Propositional Dynamic Logic for Hyperproperties

Jens Oliver Gutsfeld
Institut für Informatik, Westfälische Wilhelms-Universität Münster, Germany
jens.gutsfeld@uni-muenster.de

Markus Müller-Olm
Institut für Informatik, Westfälische Wilhelms-Universität Münster, Germany
markus.mueller-olm@uni-muenster.de

Christoph Ohrem
Institut für Informatik, Westfälische Wilhelms-Universität Münster, Germany
christoph.ohrem@uni-muenster.de

Abstract

Information security properties of reactive systems like non-interference often require relating different executions of the system to each other and following them simultaneously. Such *hyperproperties* can also be useful in other contexts, e.g., when analysing properties of distributed systems like linearizability. Since common logics like LTL, CTL, or the modal μ-calculus cannot express hyperproperties, the hyperlogics HyperLTL and HyperCTL* were developed to cure this defect. However, these logics are not able to express arbitrary ω-regular properties. In this paper, we introduce HyperPDL-Δ, an adaptation of the Propositional Dynamic Logic of Fischer and Ladner for hyperproperties, in order to remove this limitation. Using an elegant automata-theoretic framework, we show that HyperPDL-Δ model checking is asymptotically not more expensive than HyperCTL* model checking, despite its vastly increased expressive power. We further investigate fragments of HyperPDL-Δ with regard to satisfiability checking.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking; Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects

Keywords and phrases Hyperlogics, Hyperproperties, Model Checking, Automata

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.50

Related Version An extended version of this paper is available at `https://arxiv.org/abs/1910.10546`.

Funding This work was partially funded by DFG project Model-Checking of Navigation Logics (MoNaLog) (MU 1508/3).

Acknowledgements We thank the reviewers for their helpful comments.

1 Introduction

Temporal logics like LTL, CTL or CTL* have been used successfully in verification. These logics consider paths of a structure (in linear time logics) or paths and their possible extensions (in branching time logics). Notably, since they cannot refer to multiple paths at once, they cannot express *hyperproperties* that relate multiple paths to each other. Examples of hyperproperties include information security properties like non-interference [6] or properties of distributed systems like linearizablity [2]. In order to develop a dedicated logic for these properties, Clarkson et. al. [5, 12] introduced HyperLTL and HyperCTL*, which extend LTL and CTL* by path variables. However, just like LTL and CTL [22], they cannot express arbitrary ω-regular properties of traces [20], a desirable property of specification logics [1, 16]. Logics like Propositional Dynamic Logic (PDL) [13, 18] and Linear Dynamic Logic (LDL) [8, 10] are able to do so for single traces. As we seek to extend this ability to

hyperproperties, we introduce a variant of PDL for hyperproperties called HyperPDL-Δ in this paper. HyperPDL-Δ properly extends logics like HyperLTL, HyperCTL*, LDL and PDL-Δ and can express all ω-regular properties over hypertraces in a handy formalism based on regular expressions over programs. We develop a model checking algorithm for HyperPDL-Δ inspired by one for HyperCTL* [12] and show that the model checking problem for HyperPDL-Δ is decidable at no higher asymptotic cost than the corresponding problem for HyperCTL* despite the vastly increased expressive power. Our algorithm non-trivially differs from the algorithm in [12] in two ways: first of all, it handles more general, regular modalities that subsume the modalities of HyperCTL* and require different constructions. Then, we use a different notion of alternation depth (called *criticality*) which conservatively extends their notion, but requires handling structurally different operators and regular expressions. We also show that for fragments of HyperPDL-Δ similar to the fragments of HyperLTL considered in [11], the satisfiability problem is decidable.

This paper is structured as follows: Section 2 introduces Kripke Transition Systems and (alternating) Büchi automata. In Section 3, we define our new logic HyperPDL-Δ and describe properties expressible in it. Afterwards, in Section 4, we outline a model checking algorithm for HyperPDL-Δ and show that it is asymptotically optimal by providing a precise complexity classification. In Section 5, we consider fragments of HyperPDL-Δ for which the satisfiability problem is decidable. Then, in Section 6, we show that HyperPDL-Δ can express all ω-regular properties over sets of traces and compare it to existing hyperlogics with regard to expressivity. Finally, in Section 7, we provide a summary of this paper. The appendices contain two constructions only sketched in the main body of this paper. Due to lack of space, some proofs can be found in the appendix of the extended version only.

Related Work. Hyperproperties were systematically analysed in [6] and dedicated temporal logics for hyperproperties, HyperLTL and HyperCTL*, were introduced in [5]. An overview of temporal hyperlogics and discussion of their expressive power can be found in [7]. Efficient model checking algorithms for these logics were introduced in [12] by Finkbeiner et al. and our model checking algorithm for HyperPDL-Δ builds on ideas from their construction. Our satisfiability algorithm, on the other hand, is inspired by the corresponding algorithm for HyperLTL [11]. Recently, Bonakdapour et. al. proposed *regular hyperlanguages* and a corresponding automata model [3]. In contrast to our work, their model is concerned with hyperproperties over finite instead of infinite words and does not concern branching-time properties. Moreover, they study automata-theoretic questions while our focus here is on verification of hyperproperties specified by logical means. A different line of research for properties involving multiple traces at once is given by *epistemic temporal logics* [14]. An attempt to unify epistemic temporal logics and hyperlogics is given by Bozzelli et. al in [4] via a variant of HyperCTL* with past modalities. PDL was originally introduced in [13] by Fischer and Ladner and has been extended in multiple ways [18]. There are several attempts to extend temporal logic by regular properties: a variant of PDL for linear time properties of finite traces, LDL_f, was introduced in [8] and was extended upon for the infinite setting in [10] by introducing parametrised operators. Other regular extensions of temporal logics were studied e.g. by Wolper [22] or by Kupferman et. al [16]. However, all these extensions do not concern hyperproperties.

2 Preliminaries

Let AP be a finite set of atomic propositions and Σ a finite set of atomic programs. A *Kripke Transition System* (KTS) is a tuple $\mathcal{T} = (S, s_0, \{\delta_\sigma \mid \sigma \in \Sigma\}, L)$ where S is a finite set of states, $s_0 \in S$ is an initial state, $\delta_\sigma \subseteq S \times S$ is a transition relation for each $\sigma \in \Sigma$ and

$L : S \to 2^{AP}$ is a labeling function. We assume that there are no states without outgoing edges, that is for each $s \in S$, there is $s' \in S$ with $(s, s') \in \delta_\sigma$ for some $\sigma \in \Sigma$. A KTS is a combination of a Kripke Structure and a labeled transition system (LTS) where a Kripke Structure K is a KTS for $|\Sigma| = 1$ and an LTS T is a KTS with the labelling function $s \mapsto \emptyset$ for all $s \in S$. A path in a KTS $\mathcal{T}$ is an infinite alternating sequence $s_0\sigma_0 s_1\sigma_1 ... \in (S\Sigma)^\omega$ where s_0 is the initial state of $\mathcal{T}$ and $(s_i, s_{i+1}) \in \delta_{\sigma_i}$ for all $i \geq 0$. We denote by $Paths(\mathcal{T}, s)$ the set of paths in $\mathcal{T}$ starting in s and by $Paths^*(\mathcal{T}, s)$ the set of corresponding path suffixes $\{p[i, \infty] \mid p \in Paths(\mathcal{T}, s), i \in \mathbb{N}_0\}$ where $p[i, \infty]$ is the path suffix of p starting at index i. A trace is an alternating infinite sequence $t \in (2^{AP}\Sigma)^\omega$. For a path $\pi = s_0\sigma_0 s_1\sigma_1 ...$, the induced trace is given by $L(s_0)\sigma_0 L(s_1)\sigma_1 ....$ For a KTS $\mathcal{T}$ and a state $s \in S$, we write $Traces(\mathcal{T}, s)$ to denote the traces induced by paths of $\mathcal{T}$ starting in s.

An alternating Büchi automaton (ABA) is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \rho, F)$ where Q is a finite set of states, $q_0 \in Q$ is an initial state, Σ is a finite alphabet, $\rho : Q \times \Sigma \to \mathbb{B}^+(Q)$ is a transition function mapping each pair of state and input symbol to a non-empty positive boolean combination of successor states and $F \subseteq Q$ is a set of accepting states. We assume that every ABA has two distinct states $true \in F$ and $false \in Q \setminus F$ with $\rho(true, \sigma) = true$ and $\rho(false, \sigma) = false$ for all $\sigma \in \Sigma$. Thus, all maximal paths in an ABA are infinite. A tree T is a subset of $\mathbb{N}^*$ such that for every node $t \in \mathbb{N}^*$ and every positive integer $n \in \mathbb{N}$: $t \cdot n \in T$ implies (i) $t \in T$ (we then call $t \cdot n$ a child of t), and (ii) for every $0 < m < n$, $t \cdot m \in T$. We assume every node has at least one child. A path in a tree T is a sequence of nodes $t_0 t_1 ...$ such that $t_0 = \varepsilon$ and t_{i+1} is a child of t_i for all $i \in \mathbb{N}_0$. A run of an ABA $\mathcal{A}$ on an infinite word $w \in \Sigma^\omega$ is defined as a Q-labeled tree (T, r) where $r : T \to Q$ is a labelling function such that $r(\varepsilon) = q_0$ and for every node $t \in T$ with children $t_1, ..., t_k$, we have $1 \leq k \leq |Q|$ and the valuation assigning true to the states $r(t_1), ..., r(t_k)$ and false to all other states satisfies $\rho(r(t), w(|t|))$. A run (T, r) is an accepting run iff for every path $t_1 t_2 ...$ in T, there are infinitely many i with $r(t_i) \in F$. A word w is accepted by $\mathcal{A}$ iff there is an accepting run of $\mathcal{A}$ on w. The set of infinite words accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$. A nondeterministic Büchi automaton is an ABA in which every transition rule consists only of disjunctions.

We will make use of two well-known theorems about ABA:

▶ **Proposition 1** ([19]). *For every ABA $\mathcal{A}$ with n states, there is a nondeterministic Büchi automaton $MH(\mathcal{A})$ with $2^{\mathcal{O}(n)}$ states that accepts the same language.*

▶ **Proposition 2** ([19, 17]). *For every ABA $\mathcal{A}$ with n states, there is an ABA $\overline{\mathcal{A}}$ with $\mathcal{O}(n^2)$ states that accepts the complement language, i.e., $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}(\mathcal{A})}$.*

3 Propositional Dynamic Logic for Hyperproperties

In this section, we define our new logic, HyperPDL-Δ. Structurally, it consists of formulas φ referring to state labels and programs α referring to transition labels. We use the syntax of HyperCTL* as a basis for formulas but replace the modalities $\bigcirc, \mathcal{U}$ and $\mathcal{R}$ by PDL-like expressions $\langle\alpha\rangle\varphi, [\alpha]\varphi$ and $\Delta\alpha$ constructed from programs. These programs α are regular expressions over tuples of atomic programs τ capturing the transition behaviour on the considered paths. Additionally, we allow test-operators $\varphi?$ in α in order to enable constructions like conditional branching.

▶ **Definition 3** (Syntax of HyperPDL-Δ). *Let $N = \{\epsilon, \pi_1, \pi_2 \ldots\}$ be a set of path variables with a special path variable $\epsilon \in N$. A formula φ is a HyperPDL-Δ formula if it is built from the following context-free grammar:*

$$\varphi ::= \exists \pi.\varphi \mid \forall \pi.\varphi \mid a_\pi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid \Delta\alpha$$
$$\alpha ::= \tau \mid \varepsilon \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha^* \mid \varphi?$$

where $\pi \in N \setminus \{\epsilon\}$, $a \in AP$ and $\tau \in (\Sigma \cup \{\cdot\})^n$ for the number $n > 0$ of path quantifiers that α is in scope of. The constructs $\langle\alpha\rangle\varphi, [\alpha]\varphi$ and $\Delta\alpha$ are only allowed in scope of at least one quantifier.

