EPiC Series in Computing

Volume 94, 2023, Pages 23–35

Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning

EPiC Computing

Model Checking Omega-Regular Hyperproperties with AutoHyperQ

Raven Beutner and Bernd Finkbeiner

CISPA Helmholtz Center for Information Security, Germany

Abstract

Hyperproperties are commonly used to define information-flow policies and other requirements that reason about the relationship between multiple traces in a system. We consider HyperQPTL – a temporal logic for hyperproperties that combines explicit quantification over traces with propositional quantification as, e.g., found in quantified propositional temporal logic (QPTL). HyperQPTL therefore truly captures ω-regular relations on multiple traces within a system. As such, HyperQPTL can, e.g., express promptness properties, which state that there exists a common bound on the number of steps up to which an event must have happened. While HyperQPTL has been studied and used in various prior works, thus far, no model-checking tool for it exists. This paper presents `AutoHyperQ`, a fully-automatic automata-based model checker for HyperQPTL that can cope with arbitrary combinations of trace and propositional quantification. We evaluate `AutoHyperQ` on a range of benchmarks and, e.g., use it to analyze promptness requirements in a diverse collection of reactive systems. Moreover, we demonstrate that the core of `AutoHyperQ` can be reused as an effective tool to translate QPTL formulas into ω-automata.

1 Introduction

In 2008, Clarkson and Schneider [16] coined the term *hyperproperties* for the rich class of system requirements that relate multiple computations. In their definition, hyperproperties generalize trace properties, which are sets of traces, to *sets of* sets of traces. This covers a wide range of requirements, from information-flow security policies such as non-interference [29] and observational determinism [44] to properties such as robustness [25] and promptness [38]. Missing from Clarkson and Schneider's original theory was, however, a concrete specification language that could be used as a common semantic foundation and, e.g., implemented in model-checking tools that automatically verify a system against a hyperproperty.

A first milestone towards such a language was the introduction of the temporal logic HyperLTL [15], which extends LTL with quantification over traces. HyperLTL can, for instance, express observational determinism as $\forall \pi_1. \forall \pi_2. \Box(i_{\pi_1} \leftrightarrow i_{\pi_2}) \rightarrow \Box(o_{\pi_1} \leftrightarrow o_{\pi_2})$, stating that every pair of traces with identical input (modeled via atomic proposition i) also exhibits the same output (o). In the past decade, many verification methods and tools for HyperLTL have been developed (see Section 2 for an overview). HyperLTL is, however, limited in expressiveness. For example, it fails to express promptness properties which state that there must exist a bound (common across all traces of a system) up to which an event must have happened.

In this paper, we study HyperQPTL [41], a logic that – in addition to explicit trace quantification – also features propositional quantification as, e.g., found in quantified propositional temporal logic (QPTL) [42]. HyperQPTL is particularly expressive because trace and propositional quantifiers can be freely interleaved. Consequently, HyperQPTL cannot only express all ω-regular properties over multiple traces in a system but truly interweaves trace quantification and ω-regularity. For example, we can state a simple promptness property as follows:

$$\exists q. \forall \pi. \Diamond q \wedge \big(\neg q \, \mathcal{U} \, \psi(\pi)\big) \tag{1}$$

which states that there must exist an evaluation of proposition q such that **(1)** q holds at least once, and **(2)** for all traces π of the system, the desired event ψ occurs on π (denoted by $\psi(\pi)$) before the first occurrence of q. The first occurrence of q thus gives a bound up to which ψ must have happened, and – as q is quantified before the trace π – this bound is common across all traces.

This additional expressive power of HyperQPTL has been used in various different settings. Examples include causality checking in reactive systems (i.e., the question of whether some temporal property is the cause for some event, as, e.g., needed when understanding counterexamples returned by a model checker) [17]; constructing prophecies to ensure completeness during model checking [8]; showing decidability of Lewis' [39] theory of counterfactuals modulo QPTL [26]; simulating the knowledge operator and thus capturing a range of epistemic properties [41, 23]; and expressing various promptness requirements [24]. In all these applications, propositional quantification plays a crucial role, and weaker logics – such as HyperLTL – are insufficient.

However, despite HyperQPTL's importance, *practical* verification of HyperQPTL against finite-state systems was, thus far, not possible, effectively condemning all applications of HyperQPTL to be purely theoretical endeavors.

AutoHyperQ. In this paper, we present `AutoHyperQ`, an explicit-state fully-automatic model checker for HyperQPTL obtained by extending the HyperLTL model checker `AutoHyper` [10]. Our tool checks a hyperproperty by iteratively eliminating trace and propositional quantification using automata techniques – namely product-constructions with a given system (to eliminate trace quantification) and projections (to eliminate propositional quantification). To handle quantifier alternations, `AutoHyperQ` translates between non-deterministic and universal automata by utilizing automata complementations, which are outsourced to external automata tools. Importantly, `AutoHyperQ` is complete for arbitrary HyperQPTL formulas, i.e., it can verify properties with arbitrary interleaving of trace and propositional quantification.

Evaluation. To showcase `AutoHyperQ`, we verify various promptness properties on reactive systems obtained from the `SYNTCOMP` competition [35]. Our experiments demonstrate that `AutoHyperQ` can handle systems of considerable size (thousands of states) and constitutes, to the best of our knowledge, the first tool that can automatically check (a range of) promptness requirements.

QPTL to Automata. We further show that the algorithmic core of `AutoHyperQ` can be reused to translate (non-hyper) QPTL formulas into ω-automata – an important first step in most model-checking pipelines. Our experiments show that the algorithm underlying `AutoHyperQ` – when coupled with efficient automata tools such as `spot` [22] – outperforms the state-of-the-art tools for QPTL-to-automata translations.

