Bounded Model Checking for Asynchronous Hyperproperties[*]

Tzu-Han Hsu[1], Borzoo Bonakdarpour[1](✉), Bernd Finkbeiner[2], and César Sánchez[3]

[1] Michigan State University, East Lansing, MI, USA {tzuhan,borzoo}@msu.edu
[2] CISPA Helmholtz Center, Saarbrücken, Germany finkbeiner@cispa.de
[3] IMDEA Software Institute, Madrid, Spain cesar.sanchez@imdea.org

Abstract. Many types of attacks on confidentiality stem from the nondeterministic nature of the environment that computer programs operate in. We focus on verification of confidentiality in nondeterministic environments by reasoning about *asynchronous hyperproperties*. We generalize the temporal logic A-HLTL to allow nested *trajectory* quantification, where a trajectory determines how different execution traces may advance and stutter. We propose a bounded model checking algorithm for A-HLTL based on QBF-solving for a fragment of A-HLTL and evaluate it by various case studies on concurrent programs, scheduling attacks, compiler optimization, speculative execution, and cache timing attacks. We also rigorously analyze the complexity of model checking A-HLTL.

1 Introduction

Motivation. Consider the concurrent program [10] shown in Fig. 1, where `h` is a secret variable, and `await` command is a conditional critical region. This program should satisfy the following information-flow policy: "Any sequences of observable outputs produced by an interleaving should be reproducible by some other interleaving for a different value of `h`". If this is the case, then an attacker cannot successfully guess the value of `h` from the sequence of observable outputs of the `print()` statements. For example, Fig. 2 shows how one can align two interleavings of threads `T1` and `T2` with respect to the observable sequence of outputs `'abcd'`, given two different values of secret `h`. Let us call such an alignment a *trajectory* (illustrated by the sequence of dashed lines). However, if

```
Thread T1() {
  await sem>0 then
    sem = sem - 1;
    print('a');
    v = v+1;
    print('b');
    sem = sem + 1;
  }

Thread T2 (){
  print('c');
  if h then
    await sem>0 then
      sem = sem - 1;
      v = v+2;
      sem = sem + 1;
  else
    skip;
  print('d');
}
```

Fig. 1: `T1` and `T2` leak the value of `h`.

[*] This research has been partially supported by the United States NSF SaTC Award 2100989, by the Madrid Regional Gov. Project BLOQUES-CM (S2018/TCS-4339), by Project PRODIGY (TED2021-132464B-I00) funded by MCIN/AEI/10.13039/501100011033/ and the EU NextGenerationEU/PRTR, by the German Research Foundation (DFG) as part of TRR 248 (389792660), and by the European Research Council (ERC) Grant HYPER (101055412)

thread `T1` holds the semaphore and executes the critical region as an atomic operation. Then, output `‘acdb’` arising due to concurrent execution of threads `T1` and `T2` reveals the value of `h` as 0, as the same output cannot be reproduced when `h=1`. Thus, the program in Fig. 1 violates the above policy.

The above policy is an example of a *hyperproperty* [5]; i.e., a set of sets of execution traces. In addition to information-flow requirements, hyperproperties can express other complex requirements such as linearizability [12] and control conditions in cyber-physical systems such as robustness and sensitivity. The temporal logic A-HLTL [1] can express hyperproperties whose sets of traces advance at different speeds, allowing stuttering steps. For example, the above policy can be expressed in A-HLTL by the following formula: $\varphi_{\mathsf{NI}} = \forall \pi.\exists \pi'.\mathsf{E}\tau.(\mathsf{h}_{\pi,\tau} \neq \mathsf{h}_{\pi',\tau}) \wedge \Box(\mathsf{obs}_{\pi,\tau} = \mathsf{obs}_{\pi',\tau})$, where obs denotes the output observations, meaning that for all executions (i.e., interleavings) π, there should exist another execution π' and a trajectory τ, such that π and π' start from different values of `h` and τ can align all the observations along π and π' (see Fig. 2). A-HLTL can reason about *one* source of *nondeterminism* by the scheduler in the system that may lead to information leak. Indeed, the model checking algorithms proposed in [1] can discover the bug in the program in Fig. 1.

```
Thread T1 (){
  while (true){
    await sem>0 then
      sem = sem - 1;
      print('a');
      v = v+1;
      print('b');
      sem = sem + 1;
  }
}

Thread T2(){
  while (true)
    h = read(Channel1);
}

Thread T3(){
  while (true){
    print('c');
    if (h == 1) then
      await sem>0 then
        sem = sem - 1;
        v = v+2;
        sem = sem + 1;
    else
      skip;
    print('d');
  }
}

Thread T4(){
  while (true)
    l = read(Channel2);
}
```

Fig. 3: `T1` and `T2` receive inputs from asynch. channels read by `T3` and `T4`.