We call a HyperPDL-Δ formula φ *closed* iff all occurrences of path variables π in φ (as indices of atomic propositions or in atomic programs) are bound by a quantifier. In this paper, we only consider closed formulas φ. In programs α, each component of tuples $\tau \in (\Sigma \cup \{\cdot\})^n$ corresponds to one of the path variables bound by a quantifier α is in scope of. We assume that a quantifier that is in scope of $i-1$ other quantifiers quantifies path variable π_i.

Connectives inherited from HyperCTL* are interpreted analogously: quantifiers $\exists$ and $\forall$ should be read as "along some path" and "along all paths". Using different path variables π enables us to refer to multiple paths at the same time. For example, with $\forall\pi_1.\exists\pi_2.\exists\pi_3.\varphi$, one can express that for all paths π_1, there are paths π_2 and π_3 such that φ holds along these three paths. Boolean connectives are defined in the usual way. Atomic propositions $a \in AP$ express information about a state and have to be indexed by a path variable π to express on which path we expect a to hold.

Intuitively, a program α explores all paths it is in scope of synchronously and thus allows us to pose a regular constraint on the sequence of atomic programs visited on path prefixes of equal length. In addition, properties of infinite suffixes can be required at certain points during the exploration using tests $\varphi?$. In formulas φ, atomic propositions a_π on single paths suffice to relate behavior on different paths because boolean connectives are available. Referring to occurences of atomic programs on single paths in programs α in a similar way, however, would reduce expressivity since neither negation nor conjunction are present in α. As a remedy, we use tuples $\tau \in (\Sigma \cup \{\cdot\})^n$ to refer to atomic programs on paths $\pi_1, ..., \pi_n$, where σ in position i means that on path π_i, we expect a use of σ to reach the next state. A wildcard symbol $\cdot$ expresses that any atomic program is allowed on the corresponding path.

The constructs using α can be interpreted as follows: the diamond operator $\langle\alpha\rangle\varphi$ means that α matches a prefix of the current paths after which φ holds. The box operator $[\alpha]\varphi$ is the dual of $\langle\alpha\rangle\varphi$, meaning φ holds at the end of all prefixes matching α. The last construct $\Delta\alpha$ is of a different kind and expresses ω-regular rather than regular properties. It says that α occurs repeatedly, i.e., the currently quantified paths can be divided into infinitely many segments matching α. $\Delta\alpha$ expresses a variant of a Büchi condition. Instead of moving from accepting states to accepting states in a Büchi automaton, one moves from initial states to accepting states repeatedly.

Using our logic, common hyperproperties can be expressed easily and intuitively. Let us consider two examples: the first, *observational determinism* [6], states that if two executions of a system receive equal low security inputs, they are indistinguishable for a low security observer all the time. It can be expressed by $\forall\pi_1.\forall\pi_2.(\bigwedge_{a\in L}(a_{\pi_1} \leftrightarrow a_{\pi_2})) \rightarrow [\bullet^*]\bigwedge_{a\in L}(a_{\pi_1} \leftrightarrow a_{\pi_2})$, where low security observable behaviour is modelled by the atomic propositions in L. Here, we use the common boolean abbreviations $\rightarrow$ for implication and $\leftrightarrow$ for equivalence as

well as $\bullet$ as an abbreviation the program $\tau = (\cdot, ..., \cdot)$. The second example, *generalized noninterference* [6], states that high security injections do not interfere with low security observable behaviour. It can be expressed by stating that for all pairs of executions π_1, π_2 there is a third execution π_3 agreeing with π_1 on high security injections and is indistinguishable from π_2 for a low security observer: $\forall\pi_1.\forall\pi_2.\exists\pi_3.[\bullet^*]\bigwedge_{a\in H}(a_{\pi_1} \leftrightarrow a_{\pi_3}) \wedge \bigwedge_{a\in L}(a_{\pi_2} \leftrightarrow a_{\pi_3})$. Since these hyperproperties can already be expressed in HyperLTL [5], which is subsumed by our logic by encoding $\varphi_1 \mathcal{U} \varphi_2$ formulas with $\langle(\varphi_1? \cdot \bullet)^*\rangle\varphi_2$, it is no surprise, that they can be expressed in HyperPDL-Δ as well.

The ability to express arbitrary ω-regular properties however, allows a much more fine-grained analysis of a system than HyperLTL. For example, by replacing the program $\bullet^*$ with $(\bullet \cdot \bullet)^* \cdot \bullet \cdot (\sigma_1, \sigma_1)$ in the observational determinism formula, we can restrict the requirement on low security outputs to only apply for every other state with the additional constraint that some specific program σ_1 was last executed in both π_1 and π_2. As was argued in [1] for linear time specification logics, the ability to express ω-regular properties can indeed become a practical issue when in an assume-guarantee setting detailed information about the behaviour of the context has to be taken into account in order to prove properties of interest. This indicates that availability of ω-regular properties is not just a theoretical issue in specification logics.

While there are hyperlogics with the ability to express all ω-regular languages [7], these lack properties desirable for verification: HyperQPTL obtains the ability to express ω-regular properties from the addition of propositional quantification, which complicates its use for specification purposes since the user has to keep track of heterogeneous types of quantifiers. The simple property that all executions π_1 and π_2 agree on propositions from a set P in every other state for example is expressed by the HyperQPTL formula $\forall\pi_1.\forall\pi_2.\exists t : t \wedge \square(\bigcirc t \leftrightarrow \neg t) \wedge \square(t \rightarrow \bigwedge_{a\in P} a_{\pi_1} \leftrightarrow a_{\pi_2})$ [15]. The specification of this property in HyperPDL-Δ is much more direct: $\forall\pi_1.\forall\pi_2.[(\bullet \cdot \bullet)^*]\bigwedge_{a\in P} a_{\pi_1} \leftrightarrow a_{\pi_2}$. This example also illustrates another problem: due to the additional quantifier alternation, the only known model checking algorithm for HyperQPTL [20] is exponentially more expensive than that of HyperPDL-Δ for such formulas. S1S[E] on the other hand, while being even more expressive than HyperPDL-Δ, has an undecidable model checking problem [7].

Before formally defining our logic's semantics, we introduce some notation. We call a partial function $\Pi : N \rightsquigarrow \mathit{Paths}^*(\mathcal{T}, s_0)$ with $\mathit{dom}(\Pi) = \{\epsilon, \pi_1, ..., \pi_n\}$ a path assignment and denote by PA the set of all path assignments. In the context of a subformula φ, $\mathit{dom}(\Pi)$ contains exactly the variables it is in scope of as well as ϵ. The path variable ϵ refers to the most recently *assigned* path in a path assignment and is used to ensure that paths induced by quantifiers branch from the most recently quantified path. We use $\{\epsilon \rightarrow p\}$ for a path p to denote the path assignment Π with $\mathit{dom}(\Pi) = \{\epsilon\}$ and $\Pi(\epsilon) = p$. We introduce $\Pi[i, \infty]$ as a notation to manipulate path assignments Π such that $\Pi[i, \infty](\pi) = \Pi(\pi)[i, \infty]$ holds for all $\pi \in \mathit{dom}(\Pi)$. Also, $\Pi[\pi_i \rightarrow p]$ is a notation for a path assignment Π' where $\Pi'(\pi_i) = p$ and $\Pi'(\pi_j) = \Pi(\pi_j)$ for all $j \neq i$. As a convention, we do not count ϵ when determining $|\mathit{dom}(\Pi)|$. For a tuple $\tau = (\sigma_1, ..., \sigma_n)$, we write $\tau|_i$ to refer to σ_i.

We write $\Pi \models_{\mathcal{T}} \varphi$ to denote that in the context of a KTS $\mathcal{T}$, a path assignment Π fulfills a formula φ. We also write $(\Pi, i, k) \in R(\alpha)$ for a path assignment Π and two even numbers $i \leq k$ to denote that the transition labels on the paths in Π between i and k match α. Formally, we define these two relations as follows:

▶ **Definition 4** (Semantics of HyperPDL-Δ). *Given a KTS* $\mathcal{T} = (S, s_0, \{\delta_\sigma \mid \sigma \in \Sigma\}, L)$*, we inductively define both satisfaction of formulas* φ *and programs* α *on path assignments* Π*.*

$$
\begin{array}{ll}
\Pi \models_{\mathcal{T}} \exists \pi.\varphi & \text{iff there is } p \in \mathit{Paths}(\mathcal{T}, \Pi(\epsilon)(0)) \text{ s.t. } \Pi[\pi \to p, \epsilon \to p] \models_{\mathcal{T}} \varphi \\
\Pi \models_{\mathcal{T}} \forall \pi.\varphi & \text{iff for all } p \in \mathit{Paths}(\mathcal{T}, \Pi(\epsilon)(0)) : \Pi[\pi \to p, \epsilon \to p] \models_{\mathcal{T}} \varphi \\
\Pi \models_{\mathcal{T}} a_\pi & \text{iff } a \in L(\Pi(\pi)(0)) \\
\Pi \models_{\mathcal{T}} \neg\varphi & \text{iff } \Pi \not\models_{\mathcal{T}} \varphi \\
\Pi \models_{\mathcal{T}} \varphi_1 \wedge \varphi_2 & \text{iff } \Pi \models_{\mathcal{T}} \varphi_1 \text{ and } \Pi \models_{\mathcal{T}} \varphi_2 \\
\Pi \models_{\mathcal{T}} \varphi_1 \vee \varphi_2 & \text{iff } \Pi \models_{\mathcal{T}} \varphi_1 \text{ or } \Pi \models_{\mathcal{T}} \varphi_2 \\
\Pi \models_{\mathcal{T}} \langle\alpha\rangle\varphi & \text{iff there is } i \geq 0 \text{ s.t. } \Pi[i, \infty] \models_{\mathcal{T}} \varphi \text{ and } (\Pi, 0, i) \in R(\alpha) \\
\Pi \models_{\mathcal{T}} [\alpha]\varphi & \text{iff for all } i \geq 0 \text{ with } (\Pi, 0, i) \in R(\alpha) : \Pi[i, \infty] \models_{\mathcal{T}} \varphi \\
\Pi \models_{\mathcal{T}} \Delta\alpha & \text{iff there are } 0 = k_1 \leq k_2 \leq \ldots \text{ s.t. for all } i \geq 1 : \\
 & \quad (\Pi, k_i, k_{i+1}) \in R(\alpha)
\end{array}
$$

$$
\begin{array}{ll}
(\Pi, i, k) \in R(\tau) & \text{iff } k = i + 2 \text{ and for all } 1 \leq l \leq |\mathit{dom}(\Pi)| : \\
 & \quad \tau|_l = \cdot \text{ or } \Pi(\pi_l)(i+1) = \tau|_l \\
(\Pi, i, k) \in R(\varepsilon) & \text{iff } i = k \\
(\Pi, i, k) \in R(\alpha_1 + \alpha_2) & \text{iff } (\Pi, i, k) \in R(\alpha_1) \text{ or } (\Pi, i, k) \in R(\alpha_2) \\
(\Pi, i, k) \in R(\alpha_1 \cdot \alpha_2) & \text{iff there is } j \text{ s.t. } i \leq j \leq k, (\Pi, i, j) \in R(\alpha_1) \text{ and} \\
 & \quad (\Pi, j, k) \in R(\alpha_2) \\
(\Pi, i, k) \in R(\alpha^*) & \text{iff there are } l \geq 0, i = j_0 \leq j_1 \leq \ldots \leq j_l = k \text{ s.t.} \\
 & \quad \text{for all } 0 \leq m < l : (\Pi, j_m, j_{m+1}) \in R(\alpha) \\
(\Pi, i, k) \in R(\varphi?) & \text{iff } i = k \text{ and } \Pi[i, \infty] \models_{\mathcal{T}} \varphi
\end{array}
$$

A KTS $\mathcal{T}$ satisfies a formula φ, denoted by $\mathcal{T} \models \varphi$, iff $\{\epsilon \to p\} \models_{\mathcal{T}} \varphi$ holds for an arbitrary $p \in \mathit{Paths}(\mathcal{T}, s_0)$. Note that the choice of p ensures that the outermost quantified paths in a formula always branch from the starting state of $\mathcal{K}$, i.e. s_0.