Structure. The remainder of the paper is structured as follows: We discuss related work in Section 2, introduce HyperQPTL in Section 3, and present the theoretical extensions to the model-checking algorithm presented in [25] in order to handle propositional quantification in Section 4. We provide a brief overview of `AutoHyperQ` in Section 5. In Section 6, we demonstrate that `AutoHyperQ` can verify interesting promptness requirements, and, in Section 7, evaluate the QPTL-to-automaton translation of `AutoHyperQ`.

2 Related Work

Model Checking of Hyperproperties. Over the past decade, many verification methods and tools for HyperLTL [15] have been developed: `MCHyper` [25] can model-check alternation-free HyperLTL formulas by constructing the self-composition. Coenen et al. [18] verify $\forall^*\exists^*$ properties (i.e., properties where no existential quantifier is followed by a universal one) using user-provided strategies for the existentially quantified traces; thus reducing to the verification of an alternation-free formula. This strategy-based verification is incomplete in general but can be made complete by adding prophecies [8]. In practice, the automatic synthesis of prophecies is expensive and currently only applicable to small systems and *temporally safe* specifications [8, 6]. Hsu et al. [34] propose a bounded model-checking approach based on QBF solving. `AutoHyper` [10] checks HyperLTL formulas by employing automata-based techniques and constitutes the first *complete* model checker that can handle arbitrary HyperLTL properties.

HyperLTL has been extended in multiple dimensions to, e.g., support multi-agent systems [7, 11]; asynchronous hypeproperties [5, 13, 31, 7]; data from infinite domains [9]; and sequential information-flow policies [4]. None of these logics can express arbitrary ω-regular hyperproperties as they inherit the limited expressiveness of LTL [21].

HyperPDL-Δ [30] extends Propositional Dynamic Logic [27] with explicit trace quantification and can thus express ω-regular properties over tuples of traces. Crucially, only the temporal body that follows the quantifier prefix can express ω-regular relations, and we cannot interleave propositional and trace quantification as is possible in HyperQPTL and, e.g., needed to express promptness (cf. 1). Second-order HyperLTL [12] extends HyperLTL with quantification over arbitrary *sets* of traces and thus subsumes HyperQPTL. Different from HyperQPTL, model checking of second-order HyperLTL is highly undecidable.

HyperQPTL Model Checking. Our present tool, `AutoHyperQ`, builds on the foundations of `AutoHyper` [10] (which implements the algorithm for HyperLTL from [25]) and adds additional machinery to handle propositional quantification (cf. Section 4). Consequently, `AutoHyperQ` can handle a strict superset of the (HyperLTL) properties supported by `AutoHyper`. In particular, `AutoHyperQ` can, for the first time, check important properties such as promptness that are not expressible in HyperLTL. Conversely, on HyperQPTL properties without propositional quantification (aka. HyperLTL properties), `AutoHyperQ` shows similar performance to `AutoHyper` (see [10] for details).

Promptness. Promptness properties are ubiquitous in the study of reactive systems, and a range of specification languages that can express promptness have been proposed. Examples include PLTL [1], PromptKATL* [2], PROMPT-PNL [40], and Prompt-LTL [38]. Prompt-LTL and HyperQPTL have incomparable expressiveness [24]. While (theoretical) model-checking algorithms for some promptness logics exist [1, 38], they are – to the best of our knowledge – not implemented. `AutoHyperQ` is thus the first model-checking tool that is applicable to a range of promptness properties (cf. Section 6).

3 Preliminaries

Transition Systems. We fix a finite set of atomic propositions AP. A *transition system* is a tuple $\mathcal{T} = (S, S_0, \kappa, L)$ where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $\kappa \subseteq S \times S$ is a transition relation, and $L : S \to 2^{AP}$ is a labeling function. We assume that for every $s \in S$, there exists at least one $s' \in S$ with $(s, s') \in \kappa$. A path is an infinite sequence $s_0 s_1 s_2 \cdots \in S^\omega$, s.t., $s_0 \in S_0$, and $(s_i, s_{i+1}) \in \kappa$ for all $i \in \mathbb{N}$. The associated trace is given by $L(s_0)L(s_1)L(s_2)\cdots \in (2^{AP})^\omega$. We write $\mathit{Traces}(\mathcal{T}) \subseteq (2^{AP})^\omega$ for the set of all traces in $\mathcal{T}$. For a trace $t \in \mathit{Traces}(\mathcal{T})$ and $i \in \mathbb{N}$, we write $t(i) \in 2^{AP}$ to refer to the ith position in t.

HyperQPTL. Let $\mathcal{V}$ be a set of trace variables, and $\mathcal{P}$ be a set of propositional variables. HyperQPTL formulas are generated by the following grammar:

$$\varphi := \mathbb{Q}\pi.\, \varphi \mid \mathbb{Q}q.\, \varphi \mid \psi$$
$$\psi := a_\pi \mid q \mid \neg\psi \mid \psi \land \psi \mid \bigcirc \psi \mid \psi \,\mathcal{U}\, \psi$$

where $\mathbb{Q} \in \{\forall, \exists\}$ is a quantifier, $\pi \in \mathcal{V}$ is a trace variable, $q \in \mathcal{P}$ is a propositional variable, and $a \in AP$ is an atomic proposition. We use the usual derived boolean connectives $\lor, \to, \leftrightarrow$, boolean constants $\top, \bot$, and temporal operators *eventually* ($\Diamond \psi := \top \,\mathcal{U}\, \psi$) and *globally* ($\Box \psi := \neg \Diamond \neg \psi$).