Now, consider a more complex version of the same program shown in Fig. 3 inspired by modern programming languages such as `Go` and `P` that allow CSP-style concurrency. Here, new threads `T3` and `T4` read the values of secret input `h` and public input `l` from two asynchronous channels, rendering two different sources of nondeterminism: (1) the scheduler that results in different interleavings, and (2) data availability in the channels. This, in turn, means formula φ_{NI} no longer captures the following specification of the program, which should be:

> *“Any sequence of observable outputs produced by an interleaving should be reproducible by some other interleaving such that for all alignments of public inputs, there exists an alignment of the public outputs”.*

Satisfaction of this policy (not expressible in A-HLTL as proposed in [1]) prohibits an attacker from successfully determining the sequence of values of `h`.

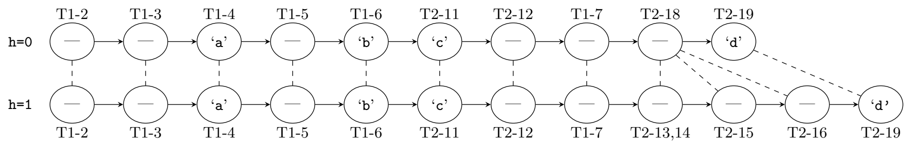


Fig. 2: Two secure interleavings for the program in Fig. 1

Contributions. In this paper, we strive for a general logic-based approach that enables model checking of a rich set of asynchronous hyperproperties. To this end, we concentrate on A-HLTL model checking for programs subject to multiple sources of nondeterminism. Our first contribution is a generalization of A-HLTL that allows nested *trajectory* quantification. For example, the above policy requires reasoning about two different trajectories that cannot be composed into one since their sources of nondeterminism are different. This observation motivates the need for enriching A-HLTL with the tools to quantify over trajectories. This generalization enables expressing policies such as follows:

$$\varphi_{\mathsf{NI}_{\mathsf{nd}}} = \forall\pi.\exists\pi'.\mathsf{A}\tau.\mathsf{E}\tau'.\big(\Diamond(\mathsf{h}_{\pi,\tau} \neq \mathsf{h}_{\pi',\tau}) \wedge \Box\,(\mathsf{l}_{\pi,\tau} = \mathsf{l}_{\pi',\tau})\big) \rightarrow \Box(\mathsf{obs}_{\pi,\tau'} = \mathsf{obs}_{\pi',\tau'}),$$

where A and E denote the universal (res., existential) trajectory quantifiers.

Our second contribution is a *bounded model checking* (BMC) algorithm for a fragment of the extended A-HLTL that allows an arbitrary number of trace quantifier alternations and up to one trajectory quantifier alternation. Following [15], we propose two bounded semantics (called *optimistic* and *pessimistic*) for A-HLTL based on the satisfaction of eventualities. We introduce a reduction to the satisfiability problem for quantified Boolean formulas (QBF) and prove that our translation provides decision procedures for A-HLTL BMC for *terminating systems*, i.e., those whose Kripke structure is acyclic. Our focus on terminating programs is due to the general undecidability of A-HLTL model checking [1]. As in the classic BMC for LTL, the power of our technique is in hunting bugs that are often in the shallow parts of reachable states.

Our third contribution is rigorous complexity analysis of A-HLTL model checking for terminating programs (see Table 1). We show that for formulas with only one trajectory quantifier the complexity is aligned with that of classic synchronous semantics of HyperLTL [4]. However, the complexity of A-HLTL model checking with multiple trajectory quantifiers is one step higher than HyperLTL model checking in the polynomial hierarchy. An interesting observation here is that the complexity of model checking a formula with two existential trajectory quantifiers is one step higher than one with only one existential quantifier although the plurality of the quantifiers does not change. Generally speaking, A-HLTL model checking for terminating programs remains in PSPACE.