4 Model Checking HyperPDL-Δ

In order to tackle the model checking problem for HyperPDL-Δ, that is to check whether $\mathcal{T} \models \varphi$ holds for arbitrarily given KTS $\mathcal{T}$ and closed HyperPDL-Δ formulas φ, we develop a new algorithm inspired by the HyperCTL* model checking algorithm from [12]. A crucial idea is to represent a path assignment Π with $\mathit{dom}(\Pi) = \{\epsilon, \pi_1, ..., \pi_n\}$ by an ω-word over $(S^n \times \Sigma^n)$. Formally, we define a translation function $\nu : PA \to \bigcup_{n \in \mathbb{N}_0} (S^n \times \Sigma^n)^\omega$ such that a path assignment Π with $\Pi(\pi_i) = s_i^0 \sigma_i^0 s_i^1 \sigma_i^1 ...$ is mapped to $\nu(\Pi) = ((s_0^0, ..., s_n^0), (\sigma_0^0, ..., \sigma_n^0))((s_0^1, ..., s_n^1), (\sigma_0^1, ..., \sigma_n^1))... \in (S^n \times \Sigma^n)^\omega$ for $n = |\mathit{dom}(\Pi)|$. Note that ϵ need not be encoded separately in $\nu(\Pi)$ since $\Pi(\pi_n) = \Pi(\epsilon)$ always holds. Similar to the notation for τ used before, we use the notation s to refer to tuples $(s_1, ..., s_n)$ and write $s|_i$ to refer to s_i. Then, given a formula φ and a KTS $\mathcal{T}$, we construct an ABA $\mathcal{A}_\varphi$ recognising $\nu(\Pi)$ iff $\Pi \models_{\mathcal{T}} \varphi$ holds.

First, we transform all formulas into a variation of negation normal form, where only existential quantifiers are allowed and negation can only occur in front of existential quantifiers, atomic propositions or Δs. This form differs from conventional NNF in that negated existential instead of universal quantifiers are used, because they can be handled more efficiently in our

setup. The transformation is done by driving all negations in the formula inwards using De Morgan's laws and the duality $\neg\langle\alpha\rangle\varphi \equiv [\alpha]\neg\varphi$ while also replacing universal quantifiers $\forall$ with $\neg\exists\neg$ and cancelling double negations successively. For example, the formula $\forall\pi.[\alpha]\neg a_\pi$ is transformed into $\neg\exists\pi.\neg[\alpha]\neg a_\pi$, $\neg\exists\pi.\langle\alpha\rangle\neg\neg a_\pi$ and then $\neg\exists\pi.\langle\alpha\rangle a_\pi$ successively.

Then, as another preprocessing step, all programs α appearing in the formula are inductively translated to an intermediate automaton representation $M_\alpha = (Q, q_0, \Sigma^n, \rho, q_f, \Psi)$. The automaton M_α can be seen as a nondeterministic finite automaton (NFA) with access to oracles for the tests in α which recognises all prefixes up to index j of the ω-word $\nu(\Pi[i,\infty])$ such that $(\Pi, i, j) \in R(\alpha)$. This is formalised in Lemma 5. The only differences in syntax when compared to a conventional NFA are (i) there is exactly one final state q_f instead of a set F, (ii) ε-edges are not eliminated and (iii) we have a state marking function Ψ mapping every state $q \in Q$ to a singleton or empty set of formulas $\Psi(q)$. State markings $\Psi(q)$ are introduced to tackle tests $\psi?$ and are later replaced by transitions to automata $\mathcal{A}_\psi$, which we define in the construction for formulas. These state markings make the standard elimination of ε-edges impossible, which is why we delay the elimination until the markings are eliminated as well. This approach is similar to the one used in [10] to transform LDL formulas into automata. However, we apply the construction in a hyperlogic context and make it more succinct by offering an alternative approach to constructing the transition function, thus avoiding an exponential blowup.

Construction of M_α. We now describe the construction of M_α. When an expression α has one or more subexpressions α_i, we assume that automata $M_{\alpha_i} = (Q_i, q_{0,i}, \Sigma^n, \rho_i, q_{f,i}, \Psi_i)$ are already constructed. Intuitively, most cases are analogous to the translation from regular expressions to NFA. For the case of a test $\psi?$, a state marked with ψ is introduced in between starting and final state. Detailed constructions are shown in Figure 1.

Let $\overset{\varepsilon}{\Rightarrow}_X \subseteq Q \times Q$ for a set of formulas X be the smallest relation such that (i) $q \overset{\varepsilon}{\Rightarrow}_{\Psi(q)} q$ and (ii) $q' \overset{\varepsilon}{\Rightarrow}_X q''$ and $q' \in \rho(q, \varepsilon)$ imply $q \overset{\varepsilon}{\Rightarrow}_{X\cup\Psi(q)} q''$. Then, for $\tau \in \Sigma^n$, let $\overset{\tau}{\Rightarrow}_X$ be the smallest relation such that $q \overset{\tau}{\Rightarrow}_X q''$ iff there is a $q' \in Q$ such that $q \overset{\varepsilon}{\Rightarrow}_X q'$ and $q'' \in \rho(q', \tau)$. These relations capture the encountered markings along ε-paths in M_α in the following way: $q \overset{\varepsilon}{\Rightarrow}_X q'$ holds if there is an ε-path from q to q' in M_α that encounters exactly the state markings in the set X. $q \overset{\tau}{\Rightarrow}_X q'$ is used to describe the same behaviour, but requires an additional τ-step at the end.

We obtain the following Lemma:

▶ **Lemma 5.** *Let Π be a path assignment with $\nu(\Pi) = (s_0\tau_1)(s_2\tau_3)...$ and α a program, then $(\Pi, i, k) \in R(\alpha)$ iff there is a state sequence $q_0q_1...q_m$ with $m = \frac{k-i}{2}$ in M_α and sets of formulas $X_0, ..., X_m$ such that*

- **(i)** *q_0 is the initial state of M_α,*
- **(ii)** *$q_m \overset{\varepsilon}{\Rightarrow}_{X_m} q_f$ for the final state q_f of M_α,*
- **(iii)** *$q_l \overset{\tau_{i+2l+1}}{\Longrightarrow}_{X_l} q_{l+1}$ for all $l < m$ and*
- **(iv)** *$\psi \in X_l$ implies $\Pi[i+2l, \infty] \models_\mathcal{T} \psi$ for all $l \leq m$.*

As mentioned, we transform φ into an ABA $\mathcal{A}_\varphi$ recognising $\nu(\Pi)$ iff $\Pi \models_\mathcal{T} \varphi$ holds. Formally, a language $\mathcal{L} \subseteq (S^n \times \Sigma^n)^\omega$ is called $\mathcal{T}$-equivalent to a formula φ, if for each Π the statements $\nu(\Pi) \in \mathcal{L}(\mathcal{A}_\varphi)$ and $\Pi \models_\mathcal{T} \varphi$ are equivalent; we say that an ABA $\mathcal{A}$ is $\mathcal{T}$-equivalent to φ iff its language $\mathcal{L}(\mathcal{A})$ is $\mathcal{T}$-equivalent to φ. As a closed formula φ is a boolean combination of quantified subformulas, the model checking problem can be solved by performing separate nonemptiness checks on $\mathcal{A}_\psi$ for all maximal quantified subformulas ψ and combining the results in accordance with the global structure of φ. For example, if φ is given as $\psi_1 \wedge \neg\psi_2$ for quantified formulas ψ_1 and ψ_2, one performs emptiness tests on $\mathcal{A}_{\psi_1}$ as well as $\mathcal{A}_{\psi_2}$ and accepts iff the first test is positive and the second test is negative.

$\boldsymbol{\tau}$ $M_\alpha = (\{q_0, q_1\}, q_0, \Sigma^n, \rho, q_1, \Psi)$, $\Psi(q_i) = \emptyset$

$$\rho(q, (\sigma_1, \ldots, \sigma_n)) = \begin{cases} \{q_1\} & \text{if } \forall i. \tau|_i = \cdot \vee \tau|_i = \sigma_i \text{ and } q = q_0 \\ \emptyset & \text{else} \end{cases}$$

$\rho(q, \varepsilon) = \emptyset$

$\boldsymbol{\varepsilon}$ $M_\alpha = (\{q_0, q_1\}, q_0, \Sigma^n, \rho, q_1, \Psi)$, $\Psi(q_i) = \emptyset$

$\rho(q_i, \tau) = \emptyset$

$$\rho(q_i, \varepsilon) = \begin{cases} \{q_{i+1}\} & \text{if } i = 0 \\ \emptyset & \text{else} \end{cases}$$

$\boldsymbol{\alpha_1 + \alpha_2}$ $M_\alpha = (Q_1 \dot\cup Q_2 \dot\cup \{q_0, q_f\}, q_0, \Sigma^n, \rho, q_f, \Psi)$,

$\Psi(q_0) = \Psi(q_f) = \emptyset$, $\Psi(q) = \Psi_i(q)$ for $q \in Q_i$

$$\rho(q, \tau) = \begin{cases} \rho_i(q, \tau) & \text{if } q \in Q_i \\ \emptyset & \text{else} \end{cases}$$

$$\rho(q, \varepsilon) = \begin{cases} \{q_{0,1}, q_{0,2}\} & \text{if } q = q_0 \\ \rho_i(q, \varepsilon) & \text{if } q \in Q_i \setminus \{q_{f,i}\} \\ \rho_i(q, \varepsilon) \cup \{q_f\} & \text{if } q = q_{f,i} \end{cases}$$

$\boldsymbol{\alpha_1 \cdot \alpha_2}$ $M_\alpha = (Q_1 \dot\cup Q_2, q_{0,1}, \Sigma^n, \rho, q_{f,2}, \Psi)$, $\Psi(q) = \Psi_i(q)$ for $q \in Q_i$

$\rho(q, \tau) = \rho_i(q, \tau)$ for $q \in Q_i$

$$\rho(q, \varepsilon) = \begin{cases} \rho_i(q, \varepsilon) & \text{if } q \in Q_i, q \neq \{q_{f,1}\} \\ \rho_1(q, \varepsilon) \cup \{q_{0,2}\} & \text{if } q = q_{f,1} \end{cases}$$

$\boldsymbol{(\alpha_1)^*}$ $M_\alpha = (Q_1 \dot\cup \{q_0, q_f\}, q_0, \Sigma^n, \rho, q_f, \Psi)$,$\Psi(q_0) = \Psi(q_f) = \emptyset$, $\Psi(q) = \Psi_1(q)$, for $q \in Q_1$

$$\rho(q, \tau) = \begin{cases} \rho_1(q, \tau) & \text{if } q \in Q_1 \\ \emptyset & \text{else} \end{cases}$$

$$\rho(q, \varepsilon) = \begin{cases} \rho_1(q, \varepsilon) & \text{if } q \notin \{q_0, q_f, q_{f,1}\} \\ \{q_{0,1}, q_f\} & \text{if } q = q_0 \\ \{q_0\} & \text{if } q = q_f \\ \rho_1(q, \varepsilon) \cup \{q_f\} & \text{if } q = q_{f,1} \end{cases}$$

$\boldsymbol{\psi?}$ $M_\alpha = (\{q_0, q_1, q_2\}, q_0, \Sigma^n, \rho, q_2, \Psi)$,$\Psi(q_0) = \Psi(q_2) = \emptyset$, $\Psi(q_1) = \{\psi\}$

$\rho(q, \tau) = \emptyset$

$$\rho(q_i, \varepsilon) = \begin{cases} \{q_{i+1}\} & \text{if } i = 0, 1 \\ \emptyset & \text{else} \end{cases}$$

Figure 1 Construction of M_α.