The semantics of HyperQPTL is given with respect to a trace assignment $\Pi : \mathcal{V} \rightharpoonup (2^{AP})^\omega$ mapping trace variables to traces, and a propositional assignment $\Delta : \mathcal{P} \rightharpoonup \mathbb{B}^\omega$, where $\mathbb{B} = \{\top, \bot\}$ is the set of booleans. Intuitively, the propositional variable $q \in \mathcal{P}$ holds in step $i \in \mathbb{N}$ iff $\Delta(q)(i) = \top$. For $\pi \in \mathcal{V}$ and $t \in (2^{AP})^\omega$, we write $\Pi[\pi \mapsto t]$ for the updated trace assignment that maps π to t. For $q \in \mathcal{P}$ and $\tau \in \mathbb{B}^\omega$ we define $\Delta[q \mapsto \tau]$ analogously. Given a transition system $\mathcal{T}$, a trace assignment Π, a propositional assignment Δ, and position $i \in \mathbb{N}$, we define:

$$
\begin{array}{lll}
\Pi, \Delta, i \models a_\pi & \text{iff} & a \in \Pi(\pi)(i) \\
\Pi, \Delta, i \models q & \text{iff} & \Delta(q)(i) = \top \\
\Pi, \Delta, i \models \neg\psi & \text{iff} & \Pi, \Delta, i \not\models \psi \\
\Pi, \Delta, i \models \psi_1 \land \psi_2 & \text{iff} & \Pi, \Delta, i \models \psi_1 \text{ and } \Pi, \Delta, i \models \psi_2 \\
\Pi, \Delta, i \models \bigcirc \psi & \text{iff} & \Pi, \Delta, i + 1 \models \psi \\
\Pi, \Delta, i \models \psi_1 \,\mathcal{U}\, \psi_2 & \text{iff} & \exists j \geq i.\, \Pi, \Delta, j \models \psi_2 \text{ and } \forall i \leq k < j.\, \Pi, \Delta, k \models \psi_1
\end{array}
$$

$$
\begin{array}{lll}
\Pi, \Delta \models_{\mathcal{T}} \psi & \text{iff} & \Pi, \Delta, 0 \models \psi \\
\Pi, \Delta \models_{\mathcal{T}} \mathbb{Q}\pi.\, \varphi & \text{iff} & \mathbb{Q}t \in \mathit{Traces}(\mathcal{T}).\, \Pi[\pi \mapsto t], \Delta \models_{\mathcal{T}} \varphi \\
\Pi, \Delta \models_{\mathcal{T}} \mathbb{Q}q.\, \varphi & \text{iff} & \mathbb{Q}\tau \in \mathbb{B}^\omega.\, \Pi, \Delta[q \mapsto \tau] \models_{\mathcal{T}} \varphi
\end{array}
$$

A system $\mathcal{T}$ satisfies a HyperQPTL property φ, written $\mathcal{T} \models \varphi$, if $\emptyset, \emptyset \models_{\mathcal{T}} \varphi$, where $\emptyset$ denotes a trace or propositional assignment with an empty domain. See [24] for more details.

ω-Automata. A non-deterministic Büchi automaton (NBA) (resp. universal co-Büchi automaton (UCA)) over alphabet Σ is a tuple $\mathcal{A} = (Q, Q_0, \delta, F)$ where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and $F \subseteq Q$ is a set of accepting (resp. rejecting) states. A *run* of $\mathcal{A}$ on a word $u \in \Sigma^\omega$ is a infinite sequence $q_0 q_1 q_2 \cdots \in Q^\omega$ such that $q_0 \in Q_0$ and for every $i \in \mathbb{N}$, $q_{i+1} \in \delta(q_i, u(i))$. A word $u \in \Sigma^\omega$ is accepted by an NBA $\mathcal{A}$ if there exists *some* run on u that visits states in F *infinitely* many times. A word $u \in \Sigma^\omega$ is accepted by a UCA $\mathcal{A}$ if *all* runs on u visit states in F only *finitely* many times. Given an NBA or UCA $\mathcal{A}$, we write $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ for the set of words accepted by $\mathcal{A}$. We can translate NBAs into UCAs and vice versa with an exponential blowup using, e.g., automata complementation.

4 Model Checking for HyperQPTL

The algorithm implemented in `AutoHyperQ` builds on the automata-based model-checking algorithm proposed by Finkbeiner et al. [25] (which is limited to HyperLTL). In this section, we extend this algorithm to also handle propositional quantification in HyperQPTL. In the following, let $\mathcal{T} = (S, S_0, \kappa, L)$ be a fixed transition system.

Zipping Assignments. We zip a trace and propositional assignment into an infinite trace. Concretely, given a trace assignment $\Pi : X \to (2^{AP})^\omega$ and propositional assignment $\Delta : Y \to \mathbb{B}^\omega$ (where $X \subseteq \mathcal{V}$ and $Y \subseteq \mathcal{P}$ are the domains of both assignments), we define the trace $\mathit{zip}(\Pi, \Delta) \in (2^{(AP \times X) \cup Y})^\omega$ by, for each $i \in \mathbb{N}$, setting

$$\mathit{zip}(\Pi, \Delta)(i) := \Big\{(a, \pi) \mid \pi \in X \land a \in AP \land a \in \Pi(\pi)(i)\Big\} \cup \Big\{q \mid q \in Y \land \Delta(q)(i) = \top\Big\}.$$

That is, $(a, \pi) \in AP \times X$ holds on $\mathit{zip}(\Pi, \Delta)$ in the ith step iff a holds in the ith step on trace $\Pi(\pi)$, and $q \in Y$ holds on $\mathit{zip}(\Pi, \Delta)$ in the ith step iff $\Delta(q)(i) = \top$.

Note that *zip* defines a bijection between pairs (Π, Δ) of assignments $\Pi : X \to (2^{AP})^\omega, \Delta : Y \to \mathbb{B}^\omega$ and traces in $(2^{(AP \times X) \cup Y})^\omega$.