Multiple Traces – Single Trajectory		
$\exists^{+}\mathsf{E}$ / $\forall^{+}\mathsf{A}$	NL-complete (Theorem 2)	
$[\exists(\exists/\forall)^{+}(\mathsf{A}/\mathsf{E})]^{k}$	Σ_{k}^{p}-complete	Thm 3
$[\forall(\exists/\forall)^{+}(\mathsf{E}/\mathsf{A})]^{k}$	Π_{k}^{p}-complete	
Multiple Traces – Multiple Trajectories		
$[\exists(\exists/\forall)^{+}(\mathsf{E}^{+}\mathsf{E})]^{k}$	Σ_{k+1}^{p}-complete	Thm 4
$[\forall(\forall/\exists)^{+}(\mathsf{A}^{+}\mathsf{A})]^{k}$	Π_{k+1}^{p}-complete	
$[\exists(\exists/\forall)^{+}\mathsf{A}^{+}\mathsf{E}^{+}]^{k}$	Σ_{k+1}^{p}-complete	Thm 5
$[\forall(\forall/\exists)^{+}\mathsf{E}^{+}\mathsf{A}^{+}]^{k}$	Π_{k+1}^{p}-complete	
A-HLTL	PSPACE	

Table 1: A-HLTL model checking complexity for acyclic models.

Finally, we have implemented our BMC technique. We evaluate our implementation on verification of four case studies: (1) information-flow security in concurrent programs, (2) information leak in speculative executions, (3) preservation of security in compiler optimization, and (4) cache-based timing attacks. These case studies exhibit a proof of concept for the highly intricate nature of information-flow requirements and how our foundational theoretical results handle them.

Related Work. The concept of hyperproperties is due to Clarkson and Schneider [5]. HyperLTL [4] and A-HLTL are currently the only logics for which practical model checking algorithms are known [8,7,15,1]. For HyperLTL, the algorithms have been implemented in the model checkers MCHyper and bounded model checker HyperQB [14]. HyperLTL is limited to synchronous hyperproperties. The A-HLTL model checking problem is known to be undecidable in general [1]. However, decidable fragments that can express observational determinism, noninterference, and linearizability have been identified. This paper generalizes A-HLTL by allowing nested trajectory quantifiers and due to the general undecidability result focuses on terminating programs.

FOL[E] [6] can express a limited form of asynchronous hyperproperties. As shown in [6], FOL[E] is subsumed by HyperLTL with additional quantification over predicates. For $S1S[E]$ and H_μ, the model checking problem is in general undecidable; for H_μ, two fragments, the k-synchronous, k-context bounded fragments, have been identified for which model checking remains decidable [11]. Other logical extensions of HyperLTL with asynchronous capabilities are studied in [3], including their decidable fragments, but their model checking problems have not been implemented and the relative expressive power with respect to other asynchronous formalisms has not been studied.

2 Extended Asynchronous HyperLTL

Preliminaries. Given a natural number $k \in \mathbb{N}_0$, we use $[k]$ for the set $\{0, \ldots, k\}$. Let AP be a set of *atomic propositions* and $\Sigma = 2^{\mathsf{AP}}$ be the *alphabet*, where we call each element of Σ a *letter*. A *trace* is an infinite sequence $\sigma = a_0 a_1 \cdots$ of letters from Σ. We denote the set of all infinite traces by Σ^ω. We use $\sigma(i)$ for a_i and σ^i for the suffix $a_i a_{i+1} \cdots$. A *pointed trace* is a pair (σ, p), where $p \in \mathbb{N}_0$ is a natural number (called the *pointer*). Pointed traces allow to traverse a trace by moving the pointer. Given a pointed trace (σ, p) and $n > 0$, we use $(\sigma, p) + n$ to denote the resulting trace $(\sigma, p + n)$. We denote the set of all pointed traces by $\mathsf{PTR} = \{(\sigma, p) \mid \sigma \in \Sigma^\omega \text{ and } p \in \mathbb{N}_0\}$.

A *Kripke structure* is a tuple $\mathcal{K} = \langle S, s_{init}, \delta, L \rangle$, where S is a set of states, $s_{init} \in S$ is the initial state, $\delta \subseteq S \times S$ is a transition relation, and $L : S \to \Sigma$ is a labeling function on the states of $\mathcal{K}$. We require that for each $s \in S$, there exists $s' \in S$, such that $(s, s') \in \delta$. □

A *path* of a Kripke structure $\mathcal{K}$ is an infinite sequence of states $s(0)s(1) \cdots \in S^\omega$, such that $s(0) = s_{init}$ and $(s(i), s(i+1)) \in \delta$, for all $i \geq 0$. A trace of $\mathcal{K}$ is a sequence $\sigma(0)\sigma(1)\sigma(2) \cdots \in \Sigma^\omega$, such that there exists a path $s(0)s(1) \cdots \in S^\omega$ with $\sigma(i) = L(s(i))$ for all $i \geq 0$. We denote by $\mathsf{Traces}(\mathcal{K}, s)$ the set of all traces of $\mathcal{K}$ with paths that start in state $s \in S$.

