \wedge \ \forall 0 \leq j < i : \ t[j, \infty] \vDash \varphi_1$. Boolean operators have the usual semantics.

Observational determinism [14, 17] is a secrecy property that is satisfied if the observable output of a system is a deterministic function of its public input, i. e., there is no pair of traces with the same public input but observably different output. Alur et al. [1] showed that observational determinism is not a regular tree property which implies that observational determinism cannot be expressed in LTL. This fact can also be shown by the following direct argument. Suppose that there is an LTL formula φ that expresses observational determinism. The set of traces T that satisfy the formula cannot be the full set of traces (with respect to some non-empty set of atomic propositions), because in that case all Kripke structures would satisfy the property. We pick a trace not in T and consider a Kripke structure that only allows this trace. Since this Kripke structure only has a single trace, it obviously satisfies observational determinism; but since that trace is not in T, it violates φ, contradicting our assumption that φ expresses observational determinism.

HyperLTL

HyperLTL [6] extends LTL with *trace quantifiers*, which allow us to *relate* multiple traces. Observational determinism, for example, can be expressed as follows: $\forall\pi.\forall\pi'.\ \Box(I_\pi = I_{\pi'}) \implies \Box(O_\pi = O_{\pi'})$, where I is the set of public inputs and O is the set of observations.

Formally, the formulas of HyperLTL are generated by the following grammar with initial symbol θ:

$$\begin{aligned}\theta &::= \exists\pi.\theta \mid \exists\pi.\varphi \mid \neg\theta\\ \varphi &::= a_\pi \mid \neg\varphi \mid \varphi\vee\varphi \mid \bigcirc\varphi \mid \varphi\,\mathrm{U}\,\varphi\end{aligned}$$

That is, HyperLTL formulas start with a quantifier prefix consisting of at least one quantifier and continue with a subformula of Boolean and temporal operators. Universal quantification is defined as $\forall\pi.\varphi \equiv \neg\exists\pi.\neg\varphi$ and $\forall\pi.\theta \equiv \neg\exists\pi.\neg\theta$.

The semantics of HyperLTL is given via trace assignments. Given a set of names N and a Kripke structure K, a *trace assignment* Π is a partial function $\Pi : N \to Tr^*(K, s_0)$. As usual, we define the update to functions: $\Pi[\pi \mapsto t](\pi) = t$ and $\Pi[\pi \mapsto t](\pi') = \Pi(\pi')$ for $\pi \neq \pi'$. The n-th element in a trace t is denoted $t(n)$ and the n-th suffix of a trace $t = a_0a_1\ldots$, written $t[n,\infty]$, is defined as the trace $a_n, a_{n+1}\ldots$ starting from the n-th label in t. We lift the suffix operation on traces to assignments and define $\Pi[i,\infty](\pi) := \Pi(\pi)[i,\infty]$.

Given a Kripke structure K, the validity of HyperLTL formulas is then defined as follows:

$$\begin{array}{lll}\Pi \vDash_K \exists\pi.\,\varphi & \text{iff} & \exists t \in Tr(K,s_0) :\ \Pi[\pi\mapsto t] \vDash_K \varphi\\ \Pi \vDash_K a_\pi & \text{iff} & a \in \Pi(\pi)(0)\\ \Pi \vDash_K \neg\varphi & \text{iff} & \Pi \nvDash_K \varphi\\ \Pi \vDash_K \varphi_1\vee\varphi_2 & \text{iff} & \Pi \vDash_K \varphi_1 \text{ or } \Pi \vDash \varphi_2\\ \Pi \vDash_K \bigcirc\varphi & \text{iff} & \Pi[1,\infty] \vDash_K \varphi\\ \Pi \vDash_K \varphi_1\,\mathrm{U}\,\varphi_2 & \text{iff} & \exists i \geq 0 :\ \Pi[i,\infty] \vDash_K \varphi_2 \text{ and}\\ & & \forall 0 \leq j < i :\ \Pi[j,\infty] \vDash_K \varphi_1\end{array}$$

Validity on states of a Kripke structure K, written $s \vDash_K \varphi$, is then defined as $\Pi_0 \vDash \varphi$, where Π_0 is the empty assignment. A Kripke structure $K = (S, s_0, \delta, \mathsf{AP}, L)$ satisfies a HyperLTL formula φ, denoted with $K \vDash \varphi$, iff $s_0 \vDash_K \varphi$.