Definition 1. *Let φ be a HyperQPTL formula with free trace variables $X \subseteq \mathcal{V}$ and free propositional variables $Y \subseteq \mathcal{V}$. An NBA or UCA $\mathcal{A}$ over $2^{(AP \times X) \cup Y}$ is* $\mathcal{T}$-equivalent *to φ if for all trace assignments $\Pi : X \to (2^{AP})^\omega$ and propositional assignments $\Delta : Y \to \mathbb{B}^\omega$ we have $\Pi, \Delta \models_\mathcal{T} \varphi$ if and only if $\mathit{zip}(\Pi, \Delta) \in \mathcal{L}(\mathcal{A})$.*

Note that our definition of $\mathcal{T}$-equivalence differs from the one used in the context of HyperLTL model checking [10, 25] as we summarize a trace *and* propositional assignment.

Model Checking. Let $\dot\varphi$ be some fixed HyperQPTL formula that is *closed*, i.e., contains no free trace and propositional variables. Our model-checking algorithm proceeds by inductively constructing a $\mathcal{T}$-equivalent automaton $\mathcal{A}_\varphi$ (either an NBA or UCA) for each subformula φ of $\dot\varphi$. For the (quantifier-free) LTL-like body of $\dot\varphi$, we can construct this automaton via a standard LTL-to-NBA construction [25]. We then, iteratively, eliminate quantifiers by computing the product with the given system $\mathcal{T}$ (to eliminate trace quantifiers) and computing the existential or universal projection (to eliminate propositional quantifiers):

- **Case $\varphi' = \exists \pi. \varphi$:** We are given an inductively constructed automaton $\mathcal{A}_\varphi = (Q, Q_0, \delta, F)$ over $2^{(AP \times (X \uplus \{\pi\})) \cup Y}$ for some $X \subseteq \mathcal{V}$ and $Y \subseteq \mathcal{P}$ that is $\mathcal{T}$-equivalent to φ. We ensure that $\mathcal{A}_\varphi$ is an NBA (by possibly translating a UCA into an NBA) and define the NBA $\mathcal{A}_{\varphi'}$ over alphabet $2^{(AP \times X) \cup Y}$ as $\mathcal{A}_{\varphi'} := (S \times Q, S_0 \times Q_0, \delta', S \times F)$ where δ' is defined as

$$\delta'\big((s, q), \sigma\big) := \Big\{(s', q') \mid (s, s') \in \kappa \ \land \ q' \in \delta\big(q, \sigma \uplus \big\{(a, \pi) \mid a \in L(s)\big\}\big)\Big\}$$

  for $\sigma \in 2^{(AP \times X) \cup Y}$. Intuitively, $\mathcal{A}_{\varphi'}$ guesses a trace in $\mathcal{T}$ and uses this trace to fill in the propositions for trace variable π (i.e., all propositions of the form (a, π) for $a \in AP$).

- **Case $\varphi' = \forall \pi. \varphi$:** We are given an automaton $\mathcal{A}_\varphi$ over $2^{(AP \times (X \uplus \{\pi\})) \cup Y}$. We ensure that this automaton is a UCA (by possibly translating from an NBA) and define $\mathcal{A}_{\varphi'}$ as the UCA that is syntactically identical to the NBA constructed in the previous case.

- **Case $\varphi' = \exists q. \varphi$:** We are given an automaton $\mathcal{A}_\varphi = (Q, Q_0, \delta, F)$ over $2^{(AP \times X) \cup Y \uplus \{q\}}$. We ensure that $\mathcal{A}_\varphi$ is an NBA, and define the NBA $\mathcal{A}_{\varphi'} := (Q, Q_0, \delta', F)$ where

$$\delta'(q, \sigma) := \delta(q, \sigma) \cup \delta(q, \sigma \uplus \{q\}).$$

  This effectively computes the existential projection of $\mathcal{A}_\varphi$ on $2^{(AP \times X) \cup Y}$.

- **Case $\varphi' = \forall q. \varphi$:** We are given an automaton $\mathcal{A}_\varphi$ over $2^{(AP \times X) \cup Y \uplus \{q\}}$. We make sure that this automaton is a UCA and define $\mathcal{A}_{\varphi'}$ as a UCA that is syntactically identical to the NBA in the previous case, effectively computing the universal projection of $\mathcal{A}_\varphi$ on $2^{(AP \times X) \cup Y}$.

Proposition 1. *For every subformula φ, $\mathcal{A}_\varphi$ is $\mathcal{T}$-equivalent to φ.*

As the final formula $\dot{\varphi}$ is closed, we obtain a $\mathcal{T}$-equivalent automaton $\mathcal{A}_{\dot{\varphi}}$ over the singleton alphabet $2^{\emptyset}$. By definition of $\mathcal{T}$-equivalence, we have $\mathcal{T} \models \dot{\varphi}$ iff $\emptyset, \emptyset \models_{\mathcal{T}} \dot{\varphi}$ iff $\mathcal{A}_{\dot{\varphi}}$ is non-empty (which we can decide [19]).

Complexity. The computationally expensive steps in the above algorithm are the transformations of NBAs into UCAs and vice versa, which – in the worst case – increase the size of the automaton exponentially. Such a transformation is necessary whenever we encounter a quantifier *alternation* within the formula. The size of the final automaton $\mathcal{A}_{\dot{\varphi}}$ is thus m-fold exponential (i.e., a tower of m exponents) in the size of $\mathcal{T}$ and $m + 1$-fold exponential in the size of (the body of) $\dot{\varphi}$, where m is the number of quantifier alternations. These bounds are tight, as already shown by Rabe for HyperLTL [41].

5 AutoHyperQ: Tool Overview

`AutoHyperQ` is written in `F#` and implements the algorithm from Section 4 by extending the HyperLTL model checker `AutoHyper` [10]. `AutoHyperQ` reads an explicit-state transition system $\mathcal{T}$ and a HyperQPTL formula φ and determines if $\mathcal{T} \models \varphi$. As for `AutoHyper` [10], `AutoHyperQ` features a pre-processor that can translate symbolic `NuSMV` [14] systems with finite variable domains into explicit-state transition systems.