The directed graph $\mathcal{F} = \langle S, \delta \rangle$ is called the *Kripke frame* of the Kripke structure $\mathcal{K}$. A *loop* in $\mathcal{F}$ is a finite sequence $s_0 s_1 \cdots s_n$, such that $(s_i, s_{i+1}) \in \delta$, for all $0 \leq i < n$, and $(s_n, s_0) \in \delta$. We call a Kripke frame *acyclic*, if the only loops are self-loops on terminal states, i.e., on states that have no other outgoing transition. Acyclic Kripke structures model terminating programs.

Extended A-HLTL. The syntax of extended $\mathsf{A\text{-}HLTL}$ is:

$$\varphi ::= \exists \pi.\varphi \mid \forall \pi.\varphi \mid \mathsf{E}\tau.\varphi \mid \mathsf{A}\tau.\varphi \mid \psi$$
$$\psi ::= \mathit{true} \mid a_{\pi,\tau} \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \psi_1 \; \mathcal{U} \; \psi_2 \mid \psi_1 \, \mathcal{R} \, \psi_2$$

where $a \in \mathsf{AP}$, π is a trace variable from an infinite supply $\mathcal{V}$ of trace variables, τ is a *trajectory variable* from an infinite supply $\mathcal{J}$ of trajectory variables (see formula $\varphi_{\mathsf{NI}_{\mathsf{nd}}}$ in Section 1 for an example). The intended meaning of $a_{\pi,\tau}$ is that proposition $a \in \mathsf{AP}$ holds in the current time in trace π and *trajectory* τ (explained later). Trace (respectively, trajectory) quantifiers $\exists\pi$ and $\forall\pi$ (respectively, $\mathsf{E}\tau$ and $\mathsf{A}\tau$) allow reasoning simultaneously about different traces (respectively, trajectories). The intended meaning of E is that there is a trajectory that gives an interpretation of the relative passage of time between the traces for which the temporal formula that relates the traces is satisfied. Dually, A means that all trajectories satisfy the inner formula. Given an $\mathsf{A\text{-}HLTL}$ formula φ, we use $\mathsf{Paths}(\varphi)$ (respectively, $\mathsf{Trajs}(\varphi)$) for the set of trace (respectively, trajectory) variables quantified in φ. A formula φ is *well-formed* if for all atoms $a_{\pi,\tau}$ in φ, π and τ are quantified in φ (i.e., $\tau \in \mathsf{Trajs}(\varphi)$ and $\pi \in \mathsf{Paths}(\varphi)$) and no trajectory/trace variable is quantified twice in φ. We use the usual syntactic sugar $\mathit{false} \triangleq \neg\mathit{true}$, and $\Diamond\varphi \triangleq \mathit{true}\,\mathcal{U}\varphi$, $\varphi_1 \rightarrow \varphi_2 \triangleq \neg\varphi_1 \vee \varphi_2$, and $\Box\varphi \triangleq \neg\Diamond\neg\varphi$, etc. We choose to add $\mathcal{R}$ (release) and $\wedge$ to the logic to enable negation normal form (NNF). As our BMC algorithm cannot handle formulas that are not invariant under stuttering, the *next* operator is not included.