Construction of $\mathcal{A}_\varphi$. We construct the ABA $\mathcal{A}_\varphi$ inductively. The alphabet of $\mathcal{A}_\varphi$ is given as $\Sigma_\varphi = S^n \times \Sigma^n$ where n is the number of path quantifiers the formula φ is in scope of. When constructing an automaton for φ with subformulas φ_i, we assume that the automata $\mathcal{A}_{\varphi_i} = (Q_{\varphi_i}, q_{0,\varphi_i}, \Sigma_{\varphi_i}, \rho_{\varphi_i}, F_{\varphi_i})$ are already constructed. Similarly, when a formula contains an expression α, we assume that not only $M_\alpha = (Q_\alpha, q_{0,\alpha}, \Sigma^n, \rho_\alpha, q_{f,\alpha}, \Psi)$ but also $\mathcal{A}_{\psi_i}$ in the case of $\langle\alpha\rangle\varphi$ and $\Delta\alpha$ or $\mathcal{A}_{\bar\psi_i}$ in case of $[\alpha]\varphi$ for each construct $\psi_i?$ in α are already constructed. Here, $\bar\psi_i$ is the negation normal form of $\neg\psi_i$. Recall that states *true* and *false* are always part of an ABA in our definition, so we will not mention them explicitly. Furthermore, let $\mathcal{T} = (S, s_0, \{\delta_\sigma \mid \sigma \in \Sigma\}, L)$ be the KTS to be checked.

The cases of the constructions shown in Figure 2 are straightforward: for a_{π_k} (resp. $\neg a_{\pi_k}$), only those words are accepted where the state first read for path π_k is labelled with a (resp. not labelled with a). For the connectives $\wedge$ and $\vee$, one checks whether both automata $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ accept (resp. at least one automaton accepts) the word.

$$
\begin{array}{ll}
\boldsymbol{a_{\pi_k}} & \mathcal{A}_\varphi = (\{q_0\}, q_0, \Sigma_\varphi, \rho, \emptyset) \\
& \rho(q_0, (s,\tau)) = \begin{cases} \mathit{true} & \text{if } a \in L(s|_k) \\ \mathit{false} & \text{else} \end{cases} \\
\boldsymbol{\neg a_{\pi_k}} & \mathcal{A}_\varphi = (\{q_0\}, q_0, \Sigma_\varphi, \rho, \emptyset) \\
& \rho(q_0, (s,\tau)) = \begin{cases} \mathit{false} & \text{if } a \in L(s|_k) \\ \mathit{true} & \text{else} \end{cases} \\
\boldsymbol{\varphi_1 \wedge \varphi_2} & \mathcal{A}_\varphi = (Q_1 \dot\cup Q_2 \dot\cup \{q_0\}, q_0, \Sigma_\varphi, \rho, F_1 \dot\cup F_2) \\
& \rho(q, (s,\tau)) = \begin{cases} \rho_1(q_{0,1}, (s,\tau)) \wedge \rho_2(q_{0,2}, (s,\tau)) & \text{if } q = q_0 \\ \rho_i(q, (s,\tau)) & \text{if } q \in Q_i, i \in \{1,2\} \end{cases} \\
\boldsymbol{\varphi_1 \vee \varphi_2} & \mathcal{A}_\varphi = (Q_1 \dot\cup Q_2 \dot\cup \{q_0\}, q_0, \Sigma_\varphi, \rho, F_1 \dot\cup F_2) \\
& \rho(q, (s,\tau)) = \begin{cases} \rho_1(q_{0,1}, (s,\tau)) \vee \rho_2(q_{0,2}, (s,\tau)) & \text{if } q = q_0 \\ \rho_i(q, (s,\tau)) & \text{if } q \in Q_i, i \in \{1,2\} \end{cases}
\end{array}
$$

Figure 2 Construction of $\mathcal{A}_\varphi$ in basic cases.

We now discuss how to handle the PDL-like modalities $\langle\alpha\rangle\varphi$, $[\alpha]\varphi$ and $\Delta\alpha$. In the correctness statement for the automata M_α constructed for this purpose, we rely on oracle requests for tests $\psi?$ (Lemma 5 (iv)). As mentioned, we eliminate these oracle requests by transitioning into the automaton $\mathcal{A}_\psi$ whenever we reach a state marked with $\psi?$.

In the construction for $\langle\alpha\rangle\varphi_1$, we want to recognise a single path where after a prefix satisfying α, φ_1 holds. This is achieved by enabling a move into $\mathcal{A}_{\varphi_1}$ whenever the final state $q_{f,\alpha}$ of M_α is reached. Since none of the states of M_α are declared final in $\mathcal{A}_\varphi$, an accepting run in $\mathcal{A}_\varphi$ cannot stay in M_α forever and thus eventually has to move into $\mathcal{A}_{\varphi_1}$ in this way. Moves into $\mathcal{A}_\psi$ for tests $\psi?$ are made conjunctively since all tests on an accepting run have to be successful. For the dual case $[\alpha]\varphi_1$, we want φ_1 to hold after all prefixes satisfying α. Thus, dual to the previous construction, whenever reaching the state $q_{f,\alpha}$, we are not only given the possibility to, but have to move into $\mathcal{A}_{\varphi_1}$ as well. Since all transitions in this construction are combined with $\wedge$ to ensure that all prefixes satisfying α are considered, we have to take care of paths in $\mathcal{A}_\varphi$ that never leave M_α by declaring all states of M_α accepting. On the other hand, paths not satisfying α due to a violation of a test $\psi?$ are ruled out by a disjunctive test for the negation of ψ. An illustration of these two cases can be found in Figure 4. To handle formulas of the form $\Delta\alpha$, we transform M_α into a Büchi automaton with a distinguished new initial state q_0 which ensures that in between two visits of q_0, α is satisfied. The state q_0 acts like the initial state $q_{0,\alpha}$ for outgoing, and like the final state $q_{f,\alpha}$ for incoming transitions and is the only accepting state (apart from those in the $\mathcal{A}_{\psi_i}$ automata). Moves into test-automtata are handled just as in the $\langle\alpha\rangle\varphi_1$ case. This ensures that an accepted input word consists of repeated segments matched by α. Negated $\Delta\alpha$ constructions can be handled by complementation using Proposition 2.

Note that when we translate state markings in M_α into transitions in $\mathcal{A}_\varphi$, $q \overset{\varepsilon}{\Rightarrow}_X q'$ may hold for exponentially many sets X, resulting in disjunctions of exponential size (in $|\alpha|$) in the transition functions for $\langle\cdot\rangle$ and Δ, and conjunctions of exponential size for $[\cdot]$. This can, however, be avoided by constructing formulas equivalent to these disjunctions and conjunctions during the inductive construction of M_α, the size of which is not larger than $3 \cdot |\alpha| + 2$. We discuss the case of disjunctions; the other case can be handled in a dual way. Two observations are exploited. First of all, the formula need not be written in a disjunctive form but can mix disjunctions and conjunctions freely. Thus, one source of exponential increase can be avoided by constructing the formulas for nested sums and concatenations

$$
\begin{array}{ll}
\langle\boldsymbol{\alpha}\rangle\boldsymbol{\varphi_1} & \mathcal{A}_\varphi = (Q_1 \dot\cup Q_\alpha \dot\cup \bigcup_i Q_{\psi_i}, q_{0,\alpha}, \Sigma_\varphi, \rho, F_1 \dot\cup \bigcup_i F_{\psi_i}) \\
& \rho(q,(s,\tau)) = \begin{cases}
\rho_1(q,(s,\tau)) & \text{if } q \in Q_1 \\
\rho_{\psi_i}(q,(s,\tau)) & \text{if } q \in Q_{\psi_i} \\
\bigvee\{q' \wedge \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q \overset{\tau}{\Rightarrow}_X q'\} \cup & \\
\quad \{\rho(q_{0,1},(s,\tau)) \wedge & \\
\qquad \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q \overset{\varepsilon}{\Rightarrow}_X q_{f,\alpha}\} & \text{if } q \in Q_\alpha
\end{cases} \\
[\boldsymbol{\alpha}]\boldsymbol{\varphi_1} & \mathcal{A}_\varphi = (Q_1 \dot\cup Q_\alpha \dot\cup \bigcup_i Q_{\bar\psi_i}, q_{0,\alpha}, \Sigma_\varphi, \rho, F_1 \dot\cup Q_\alpha \dot\cup \bigcup_i F_{\bar\psi_i}) \\
& \rho(q,(s,\tau)) = \begin{cases}
\rho_1(q,(s,\tau)) & \text{if } q \in Q_1 \\
\rho_{\bar\psi_i}(q,(s,\tau)) & \text{if } q \in Q_{\bar\psi_i} \\
\bigwedge\{q' \vee \bigvee_{\psi_i \in X} \rho_{\bar\psi_i}(q_{0,\bar\psi_i},(s,\tau)) \mid q \overset{\tau}{\Rightarrow}_X q'\} \cup & \\
\quad \{\rho(q_{0,1},(s,\tau)) \vee & \\
\qquad \bigvee_{\psi_i \in X} \rho_{\bar\psi_i}(q_{0,\bar\psi_i},(s,\tau)) \mid q \overset{\varepsilon}{\Rightarrow}_X q_{f,\alpha}\} & \text{if } q \in Q_\alpha
\end{cases} \\
\boldsymbol{\Delta\alpha} & \mathcal{A}_\varphi = (Q_\alpha \dot\cup \{q_0\} \dot\cup \bigcup_i Q_{\psi_i}, q_0, \Sigma_\varphi, \rho, \{q_0\} \dot\cup \bigcup_i F_{\psi_i}) \\
& \rho(q,(s,\tau)) = \begin{cases}
\rho_{\psi_i}(q,(s,\tau)) & \text{if } q \in Q_{\psi_i} \\
\bigvee\{q' \wedge \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q_{0,\alpha} \overset{\tau}{\Rightarrow}_X q'\} \cup & \\
\quad \{\bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q_{0,\alpha} \overset{\varepsilon}{\Rightarrow}_X q_{f,\alpha}\} & \text{if } q = q_0 \\
\bigvee\{q' \wedge \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q \overset{\tau}{\Rightarrow}_X q'\} \cup & \\
\quad \{q_0 \wedge \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \wedge & \\
\qquad \bigwedge_{\psi_i \in Y} q_{0,\psi_i} \mid q \overset{\tau}{\Rightarrow}_X q' \overset{\varepsilon}{\Rightarrow}_Y q_{f,\alpha}\} & \text{if } q \in Q_\alpha
\end{cases} \\
\neg\boldsymbol{\Delta\alpha} & \mathcal{A}_\varphi = \overline{\mathcal{A}_{\Delta\alpha}}
\end{array}
$$

Figure 3 Construction of $\mathcal{A}_\varphi$ for α formulas.

inductively using that their contribution is directly captured by disjunction and conjunction, respectively. The second observation is that the conjunction resulting from a subpath of a path subsumes the conjunction for that path. This can be exploited to show that only paths using backwards edges (i.e., edges from q_f to q_0 in *-constructions) at most once and using only particular backward edges have to be considered for treating iteration. We refer the interested reader to Appendix B for a more detailed look at this alternative construction.