Induced process equivalence

A logic *induces* an equivalence relation on Kripke structures that distinguishes two Kripke structures if and only if there is a formula in the logic that is satisfied by one of the two Kripke structures, but not by the other. A process equivalence on the linear-time end of the spectrum [16] is trace equivalence. Two Kripke structures K and K' are called *trace equivalent* if $Tr(K, s_0) = Tr(K', s_0')$.

LTL is a linear-time logic, because it induces trace equivalence. Given that LTL is a sublogic of HyperLTL, it is clear that its induced equivalence is at least as fine as trace equivalence. HyperLTL also cannot distinguish more than trace equivalence, because its semantics refers to the system only in terms of the set of traces starting from the initial state.

Theorem 1. *HyperLTL induces trace equivalence.*

Just like LTL, HyperLTL is thus a linear-time logic. This observation is helpful to understand the expressiveness of the logic. For example, we can immediately conclude that there is no HyperLTL formula that distinguishes the two scheduler models from the introduction, because the corresponding Kripke structures shown in Figure 1 are trace equivalent.

4 Branching-time temporal logics

Two Kripke structures may differ in their branching structure even if the set of traces is the same. Linear-time logics cannot distinguish such Kripke structures. This is a limitation, because the branching structure indicates *when* nondeterministic choices are made. The branching-time logics CTL, CTL*, and HyperCTL* use path quantifiers to distinguish Kripke structures with different branching structures.

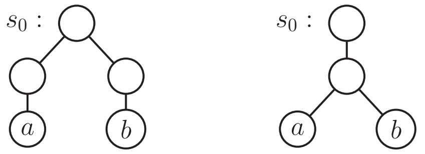
CTL/CTL*

CTL* is generated by the following grammar of state formulas Φ and path formulas φ:

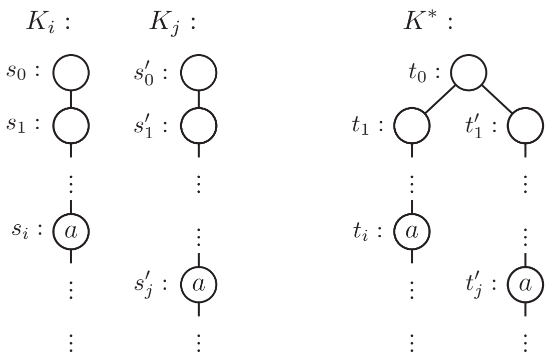
$$\begin{aligned}\Phi &::= a \mid \neg\Phi \mid \Phi\wedge\Phi \mid \mathrm{A}\,\varphi \mid \mathrm{E}\,\varphi\\ \varphi &::= \Phi \mid \neg\varphi \mid \varphi\wedge\varphi \mid \bigcirc\varphi \mid \varphi\,\mathrm{U}\,\varphi\end{aligned}$$

Again, we consider the usual derived Boolean and temporal operators. CTL is the sublogic of CTL* where every temporal operator is immediately preceded by a path quantifier. CTL* state formulas Φ are interpreted over states and path formulas φ are interpreted over paths of *a given Kripke structure* and thus have access to more information than LTL formulas. For a given Kripke structure $K = (S, s_0, \delta, \mathsf{AP}, L)$ and a state $s \in S$, we define $s \vDash_K \mathrm{A}\,\varphi$ iff $\forall p \in Paths(K, s) : p \vDash \varphi$ and symmetrically $s \vDash_K \mathrm{E}\,\varphi$ iff $\exists p \in Paths(K, s) : p \vDash \varphi$. The semantics of temporal operators for paths corresponds to their interpretation over traces in LTL.

CTL and CTL* can distinguish trace-equivalent Kripke structures that differ in their branching structure. For example, the CTL formula $\mathsf{A}\bigcirc(\mathsf{E}\bigcirc a) \wedge \mathsf{E}\bigcirc b$ distinguishes the following pair of Kripke structures:



CTL and CTL* cannot, however, express observational determinism, despite their ability to quantify over paths. In the previous section, we showed that there is no LTL formula that expresses observational determinism; we generalize the argument to CTL* as follows. Consider a family of observationally deterministic Kripke structures $K_1, K_2, \ldots$, where each K_i consists of a single branch from the initial state that only has one label a at step i:



All members of this family trivially satisfy observational determinism. We pick a pair K_i and K_j with $i \neq j$ of Kripke structures such that s_1 and s'_1 satisfy the same subformulas of φ. (We can treat path formulas as state formulas as each path uniquely corresponds to a certain state.) Such a pair of Kripke structures must exist as φ has finitely many subformulas and the family of Kripke structures is infinite. We "merge" K and K' into one Kripke structure K^*, such that they share only the initial state as depicted above. By construction, states s_1, s'_1, t_1, and t'_1 all fulfill the same subformulas. Both, t_0 and s_0, have the same label (i. e. none) and all their successors satisfy the same subformulas of φ. Hence, they also satisfy the same subformulas of φ. In particular K^* satisfies φ but not observational determinism, which contradicts the assumption.

HyperCTL*

Extending the path quantifiers of CTL* by *path variables* leads to the logic HyperCTL*, which subsumes both HyperLTL and CTL*. The formulas of HyperCTL* are generated by the following grammar:

$$\varphi ::= a_\pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi\ \mathsf{U}\ \varphi \mid \exists\pi.\ \varphi$$

We require that temporal operators only occur inside the scope of path quantifiers.

The semantics of HyperCTL* is given in terms of assignments of variables to *paths*, which are defined analogous to trace assignments. Given a Kripke structure K and a special name $\varepsilon \in N$, the validity of HyperCTL* formulas is defined as follows:

$$\begin{array}{lll}
\Pi \vDash_K a_\pi & \text{iff} & a \in L(\Pi(\pi)(0)) \\
\Pi \vDash_K \neg\varphi & \text{iff} & \Pi \nvDash_K \varphi \\
\Pi \vDash_K \varphi_1 \vee \varphi_2 & \text{iff} & \Pi \vDash_K \varphi_1 \text{ or } \Pi \vDash \varphi_2 \\
\Pi \vDash_K \bigcirc\varphi & \text{iff} & \Pi[1,\infty] \vDash_K \varphi \\
\Pi \vDash_K \varphi_1\ \mathsf{U}\ \varphi_2 & \text{iff} & \exists i \geq 0 :\ \Pi[i,\infty] \vDash_K \varphi_2 \text{ and} \\
 & & \forall 0 \leq j < i :\ \Pi[j,\infty] \vDash_K \varphi_1 \\
\Pi \vDash_K \exists\pi.\ \varphi & \text{iff} & \exists p \in \mathit{Paths}(K, \Pi(\varepsilon)(0)) : \\
 & & \Pi[\pi \mapsto p,\ \varepsilon \mapsto p] \vDash_K \varphi
\end{array}$$

The variable ε denotes the path most recently added to Π (i. e., closest in scope to π). For the empty assignment $\Pi = \{\}$, we define $\Pi(\varepsilon)(0)$ to yield the initial state. Validity on states of a Kripke structure K, written $s \vDash_K \varphi$, is defined as $\{\} \vDash \varphi$. A Kripke structure $K = (S, s_0, \delta, \mathsf{AP}, L)$ satisfies a HyperCTL* formula φ, denoted with $K \vDash \varphi$, iff $s_0 \vDash_K \varphi$.

Induced process equivalence

Since CTL* induces bisimulation [2] and is a sublogic of HyperCTL*, the induced equivalence of HyperCTL* must be at least as fine as bisimulation. A *bisimulation* for a pair of Kripke structures $K = (S, s_0, \delta, \mathsf{AP}, L)$ and $K' = (S', s'_0, \delta', \mathsf{AP}', L')$ is an equivalence relation $R \subseteq S \times S'$ on their states, such that it holds for all pairs $(s, s') \in R$ that $L(s) = L'(s')$ and for all successors $t \in \delta(s)$ of s, there exists a successor $t' \in \delta'(s')$ of s' such that $(t, t') \in R$, and vice versa. Two Kripke structures $K = (S, s_0, \delta, \mathsf{AP}, L)$ and $K' = (S', s'_0, \delta', \mathsf{AP}', L')$ are called *bisimulation equivalent* (or *bisimilar*), iff there exists a bisimulation $R \subseteq S \times S'$ and $(s_0, s'_0) \in R$.