Semantics. A *trajectory* $t : t(0)t(1)t(2)\cdots$ for a formula φ is an infinite sequence of subsets of $\mathsf{Paths}(\varphi)$, i.e., each $t_i \subseteq \mathsf{Paths}(\varphi)$, for all $i \geq 0$. Essentially, in each step of the trajectory one or more of the traces make progress or all may stutter. A trajectory is *fair* for a trace variable $\pi \in \mathsf{Paths}(\varphi)$ if there are infinitely many positions j such that $\pi \in t(j)$. A trajectory is fair if it is fair for all trace variables in $\mathsf{Paths}(\varphi)$. Given a trajectory t, by t^i, we mean the suffix $t(i)t(i+1)\cdots$. Furthermore, for a set of trace variables $\mathcal{V}$, we use $\mathsf{TRJ}_{\mathcal{V}}$ for the set of all fair trajectories for indices from $\mathcal{V}$. We also use a *trajectory assignment* $\Gamma : \mathsf{Trajs}(\varphi) \rightharpoonup \mathsf{TRJ}_{Dom(\Gamma)}$, where $Dom(\Gamma)$ is the subset of $\mathsf{Trajs}(\varphi)$ for which Γ is defined. Given a trajectory assignment Γ, a trajectory variable τ, and a trajectory t, we denote by $\Gamma[\tau \mapsto t]$ the assignment that coincides with Γ for every trajectory variable except for τ, which is mapped to t.

For the semantics of extended $\mathsf{A\text{-}HLTL}$, we need asynchronous trace assignments $\Pi : \mathsf{Paths}(\varphi) \times \mathsf{Trajs}(\varphi) \rightarrow T \times \mathbb{N}$ which map each pair (π, τ) formed by a path variable and trajectory variable into a pointed trace. Given (Π, Γ) where Π is an asynchronous trace assignment and Γ a trajectory assignment, we use $(\Pi, \Gamma) + 1$ for the successor of (Π, Γ) defined as (Π', Γ') where $\Gamma'(\tau) = \Gamma(\tau)^1$, and $\Pi'(\pi, \tau) = \Pi(\pi, \tau) + 1$ if $\pi \in \Gamma(\tau)(0)$ and $\Pi'(\pi, \tau) = \Pi(\pi, \tau)$ otherwise. Note that Π can assign the same π to different pointed traces depending on the trajectory. We use $(\Pi, \Gamma) + k$ as the k-th successor of (Π, Γ). Given an asynchronous trace assignment Π, a trace variable π, a trajectory variable τ a trace σ, and a pointer p, we denote by $\Pi[(\pi, \tau) \mapsto (\sigma, p)]$ the assignment that coincides

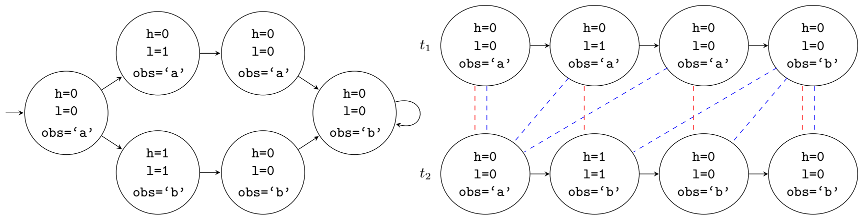


Fig. 4: Kripke structure $\mathcal{K}$ and traces t_1 and t_2 of $\mathcal{K}$, $\mathcal{K} \models \varphi_{\mathsf{NI}_{\mathsf{nd}}}$ but $\mathcal{K} \not\models \varphi_{\mathsf{NI}}$.

with Π for every pair except for (π, τ), which is mapped to (σ, p). The satisfaction of an A-HLTL formula φ over a trace assignment Π, a trajectory assignment Γ, and a set of traces T is defined as follows (we omit $\neg$, $\wedge$ and $\vee$ which are standard):

$$
\begin{array}{lll}
(\Pi, \Gamma) \models_T \exists \pi.\varphi & \text{iff} & \text{for some } \sigma \in T : \\
& & (\Pi[(\pi,\tau) \mapsto (\sigma, 0)], \Gamma) \models_T \varphi \text{ for all } \tau \\
(\Pi, \Gamma) \models_T \forall \pi.\varphi & \text{iff} & \text{for all } \sigma \in T : \\
& & (\Pi[(\pi,\tau) \mapsto (\sigma, 0)], \Gamma) \models_T \varphi \text{ for all } \tau \\
(\Pi, \Gamma) \models_T \mathsf{E}\tau.\psi & \text{iff} & \text{for some } t \in \mathsf{TRJ}_{Dom(\Pi)} : (\Pi, \Gamma[\tau \mapsto t]) \models \psi \\
(\Pi, \Gamma) \models_T \mathsf{A}\tau.\psi & \text{iff} & \text{for all } t \in \mathsf{TRJ}_{Dom(\Pi)} (\Pi, \Gamma[\tau \mapsto t]) \models \psi \\
(\Pi, \Gamma) \models \quad a_{\pi,\tau} & \text{iff} & a \in \sigma(n) \text{ where } (\sigma, n) = \Pi(\pi, \tau) \\
(\Pi, \Gamma) \models \quad \psi_1 \,\mathcal{U}\, \psi_2 & \text{iff} & \text{for some } i \geq 0 : (\Pi, \Gamma) + i \models \psi_2 \text{ and} \\
& & \text{for all } j < i : (\Pi, \Gamma) + j \models \psi_1 \\
(\Pi, \Gamma) \models \quad \psi_1 \,\mathcal{R}\, \psi_2 & \text{iff} & \text{for all } i \geq 0 : (\Pi, \Gamma) + i \models \psi_2 \text{, or} \\
& & \text{for some } i \geq 0 : (\Pi, \Gamma) + i \models \psi_1 \text{ and} \\
& & \quad \text{for all } j \leq i : (\Pi, \Gamma) + j \models \psi_2
\end{array}
$$