In the constructions for path quantifiers (Figure 5), we eliminate one component of the alphabet $\Sigma_{\varphi_1} = S^{n+1} \times \Sigma^{n+1}$ and switch to $\Sigma_\varphi = S^n \times \Sigma^n$. The eliminated component is now simulated by the state space of $\mathcal{A}_\varphi$, which ensures that said component is indeed a path in $\mathcal{T}$ by using the additional components from S and Σ. The rule for the initial state guarantees that this path from $\mathcal{T}$ starts in the state it is branching from. Negated existential quantifiers that by our definition can occur in formulas in negation normal form are handled straightforwardly by complementation.

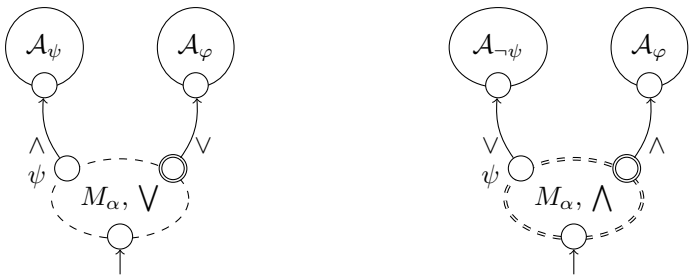

Figure 4 Illustration of the constructions for $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$. In automata shown with a single dashed line all states are non-final. In automata shown with a double dashed line all states are final.

$\exists \pi.\varphi_1$ $\quad \mathcal{A}_{\varphi_1}$ dealternised: $MH(\mathcal{A}_{\varphi_1}) = (Q_1, q_{0,1}, \Sigma_{\varphi_1}, \rho_1, F_1)$
$\mathcal{A}_\varphi = (Q_1 \times S \times \Sigma \dot\cup \{q_0\}, q_0, \Sigma_\varphi, \rho, F_1 \times S \times \Sigma)$
$\rho(q_0, (s,\tau)) = \{(q', s', \sigma') \mid q' \in \rho_1(q_{0,1}, s + s|_n, \tau + \sigma), s' \in \delta_\sigma(s|_n), \sigma, \sigma' \in \Sigma\}$
$\rho((q,s,\sigma),(s,\tau)) = \{(q', s', \sigma') \mid q' \in \rho_1(q, s + s, \tau + \sigma), s' \in \delta_\sigma(s), \sigma' \in \Sigma\}$

$\neg\exists \pi.\varphi_1$ $\quad \mathcal{A}_\varphi = \overline{\mathcal{A}_{\exists \pi.\varphi_1}}$

Figure 5 Construction of $\mathcal{A}_\varphi$ for quantifier formulas.

For the construction, we obtain the following theorem via induction:

▶ **Theorem 6.** *The automaton $\mathcal{A}_\varphi$ is $\mathcal{T}$-equivalent to φ.*

For the complexity analysis, we introduce a notion of criticality.

▶ **Definition 7** (Criticality). *The* criticality *of a HyperPDL-Δ formula φ in negation normal form equals the highest number of critical quantifiers along any path in the formula's syntax tree. A quantifier is called* critical *iff it is a non-outermost quantifier, fulfills at least one of the following three conditions:*

(i) *it is a negated quantifier,*
(ii) *it is an outermost quantifier in a test $\varphi?$ in some program α,*
(iii) *it is an outermost quantifier in the subformula φ of $[\alpha]\varphi$,*

and is not a negated outermost quantifier in a test $\varphi?$ occuring in a modality $[\alpha]$ where the automaton M_α is deterministic.
We call a HyperPDL-Δ formula with criticality 0 uncritical.

This definition is designed carefully in order to ensure that the alternation depth of HyperCTL* formulas coincides with the criticality of their direct translation to HyperPDL-Δ. Intuitively, an *uncritical* quantifier is one where the next dealternation construction $MH(\mathcal{A})$ does not cause another exponential blowup on this part of the automaton. Accordingly, a critical quantifier is one where this exponential blowup for the next dealternation construction cannot be avoided in general. Thus, the criticality of a formula accounts for the number of times an exponential blowup may happen during the construction.

Since exponential blowups occur in a nested manner, we can only bound the size of the resulting automaton by an exponential tower. As argued in the last paragraph, its height is determined by criticality rather than quantifier depth. Formally, we define a function g as $g_{p,c}(0,n) = p(n)$ and $g_{p,c}(k+1,n) = c^{g(k,n)}$ for a constant $c > 1$ and a polynomial p. We use $\mathcal{O}(g(k,n))$ as an abbreviation for $\mathcal{O}(g_{p,c}(k,n))$ for some $c > 1$ and polynomial p.

Two remarks are in order about the next Lemma. Firstly, the statement refers to the number of states of the construction, disregarding the number of transitions. However, this is harmless for our complexity analysis: while the alphabet size increases exponentially with the nesting depth of quantifiers, alphabets need not be represented explicitly and the number of transitions of the automata can be kept polynomial by delaying the substitution of the wildcard symbol $\cdot$ by concrete programs until the intersection with the system $\mathcal{T}$. We refrain from explicating this in our construction in order to increase readability. Secondly, the construction's size can also increase exponentially in the nesting level of negated $\Delta\alpha$ constructions. Since we expect that negated $\Delta\alpha$ constructions are rarely nested in formulas, we assume for the remainder of this paper a bound on this nesting level in order to simplify our complexity statement. Indeed, in Section 6, we will show that a bound of 1 suffices to express ω-regular properties, thus making this a reasonable constraint. The dependency from the nesting depth is reflected in the proof of Lemma 8 in Appendix A.

▶ **Lemma 8.** *The automaton $\mathcal{A}_\varphi$ has size $\mathcal{O}(g(k+1, |\varphi| + log(|\mathcal{T}|)))$ for formulas φ with criticality k.*

Proof (Sketch). By induction on the criticality k. In the base case, the dealternation constructions from Proposition 1 overall cause a single exponential blowup at most since in all constructions not increasing the criticality, one has to keep track of only a single state of each already dealternised subautomaton in further dealternations. Since the only dealternation is done before the system $\mathcal{T}$ is folded around the automaton, the automaton's size is only exponential in the size of φ, but not in the size of $\mathcal{T}$.

In the inductive step, the construction for the outermost critical quantifier increases the size of the automaton exponentially. For all further constructions, it can be argued that the automaton's size is asymptotically not further increased, just like in the base case. ◀

▶ **Theorem 9.** *The problem to decide whether $\mathcal{T} \models \varphi$ holds for KTS $\mathcal{T}$ and HyperPDL-Δ formulas φ with criticality k is in NSPACE$(g(k, |\varphi| + log(|\mathcal{T}|))$.*

Proof. The formula arising from the transformation of φ to our variant of negation normal form is a boolean combination of subformulas ψ with an outermost existential quantifier. Due to dealternation for the existential quantifier, the automata $\mathcal{A}_\psi$ are non-deterministic Büchi automata. We perform nonemptiness checks on these automata separately. It is well-known that the nonemptiness check for Büchi automata is possible in NLOGSPACE in the size of the automaton [9]. We then combine the results in accordance with the structure of φ. This does not add to the complexity. Thus, by Lemma 8, we obtain an NSPACE$(g(k, |\varphi| + log(|\mathcal{T}|))$ model checking algorithm for criticality k HyperPDL-Δ formulas. ◀

Since we can easily translate HyperCTL* formulas to HyperPDL-Δ while preserving the alternation depth as criticality, we can use known hardness results for HyperCTL* model checking [20] to obtain the following Theorem:

▶ **Theorem 10.** *Given a KTS $\mathcal{T}$ and a HyperPDL-Δ formula φ with criticality k, the model checking Problem for HyperPDL-Δ is hard for NSPACE$(g(k, |\varphi|))$ and NSPACE$(g(k-1, |\mathcal{T}|))$.*[1]

5 Satisfiability

While model checking of temporal logics is an essential technique for verification, it requires meaningful specifications in order to be useful. For example, if a formula is fulfilled by every or no structure, it is useless for specification purposes. To evaluate whether this is the case, satisfiability testing can be employed as a sanity check [21]. Since satisfiability checking is already undecidable for HyperLTL [11] via a reduction from the Post Correspondence Problem (PCP) and HyperLTL can be embedded into HyperPDL-Δ, we consider only restrictions of a fragment of HyperPDL-Δ where quantified paths can be traversed linearly for the purpose of this section.

[1] Note that we have not defined $g(-1, n)$ in this paper. For $k = 0$ and a fixed size formula φ, we use the definition from [12], where NSPACE$(g(-1, n))$ was defined as NLOGSPACE. Since one can see that our algorithm has NLOGSPACE complexity in this instance, the lower and upper bounds match in all cases. For $k = 0$ and a fixed size structure $\mathcal{T}$ or $k > 1$, we can use Savitch's Theorem to see that the problems are actually complete for the deterministic space classes.

▶ **Definition 11.** *A linear HyperPDL-Δ formula is of the form* $Q_1\pi_1 \dots Q_n\pi_n.\psi$ *for* $Q_i \in \{\exists, \forall\}$ *and* ψ *a quantifier-free formula. A linear HyperPDL-Δ formula* φ *is an* $\forall^*$*-formula if* $Q_i = \forall$ *for all* $1 \leq i \leq n$*. The* $\exists^*$*-fragment and the* $\exists^*\forall^*$*-fragment are defined analogously.*

For linear HyperPDL-Δ, we generalise the semantics to arbitrary sets of traces T instead of only those induced by KTS. More concretely, for a fixed set of traces T, we let the assignment functions Π map variables to traces in T instead of paths of a structure. Such a function Π is called a *trace assignment*; the set of all trace assignments is called TA. Furthermore, we let all quantifiers range over T instead of $Paths(\mathcal{T}, \Pi(\epsilon)(0))$. We further define $T \models \varphi$ to hold for a linear HyperPDL-Δ formula φ iff $\{\} \models \varphi$ holds for the trace assignment $\{\}$ with empty domain over T. For linear HyperPDL-Δ formulas and using $T = Traces(\mathcal{T}, s_0)$, this definition coincides with the definition in Section 3. We then call a linear HyperPDL-Δ formula φ *satisfiable* iff there is a non-empty set of traces T such that $T \models \varphi$ holds. The satisfiability problem for linear HyperPDL-Δ is to check whether a linear HyperPDL-Δ formula φ is satisfiable. For full linear HyperPDL-Δ, we obtain undecidability via a reduction from HyperLTL satisfiability [11]:

▶ **Theorem 12.** *The satisfiability problem for linear HyperPDL-Δ is undecidable.*

However, since the encoding relies on the availability of arbitrary combinations of quantifiers, a natural question is whether fragments of linear HyperPDL-Δ with restricted quantifier combinations yield a decidable satisfiability problem. For HyperLTL, several such restrictions were considered [11]. In [11], the satisfiability problem for the restricted fragments in HyperLTL is solved via the transformation of a HyperLTL formula to an equisatisfiable LTL formula and solving the satisfiability problem for the latter. Since there is no apparent connection of this form between HyperPDL-Δ and LTL, we use a similar type of translation, but instead translate our formulas into suitable ABA and check those for emptiness. In all cases, the lower complexity bound can be obtained via reduction from the satisfiability problem of the corresponding fragment of HyperLTL.