Internally, we store automata (both non-deterministic and universal) with symbolic alphabets, i.e., represent each transition as a boolean formula over $(AP \times X) \cup Y$ for $X \subseteq \mathcal{V}, Y \subseteq \mathcal{P}$. We store the transition formulas in disjunctive normal form to enable very efficient SAT-solving during the product construction and projection.

The expensive step during model checking is the translation of NBAs to UCAs and vice versa, which we realize using automata complementation. Our tool poses complementation queries in the `Hanoi` automaton format [3]. For the present evaluation, we use `spot` (version 2.11.4) [22], but any tool supporting the `Hanoi` format can be substituted easily.

`AutoHyperQ` is available at autohyper.github.io. All experiments in this paper were conducted on Macbook with an M1 Pro CPU and 32GB of memory. We execute all tools in a Docker container.

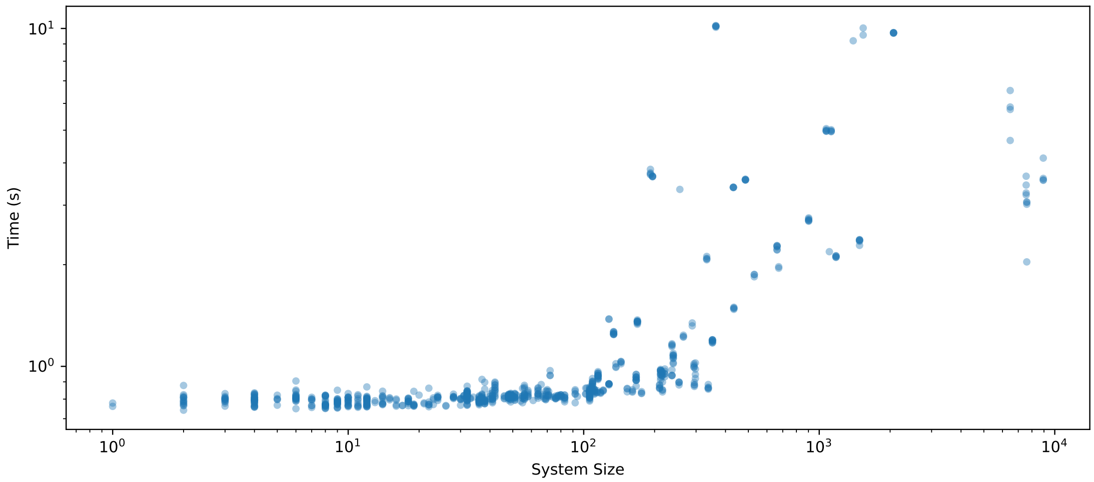


Figure 1: We use `AutoHyperQ` to check Equation (1) for each output proposition in each of the `SYNTCOMP` benchmarks. For each instance, we depict the system size and verification time (in seconds). Note that both axes are logarithmic.

6 Evaluation - Model Checking Promptness

In this section, we evaluate the model-checking capabilities of `AutoHyperQ`. As HyperQPTL is strictly more expressive than HyperLTL, `AutoHyperQ` is also applicable to existing HyperLTL benchmarks. On those instances, `AutoHyperQ` performs as fast `AutoHyper` [10], which is unsurprising as the underlying algorithm (cf. Section 4) constitutes a *proper* extension of the algorithm presented in [25] and implemented in `AutoHyper`. In our evaluation, we thus focus on properties that are *not* expressible in HyperLTL; thus truly highlighting the additional power of `AutoHyperQ`.

As we already discussed in the introduction, an important class of properties expressible in HyperQPTL are promptness requirements, i.e., properties that require a bound (common among all traces of the system) up to which some event must have happened.

SYNTCOMP Benchmarks. Promptness properties are particularly interesting in reactive systems, i.e., systems that continuously read inputs from the environment and produce outputs. To obtain an interesting set of reactive systems, we use benchmarks from annual the reactive synthesis competition (`SYNTCOMP`) [35]. `SYNTCOMP` includes a collection of LTL formulas that specify requirements for a diverse collection of reactive systems. We use existing synthesis tools (in our case `spot`'s `ltlsynt` [22]) to synthesize a strategy for each realizable LTL specification and translate them into a transition system that generates all traces of that strategy. We obtain a dataset of 317 transition systems with varying sizes.

6.1 Simple Promptness

As a first experiment, we checked – in each `SYNTCOMP` system $\mathcal{T}$ and for each output $o \in AP$ – the simple promptness property in Equation 1. That is, we check if each output is set after a fixed number of steps (common across all traces of the system). For each instance, we plot the time taken by `AutoHyperQ` against the system size in Figure 1.

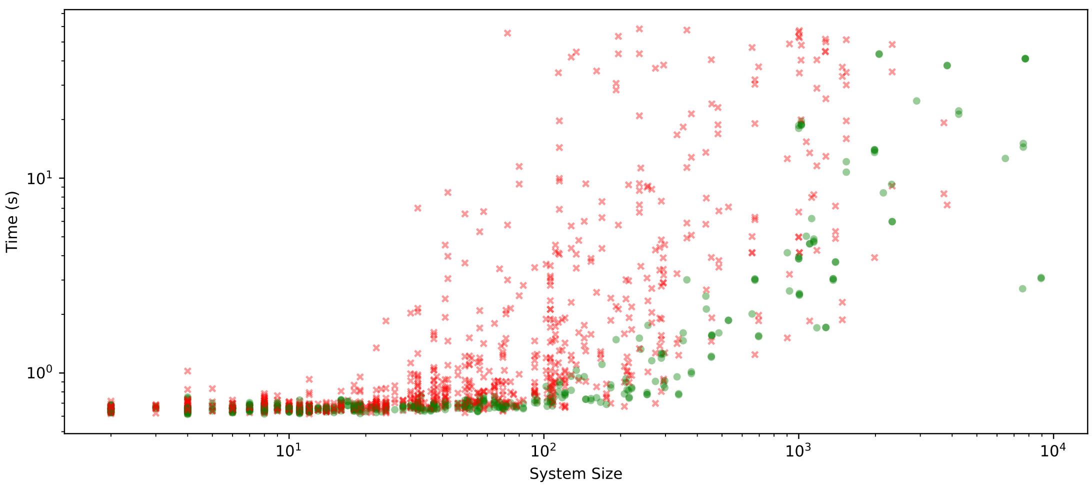


Figure 2: We use `AutoHyperQ` to verify event-specific promptness (Equation 2) in reactive systems obtained from `SYNTCOMP` benchmarks. For each instance, we depict the system size and verification time (in seconds). Each green circle marks an instance where 2 holds, and each red cross marks an instance where 2 does not hold. Note that both axes are logarithmic.