We say that a set T of traces satisfies a sentence φ, denoted by $T \models \varphi$, if $(\Pi_\emptyset, \Gamma_\emptyset) \models_T \varphi$. We say that a Kripke structure $\mathcal{K}$ satisfies an A-HLTL formula φ (and write $\mathcal{K} \models \varphi$) if and only if we have $\mathsf{Traces}(\mathcal{K}, S_{init}) \models \varphi$. An example is illustrated in Fig. 4.

3 Bounded Model Checking for A-HLTL

We first introduce the bounded semantics of A-HLTL (for at most one trajectory quantifier alternation but arbitrary trace quantifiers) which will be used to generate queries to a QBF solver to aid solving the BMC problem. The main result of this section is Theorem 1 which provides decision procedures for model checking A-HLTL for terminating systems.

3.1 Bounded Semantics of A-HLTL

The bounded semantics corresponds to the exploration of the system up to a certain bound. In our case, we will consider two bounds k and m (with $k \leq m$).

The bound k corresponds to the *maximum depth* of the unrolling of the Kripke structures and m is the *bound on trajectories length*. We start by introducing some auxiliary functions and predicates, for a given trace assignment and (Π, Γ). First, the family of functions $pos_{\pi,\tau} : \{0 \ldots m\} \to \mathbb{N}$. The meaning of $pos_{\pi,\tau}(i)$ provides how many times π has been selected in $\{\tau(0), \ldots, \tau(i)\}$. We assume that Kripke structures are equipped with an atomic proposition *halt* (one per trace variable π) which encodes whether the state is a halting state. Given (Π, Γ) we consider the predicate *halted* that holds whenever for all π and τ, $halt \in \sigma(j)$ for $(\sigma, j) = \Pi(\pi, \tau)$. In this case we write $(\Pi, \Gamma, n) \models halted$.

We define two bounded semantics which only differ in how they inspect beyond the (k, m) bounds: $\models^{hpes}_{k,m}$, called the *halting pessimistic semantics* and $\models^{hopt}_{k,m}$, called the *halting optimistic semantics*. We start by defining the bounded semantics of the quantifiers.

$$(\Pi, \Gamma, 0) \models_{k,m} \exists \pi.\ \psi \quad \text{iff} \quad \text{there is a } \sigma \in T_\pi\text{, such that for all } \tau \; (\Pi[(\pi,\tau) \to (\sigma,0)], \Gamma, 0) \models_{k,m} \psi \tag{1}$$

$$(\Pi, \Gamma, 0) \models_{k,m} \forall \pi.\ \psi \quad \text{iff} \quad \text{for all } \sigma \in T_\pi,\ \text{for all } \tau : (\Pi[(\pi,\tau) \to (\sigma,0)], \Gamma, 0) \models_{k,m} \psi \tag{2}$$

$$(\Pi, \Gamma, 0) \models_{k,m} \mathsf{E}\tau.\ \psi \quad \text{iff} \quad \text{there is a } t \in \mathsf{TRJ}_{Dom(\Pi)} : (\Pi, \Gamma[\tau \to t], 0) \models_{k,m} \psi \tag{3}$$

$$(\Pi, \Gamma, 0) \models_{k,m} \mathsf{A}\tau.\ \psi \quad \text{iff} \quad \text{for all } t \in \mathsf{TRJ}_{Dom(\Pi)} : (\Pi, \Gamma[\tau \to t], 0) \models_{k,m} \psi \tag{4}$$