Before we show that the equivalence induced by HyperCTL* is not finer than bisimulation, we need to lift bisimulation to paths and path assignments. In the following, let K and K' be Kripke two structures. A pair of paths $p \in \mathit{Paths}^*(K, s_0)$ and $p' \in \mathit{Paths}^*(K', s'_0)$ with $p = s_0 s_1 \ldots$ and $p' = s'_0 s'_1 \ldots$ is called *bisimilar*, written $p \sim p'$, if there is a bisimulation $\sim$ on the states of K and K' such that $s_j \sim s'_j$ for all $j \in \mathbb{N}$. A pair of path assignments $\Pi : N \to \mathit{Paths}^*(K, s_0)$ and $\Pi' : N \to \mathit{Paths}^*(K', s'_0)$ is called *bisimilar*, written $\Pi \sim \Pi'$, if they bind the same set of variables, $\Pi^{-1}(\mathit{Paths}^*(K, s_0)) = \Pi'^{-1}(\mathit{Paths}^*(K', s'_0)) = N$, and for all $\pi \in N$ it holds $\Pi(\pi) \sim \Pi'(\pi)$. In the special case of two empty path assignments, the path assignments are bisimilar iff the initial states are bisimilar.

We show by induction on the formula structure that a HyperCTL* formula has the same value in all bisimilar path assignments. This implies that HyperCTL* cannot distinguish two bisimilar Kripke structures, because the empty assignments {} are bisimilar for all bisimilar Kripke structures. Bisimilar path assignments satisfy, by definition, the same atomic propositions.The path quantifier $\exists\pi.\varphi$ selects a new path starting in the state $\Pi(\varepsilon)(0)$ and $\Pi'(\varepsilon)(0)$, respectively. Because these states are bisimilar, there is a pair of bisimilar paths starting in these states [2, Lemma 7.5]. Hence, the path assignments $\Pi[\pi \to p]$ and $\Pi'[\pi \to p']$ are again bisimilar and, by induction hypothesis, $\exists\pi.\varphi$ has the same value in $\Pi[\pi \to p] \vDash \varphi$ and $\Pi'[\pi \to p'] \vDash \varphi$. For the temporal operators $\bigcirc$ and U, we note that suffixes from identical positions of bisimilar paths are bisimilar again; hence, suffixes of bisimilar path assignments are bisimilar again. Therefore, by induction hypothesis, $\bigcirc\varphi$, and likewise $\varphi_1 \mathbin{\mathrm{U}} \varphi_2$, has the same value in bisimilar path assignments.

Theorem 2. *HyperCTL* induces bisimulation.*

Since bisimulation is strictly finer than trace equivalence, HyperCTL* is thus the strictly more expressive logic. For example, the two scheduler models discussed in the introduction, which cannot be distinguished by HyperLTL, can be distinguished by the HyperCTL* formula $\forall\pi.\ \Box\big(\mathrm{p1} \implies \forall\pi'.\forall\pi''.\ \Box(o_{\pi'} = o_{\pi''})\big)$, where o is the output. The formula expresses a condition on the states where the nondeterministic choice of interest is resolved, i. e., where Process 1 is to be executed next: for each of these states, it quantifies over pairs of paths starting in this state. Since each such paths corresponds to a choice of the secret, we require the paths to be pairwise observationally equivalent.

5 Related temporal logics

The quantification over paths in HyperLTL and HyperCTL* subsumes other extensions of temporal logic with operators for knowledge and information flow. *Epistemic temporal logic* [9, 15] is often used to specify information-flow security policies [3]. The logic refers to multiple *agents* that differ in their observational power, given as an equivalence relation on the states. The *knowledge modality* $K_i\varphi$ expresses that *agent i knows* φ. With perfect recall semantics, this means that all paths that are, for agent i, observably equivalent up to the current step, satisfy formula φ. HyperLTL subsumes epistemic temporal logic [6]. Since epistemic temporal logic includes LTL as a sublogic, it thus also induces trace equivalence and is therefore, like HyperLTL, an example of a linear-time temporal logic for hyperproperties.

Another temporal logic for the specification of hyperproperties is *SecLTL* [7]. SecLTL extends LTL with the *hide operator* $\mathcal{H}_{H,O}\varphi$, which specifies that the secret that is introduced by the *current branching* of the set of propositions H is not observable in the propositions O until the condition φ becomes true. SecLTL is subsumed by HyperCTL* [6]. It can, for example, express the branching-time property from the introduction, that the paths of subtrees that are rooted in some state where process 1 is about to be executed must have the same outputs: $\Box(\mathrm{p1} \implies \mathcal{H}_{\emptyset,\{o\}}\mathrm{false})$.