6.2 Event-Specific Promptness

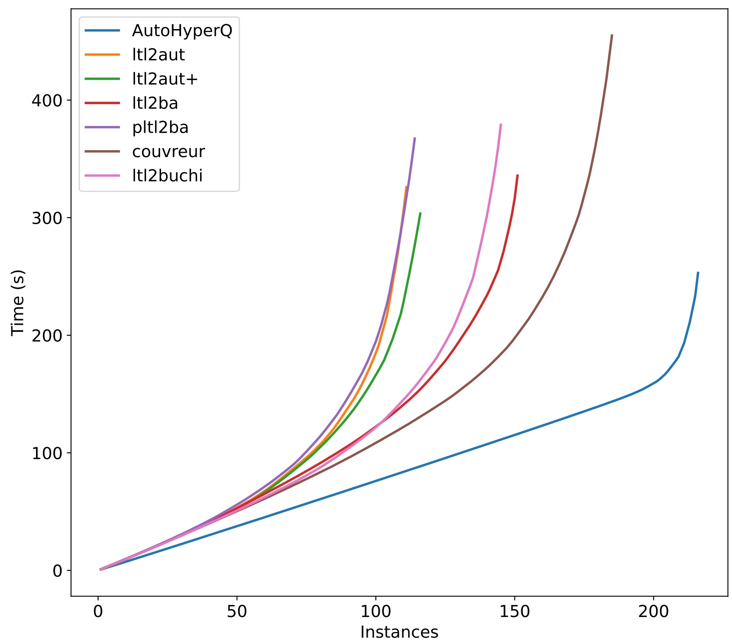
The simple promptness property used in Section 6.1 (cf. Equation (1)) demands a common bound up to which some event must have happened but does not support more general promptness requirements. For example, in many situations, one is interested whether whenever some *request*-event has occurred (e.g., "a request for some resource has been made"), some *response*-event (e.g., "a resource grant has been given") occurs within a fixed number of steps (common among all traces). To express such *event-specific* requirements in HyperQPTL, we use the alternating-color technique by Kupferman et al. [38]. Assume we are given LTL formulas ψ^{req} and ψ^{res} that describe the request-event and response-event, respectively. We construct the following HyperQPTL promptness query:

$$\begin{aligned}&\exists q.\, \forall \pi.\, \Box\Diamond q \land \Box\Diamond \neg q \land \\ &\qquad \Box\Big(\psi^{req}(\pi) \to \Big(\big(q \to (q\,\mathcal{U}(\neg q\,\mathcal{U}\,\psi^{res}(\pi)))\big) \land \big(\neg q \to (\neg q\,\mathcal{U}(q\,\mathcal{U}\,\psi^{res}(\pi)))\big)\Big)\end{aligned} \tag{2}$$

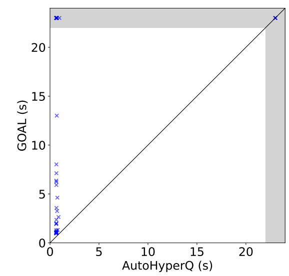
Proposition q gives the color of each step, and we require that the color alternates infinitely often ($\Box\Diamond q \land \Box\Diamond \neg q$). We then demand that whenever ψ^{req} holds on π (denoted by $\psi^{req}(\pi)$), ψ^{res} holds within two color changes (see [38, 24] for details).[1]

Experiments. For each transition system $\mathcal{T}$ generated from the `SYNTCOMP` benchmarks, we use `spot`'s `randltl` [22] to randomly generate 5 request and response events (i.e., concrete LTL formulas ψ^{req} over $\mathcal{T}$'s inputs and ψ^{res} over $\mathcal{T}$'s outputs) and use `AutoHyperQ` to (fully automatically) check the above promptness requirement. We plot the time taken by `AutoHyperQ` on each instance against the size of $\mathcal{T}$ in Figure 2.

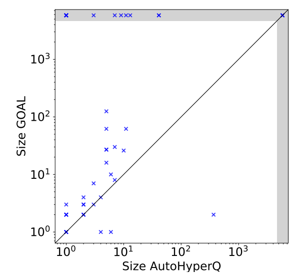
[1]The color alternation of q is common among all traces of the system, but the length of each coloring sequence might vary based on the current timestep. The above formula thus expresses a time-dependent promptness requirement, i.e., for every $n \in \mathbb{N}$ and all time points $i \leq n$ where ψ^{req} holds, there exists a common bound (which can depend on n) up to which ψ^{res} must have happened. This is similar to the treatment used in [24].



(a)



(b)



(c)

Figure 3: In Figure 3a, we compare `AutoHyperQ` with the 6 best QPTL-to-NBA algorithms implemented in `GOAL` on `SYNTCOMP` benchmarks. We set the timeout to 20 seconds. To keep the experiment reproducible in reasonable time, we restrict to the smallest 250 realizable `SYNTCOMP` benchmarks. In Figures 3b and 3c, we compare `AutoHyperQ` against `GOAL`'s `couvreur` on 50 randomly generated QPTL formulas with 2 quantifier alternations. We compare the running time (in Figure 3b) and the size of the resulting automaton (in Figure 3c). The gray area denotes a timeout (which we set to 20 seconds).