▶ **Theorem 13.** *The satisfiability problem for the* $\forall^*$*-fragment of HyperPDL-Δ is PSPACE-complete.*

Proof (Sketch). Let φ be a $\forall^*$-fragment formula, i.e. $\varphi \equiv \forall\pi_1 \dots \forall\pi_n.\psi$ for a quantifier-free formula ψ. We manipulate ψ by substituting $\pi_1, ..., \pi_n$ with a single fresh variable π and obtain a formula ψ'. This is done by (i) replacing every occurence of an atomic proposition a_{π_i} with a_π and (ii) compressing each tuple of atomic programs τ into a program α with only 1-tuples. The compression discriminates three cases. If τ only consists of wildcard programs, i.e. $\tau = \bullet$, then it is compressed to $(\cdot)$. If τ is composed from the set $\{\sigma, \cdot\}$ for some atomic program σ, then it is compressed to (σ). Otherwise, i.e. if τ is composed of distinct non-wildcard atomic programs, it is compressed to $\mathit{false}? \cdot (\cdot)$. For example, for $\varphi \equiv \forall\pi_1\forall\pi_2\langle(\cdot, \sigma)\rangle a_{\pi_1} \wedge [\bullet + (\sigma, \sigma')]\neg a_{\pi_2}$, ψ' is given by $\langle(\sigma)\rangle a_\pi \wedge [(\cdot) + \mathit{false}? \cdot (\cdot)]\neg a_\pi$. Let $\mathcal{A}$ be the ABA for ψ' as described in Section 4. We can test $\mathcal{A}$ for emptiness in PSPACE [9]. $\mathcal{A}$ is non-empty iff φ is satisfiable: any word w accepted by $\mathcal{A}$ gives rise to the trace set $\{w\}$ satisfying φ. Analogously, a non-empty trace set T with $T \models \varphi$ can be used to obtain a word accepted by $\mathcal{A}$ since $\mathcal{A}$ accepts all traces satisfying ψ' and we can thus pick any trace t in T as a witness.

The lower bound directly follows by a reduction from the satisfiability problem for the $\exists^*\forall^*$ fragment of HyperLTL [11]. ◀

Similarly, we can show that the $\exists^*$ fragment is also PSPACE-complete.

▶ **Theorem 14.** *The satisfiability problem for the $\exists^*$-fragment of HyperPDL-Δ is PSPACE-complete.*

Proof (Sketch). Let φ be a $\exists^*$-fragment formula, i.e. $\varphi \equiv \exists \pi_1 \dots \exists \pi_n . \psi$ for a quantifier-free formula ψ. We construct the alternating Büchi automaton $\mathcal{A}$ for ψ. Non-emptiness of $\mathcal{A}$ and satisfiability of φ are equivalent: every trace set T fulfilling φ contains traces $t_1 \dots t_n$ fulfilling ψ and these give rise to a word accepted by $\mathcal{A}$. On the other hand, if $\mathcal{L}(\mathcal{A})$ is non-empty, the trace set T induced by an arbitrary $w \in \mathcal{L}(\mathcal{A})$ fulfills φ.

The lower bound follows straightforwardly by a reduction from the satisfiability problem for the $\exists^*$ fragment of HyperLTL [11]. ◀

For the $\exists^*\forall^*$-fragment, we eliminate the universal quantifiers by taking the variables bound by existential quantifiers and replacing the variables bound by universal quantifiers by all possible combinations of them. Unlike the previous two fragments, this increases the complexity beyond PSPACE.

▶ **Theorem 15.** *The satisfiability problem is EXPSPACE-complete for the $\exists^*\forall^*$-fragment of HyperPDL-Δ.*

Proof (Sketch). Let φ be a $\exists^*\forall^*$-formula, i.e. $\varphi \equiv \exists \pi_1 \dots \exists \pi_n \forall \pi'_1 \dots \forall \pi'_m . \psi$ for a quantifier-free formula ψ. For a formula ψ and path variables π, π', we define the substitution $\psi[\pi/\pi']$ to be the variant of ψ in which all occurences of π' have been replaced by π similar to the substitution in the proof of Theorem 13. The main difference here is that only two instead of n path variables are compressed into one. Thus only two instead of all atomic programs have to be considered when determining the replacement of a tuple. Let $\varphi' \equiv \exists \pi_1 \dots \exists \pi_n . \bigwedge_{j_1=1}^{n} \cdots \bigwedge_{j_m=1}^{n} \psi[\pi_{j_1}/\pi'_1] \dots [\pi_{j_m}/\pi'_m]$. For example, for $\varphi \equiv \exists \pi_1 . \exists \pi_2 . \forall \pi'_1 . \langle (\cdot, \cdot, \sigma) \rangle a_{\pi_1} \wedge \neg a_{\pi_2} \wedge a_{\pi'_1}$, $\varphi' = \exists \pi_1 . \exists \pi_2 . (\langle (\sigma, \cdot) \rangle a_{\pi_1} \wedge \neg a_{\pi_2} \wedge a_{\pi_1}) \wedge (\langle (\cdot, \sigma) \rangle a_{\pi_1} \wedge \neg a_{\pi_2} \wedge a_{\pi_2})$. φ' is an $\exists^*$ formula and is equisatisfiable to φ: any trace assignment satisfying φ naturally induces a model of φ'. For the reverse direction, assume $\Pi \models \varphi'$. φ' contains all possible combinations of assignments for the variables $\pi'_1 \dots \pi'_m$ with traces chosen for the existentially quantified variables $\pi_1 \dots \pi_n$. Then $T = \{\Pi(\pi_i) \mid 1 \leq i \leq n\} \models \varphi$. φ' is constructible in EXPTIME. Therefore the satisfiability check is possible in EXPSPACE due to Theorem 14.

The lower bound easily follows by a reduction from the satisfiability problem for the $\exists^*\forall^*$ fragment of HyperLTL [11]. ◀

As in [11], from Theorem 15, we obtain that it is an EXPSPACE-complete problem to decide whether one uncritical HyperPDL-Δ formula is implied by another. In particular, the uncritical fragment includes properties like variants of observational determinism enriched by regular predicates.

6 Expressivity Results

As mentioned in the introduction, a desirable property of temporal logics is the ability to specify arbitrary ω-regular properties. We show that HyperPDL-Δ indeed has this property.

▶ **Theorem 16.** *Let Π be a trace assignment and $\pi_1 \dots \pi_n$ be the variables bound by Π. Let $\nu_{AP} : TA \to ((2^{AP})^n \times \Sigma^n)^\omega$ be the analog of ν for trace assignments. For a given ω-regular language $\mathcal{L}$ over $(2^{AP})^n \times \Sigma^n$, there is a quantifier-free HyperPDL$-\Delta$ formula φ with path variables $\pi_1 \dots \pi_n$ such that $\Pi \models \varphi$ iff $\nu_{AP}(\Pi) \in \mathcal{L}$.*

Proof. Let $\mathcal{L}$ be an ω-regular language over $(2^{AP})^n \times \Sigma^n$. It is well-known [9] that $\mathcal{L} = \bigcup_{i=1}^{k} \mathcal{L}_{i,0}\mathcal{L}_{i,1}^{\omega}$ holds for some regular languages $\mathcal{L}_{i,0}$, $\mathcal{L}_{i,1}$. Let $r_{i,j}$ be a regular expression for $\mathcal{L}_{i,j}$. Every symbol in $r_{i,j}$ has the form $((P_1, ..., P_n), \tau)$ for $P_k \subseteq AP$ and $\tau \in \Sigma^n$. Let $\alpha_{i,j}$ be the regular expression obtained by replacing each such symbol in $r_{i,j}$ by $(\bigwedge_{l=1}^{n} \bigwedge_{a \in P_l} a_{\pi_l} \wedge \bigwedge_{a \notin P_l} \neg a_{\pi_l})? \cdot \tau$. Then $\varphi \equiv \bigvee_{i=1}^{k} \langle \alpha_{i,0} \rangle \Delta \alpha_{i,1}$ yields the desired formula. ◀

It follows from this theorem that HyperPDL-Δ can express an infinitary version of the regular hyperlanguages recently proposed in [3].

We now compare our logic to other hyperlogics. For this purpose we introduce a logic that adds to HyperCTL* the ability to quantify over atomic propositions.

▶ **Definition 17** ([7]). *The logic HyperQCTL* is obtained by adding to the syntax of HyperCTL* the rules $\varphi ::= q \mid \exists q.\varphi$ and to the semantics the rules*

$$\Pi \models_{\mathcal{T}} q \quad \text{iff} \quad q \in \Pi(\pi_q)(0)$$
$$\Pi \models_{\mathcal{T}} \exists q.\varphi \quad \text{iff} \quad \exists t \in (2^{\{q\}})^{\omega}.\Pi[\pi_q \to t] \models_{\mathcal{T}} \varphi$$

The sub-logic HyperQPTL consists of the HyperQCTL* formulas where both path quantifiers $Q\pi.\varphi$ and propositional quantifiers $Qq.\varphi$ only occur at the front of the formula.

▶ **Theorem 18.**
1. *HyperCTL* $<$ HyperPDL-Δ $\leq$ HyperQCTL**
2. *HyperLTL $<$ Linear HyperPDL-Δ $\leq$ HyperQPTL*

Proof. Part one of both claims is straightforward: Embedding HyperLTL and HyperCTL* into (linear) HyperPDL-Δ works as described in Section 3. An embedding in the other direction is impossible due to the inability of HyperLTL and HyperCTL* to express arbitrary ω-regular properties [20].

For the second part of the first claim, we observe that the semantics of HyperPDL-Δ can straightforwardly be encoded in MSO[E], which is equally expressive as HyperQCTL* by [7]. The last claim can be shown by a direct translation via Büchi automata: A quantifier-free HyperPDL-Δ formula can be translated into a Büchi Automaton as described in Section 4. By [15], there is a QPTL formula for that automaton, where the quantifiers can be reattached to yield the desired formula. ◀

By the results of [7], we obtain that linear HyperPDL-Δ is stricly less expressive than S1S[E], which, as mentioned, has an undecidable model checking problem. We leave a more precise localisation of linear and unrestricted HyperPDL-Δ in the hierarchies of hyperlogics from [7] for future work. This includes comparisons with FO[<,E] and MPL[E] and an answer to the question if the second inequalities from the claims in Theorem 18 are indeed strict.

7 Conclusion

We introduced the logic HyperPDL-Δ as a variant of Propositional Dynamic Logic for hyperproperties that can express all ω-regular properties. Our model checking algorithm has the same complexity as model checking HyperCTL*, despite the increased expressive power. Finally, we showed that satisfiability checking for certain fragments has the same complexity as for structurally similar, but less expressive fragments of HyperLTL.

Future work includes implementing a model checker for HyperPDL-Δ. It would also be interesting to explore alternative model checking and satisfiability testing algorithms for subfragments of HyperPDL-Δ, possibly by exploiting classical techniques for PDL.

References

1 Roy Armoni, Limor Fix, Alon Flaisher, Rob Gerth, Boris Ginsburg, Tomer Kanza, Avner Landver, Sela Mador-Haim, Eli Singerman, Andreas Tiemeyer, et al. The ForSpec temporal logic: A new temporal property-specification language. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 296–311. Springer, 2002. doi:10.1007/3-540-46002-0_21.

2 Borzoo Bonakdarpour, Cesar Sanchez, and Gerardo Schneider. Monitoring hyperproperties by combining static analysis and runtime verification. In *International Symposium on Leveraging Applications of Formal Methods*, pages 8–27. Springer, 2018. doi:10.1007/978-3-030-03421-4_2.

3 Borzoo Bonakdarpour and Sarai Sheinvald. Automata for hyperlanguages, 2020. arXiv:2002.09877.

4 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In *FoSSaCS*, volume 9034 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11.

5 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *POST 2014*, pages 265–284, 2014. doi:10.1007/978-3-642-54792-8_15.

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

7 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785713.

8 Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

9 Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi:10.1017/CBO9781139236119.

10 Peter Faymonville and Martin Zimmermann. Parametric linear dynamic logic. *Information and Computation*, 253:237–256, 2017. GandALF 2014. doi:10.1016/j.ic.2016.07.009.

11 Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *CONCUR 2016*, pages 13:1–13:14, 2016. doi:10.4230/LIPIcs.CONCUR.2016.13.

12 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *CAV 2015*, pages 30–48, 2015. doi:10.1007/978-3-319-21690-4_3.

13 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1.