For the Boolean operators, for $i \leq m$:

$$(\Pi, \Gamma, i) \models_{k,m} \texttt{true} \tag{5}$$

$$(\Pi, \Gamma, i) \models_{k,m} a_{\pi,\tau} \quad \text{iff} \quad a \in (\sigma, j) \text{ where } (\sigma, j) = \Pi(\pi,\tau)(i) \text{ and } j \leq k \tag{6}$$

$$(\Pi, \Gamma, i) \models_{k,m} \neg a_{\pi,\tau} \quad \text{iff} \quad a \notin (\sigma, j) \text{ where } (\sigma, j) = \Pi(\pi,\tau)(i) \text{ and } j \leq k \tag{7}$$

$$(\Pi, \Gamma, i) \models_{k,m} \psi_1 \vee \psi_2 \quad \text{iff} \quad (\Pi, \Gamma, i) \models_{k,m} \psi_1 \text{ or } (\Pi, \Gamma, i) \models_{k,m} \psi_2 \tag{8}$$

$$(\Pi, \Gamma, i) \models_{k,m} \psi_1 \wedge \psi_2 \quad \text{iff} \quad (\Pi, \Gamma, i) \models_{k,m} \psi_1 \text{ and } (\Pi, \Gamma, i) \models_{k,m} \psi_2 \tag{9}$$

For the temporal operators, we must consider the cases of falling of the paths (beyond k) and falling of the traces (beyond m). We define the predicate *off* which holds for (Π, Γ, i) if for some (π, τ), $pos_{\pi,\tau}(i) > k$ and $halt_\pi \notin \sigma(k)$ where σ is the trace assigned to π. Note that *halted* implies that *off* does not hold because all paths (including those at k or beyond) satisfy *halt*.

We define two semantics that differ on how to interpret when the end of the unfolding of the traces and trajectories is reached. The *halting pessimistic* semantics, denoted by $\models^{hpes}_{k,m}$ take (1)-(9) above and add (10)-(13) together with $(\Pi, \Gamma, i) \not\models_{k,m}$ *off*. Rules (10) and (11) define the semantics of the temporal operators for the case $i < m$, that is, before the end of the unrolling of the trajectories (recall that we do not consider $\bigcirc$):

$$(\Pi, \Gamma, i) \models_{k,m} \quad \psi_1 \,\mathcal{U}\, \psi_2 \quad \text{iff} \quad (\Pi, \Gamma, i) \models_{k,m} \psi_2\text{, or } (\Pi, \Gamma, i) \models_{k,m} \psi_1\text{, and } (\Pi, \Gamma, i) + 1 \models_{k,m} \psi_1 \,\mathcal{U}\, \psi_2 \tag{10}$$

$$(\Pi, \Gamma, i) \models_{k,m} \quad \psi_1 \mathcal{R} \psi_2 \quad \text{iff} \quad (\Pi, \Gamma, i) \models_{k,m} \psi_2, \text{ and } (\Pi, \Gamma, i) \models_{k,m} \psi_1, \text{ or } (\Pi, \Gamma, i) + 1 \models_{k,m} \psi_1 \mathcal{R} \psi_2 \tag{11}$$

For the case of $i = m$, that is, at the bound of the trajectory:

$$(\Pi, \Gamma, m) \models^{hpes}_{k,m} \quad \psi_1 \mathcal{U} \psi_2 \quad \text{iff} \quad (\Pi, \Gamma, m) \models_{k,m} \psi_2 \tag{12}$$

$$(\Pi, \Gamma, m) \models^{hpes}_{k,m} \quad \psi_1 \mathcal{R} \psi_2 \quad \text{iff} \quad (\Pi, \Gamma, m) \models_{k,m} \psi_1 \wedge \psi_2, \text{ or } (\Pi, \Gamma, m) \models_{k,m} \mathit{halted} \wedge \psi_2 \tag{13}$$

The *halting optimistic* semantics, denoted by $\models^{hopt}_{k,m}$ take rules (1)-(11) and (12′)-(13′), but now if $(\Pi, \Gamma, i) \models^{hopt}_{k,m}$ *off* then $(\Pi, \Gamma, i) \models^{hopt}_{k,m} \varphi$ holds for every formula. Again, rules (10) and (11) define the semantics of the temporal operators for the case $i < m$. Then, for $i = m$:

$$(\Pi, \Gamma, m) \models^{hopt}_{k,m} \quad \psi_1 \mathcal{U} \psi_2 \quad \text{iff} \quad (\Pi, \Gamma, m) \models_{k,m} \psi_2, \text{ or } (\Pi, \Gamma, m) \not\models_{k,m} \mathit{halted} \wedge \psi_1 \tag{12′}$$

$$(\Pi, \Gamma, m) \models^{hopt}_{k,m} \quad \psi_1 \mathcal{R} \psi_2 \quad \text{iff} \quad (\Pi, \Gamma, m) \models_{k,m} \psi_2 \tag{13′}$$