Our experiments show that `AutoHyperQ` can verify promptness in systems of considerable size, despite the fact that the promptness formula contains a quantifier alternation and thus requires an expensive automaton complementation. We stress that, to the best of our knowledge, `AutoHyperQ` is the first model-checking tool that can handle promptness requirements expressed in some general temporal logic.

7 Evaluation - QPTL Translation

An important first step in explicit-state model checking (using, e.g., `SPIN` [33]) is to transform the specification into an ω-automaton. The algorithmic core of `AutoHyperQ` – in particular, its projection functionality coupled with the translation from non-deterministic to universal automata and vice versa – can be reused to convert a QPTL formula into an ω-automaton.

In this section, we compare the QPTL-to-NBA translation based on `AutoHyperQ` against `GOAL` [43] – a library that implements multiple QPTL-to-NBA translation algorithms proposed in the literature [36, 28, 20, 37]. By focusing on QPTL-to-NBA translations (which can be seen as a *specialized* case of HyperQPTL model checking), we can compare the automata-based core of `AutoHyperQ` (cf. Section 4) with existing tools (which is not possible for HyperQPTL model checking as, prior to `AutoHyperQ`, no tool support existed). We emphasize that all expensive computations in `AutoHyperQ` are condensed into automata complementations for which we rely entirely on external tools (in our case `spot` [22]). Our experiments in this section thus do not evaluate the internal performance of `AutoHyperQ` but rather show that the underlying algorithm – when coupled with efficient automata complementation tools – works well in practice.

Evaluation on SYNTCOMP Benchmarks. To obtain a realistic set of QPTL formulas, we resort to the `SYNTCOMP` benchmarks already used in Section 6. Given an LTL formula ψ and sets I and O of input and output propositions (as specified in each `SYNTCOMP` benchmark), we construct a QPTL formula $\mathit{transform}(\psi) := \forall_{a \in I}\, a.\, \exists_{a \in O}\, a.\, \psi$. The idea is that $\mathit{transform}(\psi)$ is satisfiable iff, for any input sequence, there exists some output sequence that satisfies ψ. Note that this is a weaker requirement than realizability of ψ. If ψ is realizable, then $\mathit{transform}(\psi)$ is satisfiable. Conversely, $\mathit{transform}(\psi)$ might be satisfiable, but ψ may not be realizable (as a strategy, e.g., needs information on future inputs). We use `AutoHyperQ` and `GOAL` to translate $\mathit{transform}(\psi)$ into an NBA (over alphabet $2^{\emptyset}$) and depict the running times as a survival plot in Figure 3a. We observe that `AutoHyperQ` can translate more instances and performs faster than all existing algorithms implemented in `GOAL`.

Evaluation on Random Benchmarks. In QPTL, the number of quantifiers (or, more precisely, the number of alternations) has a direct impact on the complexity of the QPTL-to-NBA translation. We use `spot`'s `randltl` to randomly sample LTL formulas and transform them into QPTL formulas by randomly adding quantification over (some of the) propositions. We translate the resulting QPTL formulas into NBAs using both `AutoHyperQ` and `GOAL`'s `couvreur` (which performs best out of all algorithms implemented in `GOAL`). We depict the time taken by both solvers in Figure 3b and the sizes of the resulting automata in Figure 3c. We observe that `AutoHyperQ` performs faster than `GOAL` (Figure 3b) and produces (in most cases) smaller automata (Figure 3c) – an important prerequisite for efficient model checking.

8 Conclusion and Future Work

The combination (and arbitrary interleaving) of trace and propositional quantification makes HyperQPTL an attractive hyperlogic. It has already been used (in theory) in various important settings where less powerful logics, such as HyperLTL, are not sufficient. In this paper, we have presented `AutoHyperQ`, the first practical model-checking tool for HyperQPTL, making the existing (thus far, only theoretical) use cases of HyperQPTL [17, 8, 41, 24] applicable in practice. Our early experiments show that verification of important properties such as promptness is possible in realistic reactive systems.

Having access to a fully-automatic HyperQPTL model checker opens numerous interesting avenues for future work. As an immediate next step, we plan to use `AutoHyperQ` to automatically check causes in reactive systems using the theory developed by Coenen et al. [17] based on Halpern and Pearl's [32] actual causality. Such use cases demonstrate how our HyperQPTL model checker can provide elegant algorithmic solutions to seemingly unrelated problems; in this case, the explanation of counterexamples using techniques from causality analysis.

Acknowledgments. This work was partially supported by the DFG in project 389792660, and by the ERC Grant HYPER (No. 101055412). R. Beutner carried out this work as a member of the Saarbrücken Graduate School of Computer Science.

References

[1] Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron A. Peled. Parametric temporal logic for "model measuring". *ACM Trans. Comput. Log.*, 2(3), 2001.

[2] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. Prompt alternating-time epistemic logics. In *International Conference on Principles of Knowledge Representation and Reasoning, KR 2016*. AAAI Press, 2016.

[3] Tomás Babiak, Frantisek Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Kretínský, David Müller, David Parker, and Jan Strejcek. The Hanoi omega-automata format. In *International Conference on Computer Aided Verification, CAV 2015*, volume 9206 of *Lecture Notes in Computer Science*. Springer, 2015.

[4] Ezio Bartocci, Thomas Ferrère, Thomas A. Henzinger, Dejan Nickovic, and Ana Oliveira da Costa. Flavors of sequential information flow. In *International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2022*, volume 13182 of *Lecture Notes in Computer Science*. Springer, 2022.

[5] Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A temporal logic for asynchronous hyperproperties. In *International Conference on Computer Aided Verification, CAV 2021*, volume 12759 of *Lecture Notes in Computer Science*. Springer, 2021.