14 Joseph Y. Halpern, Ron van der Meyden, and Moshe Y. Vardi. Complete axiomatizations for reasoning about knowledge and time. *SIAM J. Comput.*, 33(3):674–703, 2004. doi:10.1137/S0097539797320906.

15 Yonit Kesten and Amir Pnueli. Complete proof system for QPTL. *J. Log. Comput.*, 12(5):701–745, 2002. doi:10.1093/logcom/12.5.701.

16 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Extended temporal logic revisited. In *CONCUR 2001*, pages 519–535, 2001. doi:10.1007/3-540-44685-0_35.

17 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Logic*, 2(3):408–429, July 2001. doi:10.1145/377978.377993.

18 Martin Lange. Model checking propositional dynamic logic with all extras. *J. Applied Logic*, 4(1):39–49, 2006. doi:10.1016/j.jal.2005.08.002.

19 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32(3):321–330, 1984. doi:10.1016/0304-3975(84)90049-5.

20 Markus N. Rabe. *A temporal logic approach to Information-flow control*. PhD thesis, Saarland University, 2016. doi:10.22028/D291-26650.

21 Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. In *SPIN 2007*, pages 149–167, 2007. doi:10.1007/978-3-540-73370-6_11.

22 Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5.

A Detailed Complexity Analysis

Proof of Lemma 8. By induction on the criticality k.

Base case: φ has criticality 0. We show inductively that $\mathcal{A}_\varphi$ has size $2^{\mathcal{O}(p(n)+p'(log(m)))}$ in the size n of φ and the size m of $\mathcal{T}$ for some polynomials p, p'. First, notice that $|M_\alpha|$ is linear in the size of α. This can easily be shown by a structural induction, where each construction adds a constant number of states to its subautomata only.

Basic constructions $\mathcal{A}_{a_\pi}$ and $\mathcal{A}_{\neg a_\pi}$ have constant size. For boolean connectives as well as $\langle\alpha\rangle\varphi, [\alpha]\varphi$ and $\Delta\alpha$, the construction of $\mathcal{A}_\varphi$ again just adds a constant number of states to the automata for the subformulas. The construction for $\neg\Delta\alpha$ introduces a quadratic increase by Proposition 2. Throughout the whole construction, this results in an exponential increase in the nesting depth of negated $\Delta\alpha$ constructs at most. More precisely, when bounding this nesting depth to a constant d, the polynomial p on top of the exponential tower has degree at most $2d$. Existential quantifiers increase the size of the automaton exponentially in the size of the formula φ and add a factor polynomial in the size of the structure $\mathcal{T}$. Using logarithmic laws, this translates to the form above. Note that the factor depending on $|\mathcal{T}|$ is added after the exponential blowup from the dealternation construction $MH(\mathcal{A})$.

It remains to show that the dealternation construction $MH(\mathcal{A})$ introduces an exponential blowup of the structure's size at most once for formulas of criticality 0, regardless of how many quantifiers the formula contains. In order to do this, we have to look closer at the proof of Proposition 1. We show that once the dealternation construction $MH(\mathcal{A})$ is done for the innermost quantifiers, at most one state of each dealternised automaton has to be tracked in further dealternations. Thus, the exponential size of the subautomaton is added as a factor rather than in an exponent when determining the size of the state space of the full automaton.

Our claim can be shown by an induction over the number of constructions on top of the dealternised automaton. In the base case, no construction is done on top of a dealternised automaton $\mathcal{A}_\varphi$. Since $\mathcal{A}_\varphi$ is a Büchi automaton, a run of the resulting automaton is a path rather than a tree on every word. Thus, only one state has to be tracked. In the inductive step, we discriminate cases for the outermost construction. By the induction hypothesis, at most one state of each dealternised automaton has to be tracked in each subautomaton. For the construction $\varphi_1 \vee \varphi_2$, a run tree moving into $\mathcal{A}_{\varphi_1}$ or $\mathcal{A}_{\varphi_2}$ never returns to the initial state. Thus, since $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ are unconnected, we track states of only one of the automata. Then, the claim is implied by the induction hypothesis. The construction for $\mathcal{A}_{\varphi_1 \wedge \varphi_2}$ works similarly, with the difference that we have to track states of both subautomata when a run moves into this automaton over the initial state. This does, however, not lead to an increase in states of each dealternised automaton that have to be tracked, since these subautomata are unconnected. The next construction we have to consider is $\mathcal{A}_{\langle\alpha\rangle\varphi}$. Here, a run has the property that at most one state of M_α has to be tracked, which can be replaced by states of $\mathcal{A}_\varphi$ at some point. Additionally, arbitrarily many states of $\mathcal{A}_\psi$ for subformulas ψ of α can be tracked. However, this is no contradiction to our claim, since α may not

contain any quantified subformulas and thus $\mathcal{A}_\psi$ may not contain dealternised automata in uncritical formulas. Thus, since the induction hypothesis states that at most one state of each dealternised automaton of $\mathcal{A}_\varphi$ has to be tracked at any point, this shows our claim. As another case, we consider the construction for $\exists\pi.\varphi$. Here, since only a disjunctive transition is added on top of $\mathcal{A}_\varphi$, we can argue similar as in the case for $\varphi_1 \vee \varphi_2$ with the difference that we consider only a single subautomaton. Due to the exemption rule in the definition of criticality, there is an additional construction to be considered: $[\alpha]\varphi$, where α is deterministic and the outermost quantifier inside α is negated.[2] Since tests ψ in α are handled by disjunctively transitioning into the automaton for $\neg\psi$ in the $[\alpha]\varphi$ construction, this cancels out the negation of the quantifier. Therefore, no critical negation construction has to be performed on a dealternised subautomaton during the construction of $\mathcal{A}_{\neg\psi}$. Then, since due to the fact that M_α is deterministic, conjunctions of transitions in M_α behave the same as disjunctions and we can argue just as in the case for $\mathcal{A}_{\langle\alpha\rangle\varphi}$. Finally, observe that we do not have to consider constructions for $\Delta\alpha$, $\neg\Delta\alpha$, or general $[\alpha]\varphi$, since the resulting formula is not uncritical when any of these contain a quantified subformula.

Inductive step: φ has criticality $k+1$. On the path in φ's syntax tree inducing the criticality, we inspect the outermost critical quantifier. Its subformulas φ_i have criticality at most k. Using the induction hypothesis on all subformulas, we obtain automata of size at most $\mathcal{O}(g(k+1, |\varphi_i|+log(|\mathcal{T}|)))$. The next dealternation will result in an automaton of size $2^{\mathcal{O}(|\mathcal{A}_{\varphi_i}|)}$ (by Proposition 1) which can be bounded by $2^{\mathcal{O}(g(k+1,|\varphi_i|+log(|\mathcal{T}|)))} = \mathcal{O}(g(k+2, |\varphi|+log(|\mathcal{T}|)))$. Since we inspected the outermost critical quantifier on the path inducing the criticality of the formula, any of the subsequent constructions will not cause a further exponential blowup of the automaton's size, as argued in the base case. ◀

B Alternative Construction for the Transition Function of $\mathcal{A}_\varphi$

In the main body of the paper, we have argued that the transition function in $\mathcal{A}_\varphi$ for φ containing α can be exponential in the size of α. Consider the automaton in Figure 6 where states annotated with $\psi_{i,j}$ are marked with the corresponding formula and each unannotated edge stands for an ε-transition. Such an automaton can occur when α has the form $(\psi_{0,1}? + \psi_{0,2}?)(\psi_{1,1}? + \psi_{1,2}?)...(\psi_{n,1}? + \psi_{n,2}?)\tau$ and should serve as an illustrative example. We consider the case where this α is used in a $\langle . \rangle$ formula. For ease of presentation, we assume that $\psi_{i,j}$ can be tested by moving into a state $p_{i,j}$. It is possible to reach q_f from q_0 with an exponential number of ε-paths, each with a different combination of markings. Therefore we have $q_0 \overset{\tau}{\Rightarrow}_X q$ for exponentially many X, transferring into the size of the transition function when constructing $\rho(q_0, \tau) \equiv \bigvee\{q \wedge \bigwedge_{\psi_{i,j}\in X} p_{i,j} \mid q_0 \overset{\tau}{\Rightarrow}_X q\}$.

For this example, it is easy to see that these exponentially many different combinations of transitions could equally be represented by a formula of a much smaller size, namely $(p_{0,1} \vee p_{0,2}) \wedge (p_{1,1} \vee p_{1,2}) \wedge ... \wedge (p_{n,1} \vee p_{n,2})$. This is due to the fact that conjunction and disjunction closely resemble the behaviour of concatenation and sum constructions in M_α when considering ε-paths for the construction of ρ. We will show here, that using these ideas it is possible to construct such a formula of size not greater than $3 \cdot |\alpha| + 2$ for every α.

[2] There are additional forms of α, where explosion through a negated quantifier inside the modality $[\alpha]$ can be avoided. This includes all forms where in any run in M_α, a test for $\neg\psi$ occurs only in a situation where all states occurring at the same level of the run can transition into $\mathcal{A}_{\neg\psi}$. Then, when a transition into $\mathcal{A}_{\neg\psi}$ can be taken in one of the states, it can be taken in all of the states. Since they are on the same level, the same continuation in $\mathcal{A}_{\neg\psi}$ can be used for all these branches, such that only a single state of each dealternised subautomaton of $\mathcal{A}_{\neg\psi}$ needs to be tracked.

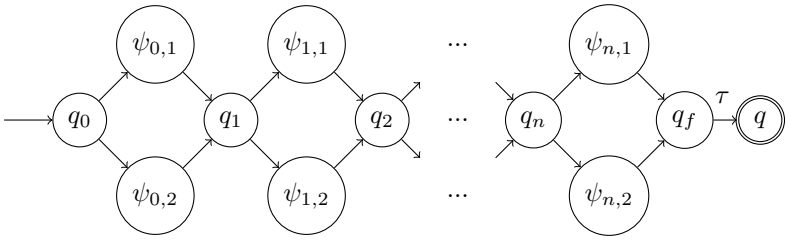

Figure 6 Automaton M_α with an exponential number of test combinations.

We proceed by constructing a function εp such that $\varepsilon p(q, q', \alpha) = \vartheta$ for a formula ϑ equivalent to $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\}$ for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from M_α. Here we use variables v_i as placeholders for formulas ψ_i to be later replaced by a transition into the corresponding automaton $\mathcal{A}_{\psi_i}$. For τ-transitions, this can straightforwardly be extended to a function τp which can then be used to construct the τ transition function for a state q in a more succinct way. For a $\langle\alpha\rangle\varphi$ formula and some $q \in Q_\alpha$ we then have $\rho(q, (s, \tau)) = \bigvee\{q' \wedge \tau p(q, q', \alpha)[\rho_{\psi_i}(q_{0,\psi_i}, (s, \tau))/v_i] \mid q' \in Q\}$. Some remarks are in order for this transition function construction: (i) in this construction, opposed to the one used before, each $q' \in Q$ can occur at most once in the disjunction, (ii) for $q' \in Q$ such that there is no X with $q \overset{\tau}{\Rightarrow}_X q'$, i.e. q' is not reachable from q with τ, we have $\tau p(q, q', \alpha) \equiv \mathit{false}$ and thus the state can be eliminated from the disjunction and (iii) for [.] formulas, τp and ρ can similarly be constructed in a dual way.

Since there is no direct way to create the desired formula for $\alpha = \alpha_1^*$ while meeting the size constraints, we cannot do a direct inductive construction for εp. Instead we perform an inductive construction dp that is similar to εp but does not take *backwards edges* (q_f, q_0) originating from α^*-constructions into account. Then, εp can be constructed from dp by only considering a single backwards edge for each pair of states. The idea behind this is that each path p_* considering more than one backwards edge is *subsumed* by some path p_1 considering only one backwards edge in the sense that if p_* visits the set X_* of markings and p_1 visits the set X_1 of markings, then $X_1 \subseteq X_*$. Then, the conjunction over X_* is implied by the conjunction over X_1. Since in the construction of ρ, we perform a disjunction over all paths with a conjunction over all seen markings inside, we can then omit p_* from the disjunction.