6 Conclusions

The debate about the relative benefits of linear-time vs. branching-time frameworks goes back to the 1980s. While it has been argued that the linear-time view suffices to specify the correct functionality of a system [12], many researchers have pointed out that the branching-time framework is semantically more expressive. A key insight, attributed by Robin Milner to Carl-Adam Petri, is that “information enters a non-deterministic process in finite quantities throughout time”, and that the branching-time view allows us to observe “in which states, and in what ways” this happens [11]. We need branching time, thus, to observe the *introduction of information* in a nondeterministic system.

With hyperproperties, the debate continues in a second dimension. The expressiveness added by hyperproperties differs from the expressivness added by branching-time properties. Hyperproperties allow us to track the *flow of information* in the system. In particular, we can specify under which circumstances information becomes visible on specific output variables.

While branching and hyperproperties are thus already useful individually, it is the combination in a logic like HyperCTL* that allows us to track information all the way from the point of entry, via some nondeterministic choice, to the point of exit through some externally observable variable.

References

1. R. Alur, P. Černý, and S. Zdancewic. Preserving secrecy under refinement. In *Proc. of ICALP'06*, pages 107–118, 2006.
2. C. Baier and J. P. Katoen. *Principles of model checking*. The MIT Press, 2008.
3. M. Balliu, M. Dam, and G. Le Guernic. Epistemic temporal logic for information flow security. In *Proc. of PLAS'11*, 2011.
4. 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.
5. M. R. Clarkson and F. B. Schneider. Hyperproperties. *J. of Computer Security*, 18(6):1157–1210, 2010.
6. M. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez. Temporal logics for hyperproperties. In *Proc. of POST*, 2014.
7. R. Dimitrova, B. Finkbeiner, M. Kovács, M. N. Rabe, and H. Seidl. Model checking inf. flow in reactive systems. In *Proc. of VMCAI'12*, pages 169–185, 2012.
8. E. A. Emerson and J. Y. Halpern. “Sometimes” and “not never” revisited: on branching versus linear time temporal logic. *JACM*, 33:151–178, 1986.
9. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
10. J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symp. on Security and Privacy*, pages 11–20, 1982.
11. R. Milner. What is a process?, September 2009. http://www.cs.rice.edu/~vardi/papers/milner09.pdf.
12. S. Nain and M. Y. Vardi. Branching vs. linear time: Semantical perspective. In *Proc. of ATVA*, volume 4762 of *LNCS*, pages 19–34. Springer Verlag, 2007.
13. A. Pnueli. The temporal logic of programs. In *Proc. of FOCS'77*, pages 46–57, 1977.
14. A. W. Roscoe. CSP and determinism in security modelling. In *Proc. of the IEEE Symp. on Security and Privacy*, pages 114–127. IEEE Computer Society Press, 1995.
15. R. van der Meyden and N. V. Shilov. Model checking knowledge and time in systems with perfect recall. In *FSTTCS*, volume 1738 of *LNCS*, pages 432–445. Springer, 1999.
16. R. J. van Glabbeek. The linear time – branching time spectrum. In *Proceedings of CONCUR*, volume 458 of *LNCS*, pages 278–297. Springer Verlag, 1990.
17. S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *Proc. of CSFW'03*, 2003.

Bionotes

Prof. Bernd Finkbeiner, Ph.D.
Universität des Saarlandes,
66123 Saarbrücken, Germany
finkbeiner@cs.uni-saarland.de

Bernd Finkbeiner studied Computer Science at Technische Universität München (Diplom 1996), the University of Delaware (M.Sc. 1995), and Stanford University (Ph.D. 2002). Since 2002, he is a Professor for Computer Science at Saarland University. His research interests are the specification and automatic verification and synthesis of reactive systems.

Markus N. Rabe
Universität des Saarlandes, Fachrichtung Informatik, 66123 Saarbrücken, Germany
rabe@react.uni-saarland.de

Markus N. Rabe studied Computer Science at Saarland University (B.Sc. 2008, M.Sc. 2009). Since 2010, he is a Ph.D. student in Computer Science at Saarland University. His research interests are the specification and algorithmic verification of security-relevant systems and of probabilistic systems.