[6] Raven Beutner, David Carral, Bernd Finkbeiner, Jana Hofmann, and Markus Krötzsch. Deciding hyperproperties combined with functional specifications. In *ACM/IEEE Symposium on Logic in Computer Science, LICS 2022*. ACM, 2022.

[7] Raven Beutner and Bernd Finkbeiner. A temporal logic for strategic hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2021*, volume 203 of *LIPIcs*. Schloss Dagstuhl, 2021.

[8] Raven Beutner and Bernd Finkbeiner. Prophecy variables for hyperproperty verification. In *IEEE Computer Security Foundations Symposium, CSF 2022*. IEEE, 2022.

[9] Raven Beutner and Bernd Finkbeiner. Software verification of hyperproperties beyond k-safety. In *International Conference on Computer Aided Verification, CAV 2022*, volume 13371 of *Lecture Notes in Computer Science*. Springer, 2022.

[10] Raven Beutner and Bernd Finkbeiner. AutoHyper: Explicit-state model checking for HyperLTL. In *International Conferenc on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2023*, volume 13993 of *Lecture Notes in Computer Science*. Springer, 2023.

[11] Raven Beutner and Bernd Finkbeiner. HyperATL*: A logic for hyperproperties in multi-agent systems. *Log. Methods Comput. Sci.*, 2023.

[12] Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Second-order hyperproperties. In *International Conference on Computer Aided Verification, CAV 2023*, Lecture Notes in Computer Science. Springer, 2023.

[13] Laura Bozzelli, Adriano Peron, and César Sánchez. Asynchronous extensions of HyperLTL. In *ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*. IEEE, 2021.

[14] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *International Conference on Computer Aided Verification, CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*. Springer, 2002.

[15] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *International Conference on Principles of Security and Trust, POST 2014*, volume 8414 of *Lecture Notes in Computer Science*. Springer, 2014.

[16] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *IEEE Computer Security Foundations Symposium, CSF 2008*. IEEE, 2008.

[17] Norine Coenen, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Niklas Metzger, and Julian

Siber. Temporal causality in reactive systems. In *International Symposium on Automated Technology for Verification and Analysis, ATVA 2022*, volume 13505 of *Lecture Notes in Computer Science*. Springer, 2022.

[18] Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In *International Conference on Computer Aided Verification, CAV 2019*, volume 11561 of *Lecture Notes in Computer Science*. Springer, 2019.

[19] Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper, and Mihalis Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods Syst. Des.*, 1(2), 1992.

[20] Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. In *World Congress on Formal Methods in the Development of Computing Systems, FM 1999*, volume 1708 of *Lecture Notes in Computer Science*. Springer, 1999.

[21] Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*. Amsterdam University Press, 2008.

[22] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From spot 2.0 to spot 2.10: What's new? In *International Conference on Computer Aided Verification, CAV 2022*, volume 13372 of *Lecture Notes in Computer Science*. Springer, 2022.

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

[24] Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing ω-regular hyperproperties. In *International Conference on Computer Aided Verification, CAV 2020*, volume 12225 of *Lecture Notes in Computer Science*. Springer, 2020.

[25] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *International Conference on Computer Aided Verification, CAV 2015*, volume 9206 of *Lecture Notes in Computer Science*. Springer, 2015.

[26] Bernd Finkbeiner and Julian Siber. Counterfactuals modulo temporal logics. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2023*, EPiC Series in Computing. EasyChair, 2023.

[27] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2), 1979.

[28] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *International Symposium on Protocol Specification, Testing and Verification, IFIP 1995*, volume 38, 1995.

[29] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy, SP 1982*. IEEE, 1982.

[30] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Propositional dynamic logic for hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2020*, volume 171 of *LIPIcs*. Schloss Dagstuhl, 2020.

[31] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL), 2021.

[32] Joseph Y Halpern and Judea Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *The British journal for the philosophy of science*, 2005.

[33] Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5), 1997.

[34] Tzu-Han Hsu, César Sánchez, and Borzoo Bonakdarpour. Bounded model checking for hyperproperties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2021*, volume 12651 of *Lecture Notes in Computer Science*. Springer, 2021.

[35] Swen Jacobs, Guillermo A. Pérez, Remco Abraham, Véronique Bruyère, Michaël Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville,

Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara J. Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaëtan Staquet, Clément Tamines, Leander Tentrup, and Adam Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021. *CoRR*, abs/2206.00251, 2022.

[36] Yonit Kesten, Zohar Manna, Hugh McGuire, and Amir Pnueli. A decision algorithm for full propositional temporal logic. In *International Conference on Computer Aided Verification, CAV 1993*, volume 697 of *Lecture Notes in Computer Science*. Springer, 1993.

[37] Yonit Kesten and Amir Pnueli. Complete proof system for QPTL. *J. Log. Comput.*, 12(5), 2002.

[38] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods Syst. Des.*, 34(2), 2009.

[39] David Kellogg Lewis. *Counterfactuals*. Cambridge, MA, USA: Blackwell, 1973.

[40] Dario Della Monica, Angelo Montanari, Aniello Murano, and Pietro Sala. Prompt interval temporal logic. In *European Conference on Logics in Artificial Intelligence, JELIA 2016*, volume 10021 of *Lecture Notes in Computer Science*, 2016.

[41] Markus N. Rabe. *A temporal logic approach to information-flow control*. PhD thesis, Saarland University, 2016.

[42] Aravinda Prasad Sistla. *Theoretical issues in the design and verification of distributed systems*. PhD thesis, Harvard University, 1983.

[43] Ming-Hsien Tsai, Yih-Kuen Tsay, and Yu-Shiang Hwang. GOAL for games, omega-automata, and logics. In *International Conference on Computer Aided Verification, CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*. Springer, 2013.

[44] Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *Computer Security Foundations Workshop, CSFW 2003*. IEEE, 2003.