Construction of dp and εp. First, we construct dp inductively.

$$\alpha = \tau \quad dp(q, q', \alpha) = \begin{cases} \mathit{false} & \text{if } q \neq q' \\ \mathit{true} & \text{else} \end{cases}$$

$$\alpha = \varepsilon \quad dp(q, q', \alpha) = \begin{cases} \mathit{false} & \text{if } q = q_1 \text{ and } q' = q_0 \\ \mathit{true} & \text{else} \end{cases}$$

$$\alpha = \alpha_1 + \alpha_2 \quad dp(q, q', \alpha) = \begin{cases} dp(q, q', \alpha_i) & \text{if } q, q' \in Q_i \\ dp(q_{0,i}, q', \alpha_i) & \text{if } q = q_0 \text{ and } q' \in Q_i \\ dp(q, q_{f,i}, \alpha_i) & \text{if } q \in Q_i \text{ and } q' = q_f \\ dp(q_{0,1}, q_{f,1}, \alpha_1) \vee dp(q_{0,2}, q_{f,2}, \alpha_2) & \text{if } q = q_0 \text{ and } q' = q_f \\ \mathit{false} & \text{else} \end{cases}$$

$$\alpha = \alpha_1 \cdot \alpha_2 \quad dp(q,q',\alpha) = \begin{cases} dp(q,q',\alpha_i) & \text{if } q,q' \in Q_i \\ dp(q,q_{f,1},\alpha_1) \wedge dp(q_{0,2},q',\alpha_2) & \text{if } q \in Q_1 \text{ and } q' \in Q_2 \\ \mathit{false} & \text{else} \end{cases}$$

$$\alpha = (\alpha_1)^* \quad dp(q,q',\alpha) = \begin{cases} dp(q,q',\alpha_1) & \text{if } q,q' \in Q_1 \\ \mathit{true} & \text{if } q = q_0 \text{ and } q' = q_f \\ dp(q_{0,1},q',\alpha_1) & \text{if } q = q_0 \text{ and } q' \in Q_1 \\ dp(q,q_{f,1},\alpha_1) & \text{if } q \in Q_1 \text{ and } q' = q_f \\ \mathit{false} & \text{else} \end{cases}$$

$$\alpha = \psi_k? \quad dp(q,q',\alpha) = \begin{cases} \mathit{true} & \text{if } q = q' \neq q_1 \\ \mathit{false} & \text{if } q = q_i, q' = q_j, i > j \\ v_k & \text{else} \end{cases}$$

Using dp, we are now able to construct εp directly for all q, q' and α.

$$\varepsilon p(q,q',\alpha) = dp(q,q',\alpha) \vee (dp(q,q_{f,\bar{\alpha}},\alpha) \wedge dp(q_{0,\bar{\alpha}},q',\alpha))$$

Here $\bar{\alpha}$ is the innermost $*$-construction that contains both q and q'. In case no such $\bar{\alpha}$ exists, both $dp(q,q_{f,\bar{\alpha}},\alpha)$ and $dp(q_{0,\bar{\alpha}},q',\alpha)$ are given by *false* instead.

Theoretical justification. In order to use this succinct alternative in our construction, we have to argue that it indeed has the desired properties. Therefore we establish a number of theorems:

▶ **Theorem 19.** *$|dp(q,q',\alpha)| \leq |\alpha|$ and $|\varepsilon p(q,q',\alpha)| \leq 3 \cdot |\alpha| + 2$ for all q, q' and α.*

Proof. The first claim can be established by a straightforward structural induction on α. It is easy to see that in each case of the construction, at most one operator is added to $dp(q,q',\alpha)$ and each partial term is used at most once.

The second claim follows directly from the first claim and the definition of εp. ◀

▶ **Lemma 20.** *We have $dp(q,q',\alpha) \equiv \bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\}$ for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from M_α where all backwards edges from $*$-constructions are removed.*

Proof. We show this claim by a structural induction on α.

Case $\alpha = \tau$: There are two unmarked states q_0, q_1 in M_α with a τ-transition connecting them. There are no ε-transitions. Thus, we have $q \overset{\varepsilon}{\Rightarrow}_X q'$ iff $q = q'$ and $X = \emptyset$. Therefore we have $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \mathit{false}$ for $q \neq q'$ and $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \mathit{true}$ for $q = q'$, establishing the claim.

Case $\alpha = \varepsilon$: There are two unmarked states q_0, q_1 in M_α with an ε-transition connecting q_0 to q_1. Thus, we have $q \overset{\varepsilon}{\Rightarrow}_X q'$ iff $X = \emptyset$ and either $q \neq q_1$ or $q' \neq q_0$. Therefore we have $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \mathit{false}$ for $q = q_1$,$q' = q_0$ and $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \mathit{true}$ else, establishing the claim.

Case $\alpha = \alpha_1 + \alpha_2$: By induction hypothesis, the claim holds for α_1 and α_2. To obtain M_α from M_{α_1} and M_{α_2}, a new starting and final state are added with ε-transitions to the old starting states and from the old final states, respectively. We consider different cases how a path inducing $q \overset{\varepsilon}{\Rightarrow}_X q'$ could have been constructed. In the first case, where both q and q' are in the same automaton M_{α_i}, no additional paths could have been introduced by the new transitions. Thus, the claim follows immediately from the induction hypothesis. In the second case, where $q = q_0$ and q' is in M_{α_i} a path must take the ε-transition to $q_{0,i}$ and then take a path between $q_{0,i}$ and q'. Since q_0 is not

marked, we have $q_0 \overset{\varepsilon}{\Rightarrow}_X q'$ iff $q_{0,i} \overset{\varepsilon}{\Rightarrow}_X q'$, establishing the claim by induction hypothesis. The third case, where q is in M_{α_i} and $q' = q_f$ is analogous to the second one. Another case is $q = q_0$ and $q' = q_f$. Since M_{α_1} and M_{α_2} are not connected, we have $q \overset{\varepsilon}{\Rightarrow}_X q'$ iff $q_{0,1} \overset{\varepsilon}{\Rightarrow}_X q_{f,1}$ or $q_{0,2} \overset{\varepsilon}{\Rightarrow}_X q_{f,2}$ with the same argument as used in cases two and three. Since no paths are added from $q_{0,i}$ to $q_{f,i}$ when going over from M_{α_i} to M_α, $q_{0,i} \overset{\varepsilon}{\Rightarrow}_X q_{f,i}$ holds for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from M_{α_i} iff it holds for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from M_α. Therefore we have $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q_{0,1} \overset{\varepsilon}{\Rightarrow}_X q_{f,1}\} \vee \bigvee\{\bigwedge_{\psi_i \in X} v_i | q_{0,2} \overset{\varepsilon}{\Rightarrow}_X q_{f,2}\} \equiv dp(q_{0,1}, q_{f,1}, \alpha_1) \vee dp(q_{0,2}, q_{f,2}, \alpha_2) = dp(q, q', \alpha)$ using the induction hypothesis. The remaining cases include $q = q_f$ with $q' = q_0$ and q being in M_{α_i} with q' being in $M_{\alpha_{1-i}}$. In both cases, q' is not reachable from q using only ε-transitions. Thus, the claim is established with similar arguments as in previous cases.

Case $\alpha = \alpha_1 \cdot \alpha_2$: We consider three cases. In the first one, q and q' are both in M_{α_i}. Since only transitions from M_{α_1} into M_{α_2} are possible but not backwards, a path inducing $q \overset{\varepsilon}{\Rightarrow}_X q'$ has to stay in M_{α_i} the whole time. The claim then follows from the induction hypothesis. In the second case, q is in M_{α_1} and q' is in M_{α_2}. A path from q to q' has to transition through $q_{f,1}$ and $q_{0,2}$ to be able to switch automata, thus we have $q \overset{\varepsilon}{\Rightarrow}_X q'$ iff $q \overset{\varepsilon}{\Rightarrow}_Y q_{f,1}$ and $q_{0,2} \overset{\varepsilon}{\Rightarrow}_Z q'$ for some Y, Z with $X = Y \cup Z$. Since for state pairs inside one of the subautomata it does not matter whether $\overset{\varepsilon}{\Rightarrow}_X$ was constructed from M_α or M_{α_i}, the claim follows from the induction hypothesis. In the last case, q is in M_{α_2} and q' is in M_{α_1}. Since q' is not reachable from q, $q \overset{\varepsilon}{\Rightarrow}_X q'$ can not hold for any X and the claim follows immediately.

Case $\alpha = \alpha_1^*$: We consider five cases. In the first case, we have $q, q' \in Q_1$. Since the backwards edge that was added during the construction is ignored for this lemma, no new paths from q to q' are added compared to M_{α_1}. Therefore the claim follows from the induction hypothesis. In the second case, we have $q = q_0$ and $q' = q_f$. The claim follows immediately from the fact that there is an ε-edge in between the two states. The third case, where $q = q_0$ and $q' \in Q_1$, and the fourth case, where $q \in Q_1$ and $q' = q_f$ work in a similar way by considering the added ε edges between old and new starting and final states and by using the induction hypothesis. In the last case, we have $q = q_f$ and $q' = q_0$. The claim holds since the backwards edge connecting the two states is ignored for this lemma.

Case $\alpha = \psi_k?$: We consider the different cases how q' can be reached by ε-transitions from q in M_α. In the first case, $q = q'$ with $q \neq q_1$, we have trivial reachability without encountering a state marking. Here, the claim is established as in previous cases. In the second case, where $q = q_i, q' = q_j$ with $i > j$, q' is not reachable from q since the ε-transitions only point in the other direction. The claim is again established as in previous cases. In all other cases, q' can be reached from q with ε-transitions, but only with encountering the state marking in q_1. Since this is the only state marking in M_α, we have $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv v_i = dp(q, q', \alpha)$, establishing the claim. ◀

▶ **Theorem 21.** *We have $\varepsilon p(q, q', \alpha) \equiv \bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\}$ for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from the automaton M_α.*

Proof. Compared to the claim made about dp in Lemma 20, backwards edges must now be considered in our claim about εp. The central observation is that the contribution of all ε-paths from q to q' is already captured by two particular types of ε-paths: either by going from q to q' directly without taking a backwards edge, or by taking only the backwards edge from the construction of $M_{\bar{\alpha}}$ exactly once (where $\bar{\alpha}$ is the innermost *-construction

that contains both q and q'). As was shown in Lemma 20, the first type is captured by $dp(q, q', \alpha)$. It is also straightforward to see from Lemma 20 that the second type is captured by $dp(q, q_{f,\bar{\alpha}}, \alpha) \wedge dp(q_{0,\bar{\alpha}}, q', \alpha)$.

We now argue that further backwards edges need not be considered and thus all paths are subsumed by these two cases. In order to use a backwards edge outside of $M_{\bar{\alpha}}$, a path has to leave $M_{\bar{\alpha}}$ via $q_{f,\bar{\alpha}}$ and finally reenter it via $q_{0,\bar{\alpha}}$. The contribution of such paths to the disjunction is subsumed by paths taking the backwards edge from $q_{f,\bar{\alpha}}$ to $q_{0,\bar{\alpha}}$ directly. Backwards edges on the paths from q to $q_{f,\bar{\alpha}}$ or on the paths from $q_{0,\bar{\alpha}}$ to q' on the other hand that originate in a final state q_f of some subautomaton only lead to paths that later visit q_f a second time. Thus their contribution is again subsumed by the contribution of the path where loops from q_f to itself are cut out. ◀