Similar to [15] for the case of HyperLTL, the pessimistic semantics capture the case where we assume that pending eventualities will not become true in the future after the end of the trace (this is also assumed in LTL BMC). Dually, the optimistic semantics assume that all pending eventualities at the end of the trace will be fulfilled. Therefore, the following hold (proofs in [13]).

Lemma 1. *Let $k \leq k'$ and $m \leq m'$.*
1. *If $(\Pi, \Gamma, 0) \models^{hpes}_{k,m} \varphi$, then $(\Pi, \Gamma, 0) \models^{hpes}_{k',m'} \varphi$.*
2. *If $(\Pi, \Gamma, 0) \not\models^{hopt}_{k,m} \varphi$, then $(\Pi, \Gamma, 0) \not\models^{hopt}_{k',m'} \varphi$.*

Lemma 2. *The following hold for every k and m,*
1. *If $(\Pi, \Gamma, 0) \models^{hpes}_{k,m} \varphi$, then $(\Pi, \Gamma, 0) \models \varphi$.*
2. *If $(\Pi, \Gamma, 0) \not\models^{hopt}_{k,m} \varphi$, then $(\Pi, \Gamma, 0) \not\models \varphi$.*

3.2 From Bounded Semantics to QBF Solving

Let $\mathcal{K}$ be a Kripke structure and φ be an A-HLTL formula. Based on the bounded semantics introduced previously, our main approach is to generate a QBF query (with bounds k, m), which can use either the pessimistic or the optimistic semantics. We use $[\![\mathcal{K}, \varphi]\!]^{hpes}_{k,m}$ if the pessimistic semantics are used and $[\![\mathcal{K}, \varphi]\!]^{hopt}_{k,m}$ if the optimistic semantics are used. Our translations will satisfy that

(1) if $[\![\mathcal{K}, \varphi]\!]^{hpes}_{k,m}$ is SAT, then $\mathcal{K} \models \varphi$;
(2) if $[\![\mathcal{K}, \varphi]\!]^{hopt}_{k,m}$ is UNSAT, then $\mathcal{K} \not\models \varphi$;
(3) if the Kripke structure is unrolled to the diameter and the trajectories up to a maximum length (see below), then $[\![\mathcal{K}, \varphi]\!]^{hpes}_{k,m}$ is SAT if and only if $[\![\mathcal{K}, \varphi]\!]^{hopt}_{k,m}$ is SAT.

The first step to define $[\![\mathcal{K},\varphi]\!]^{hopt}_{k,m}$ and $[\![\mathcal{K},\varphi]\!]^{hpes}_{k,m}$ is to encode the unrolling of the models up-to a given depth k. For a path variable π corresponding to Kripke structure $\mathcal{K}$, we introduce $(k+1)$ copies $(x^0,\ldots,x^k)$ of the Boolean variables that define the state of $\mathcal{K}$ and use the initial condition I and the transition relation R of $\mathcal{K}$ to relate these variables. For example, for $k=3$, we unroll the transition relation up-to 3 as follows:

$$[\![\mathcal{K}]\!]_3 = I(x^0) \wedge R(x^0,x^1) \wedge R(x^1,x^2) \wedge R(x^2,x^3).$$

Encoding positions. For each trajectory variable τ and given the bound m on the unrolling of trajectories, we add $\mathsf{Paths}(\varphi) \times (m+1)$ variables $t^0_\pi \ldots t^m_\pi$, for each π. The intended meaning of t^j_π is that t^j_π is true whenever $\pi \in t(j)$, that is, when t dictates that π moves at time instant j. In order to encode sanity conditions on trajectories, that are crucial for completeness, it is necessary to introduce a family of variables that captures how much π has moved according to τ after j steps. There is a variable *pos* for each trace variable π, each trajectory τ and each $i \leq k$ and $j \leq m$. We represent this variable by $pos^{i,j}_{\pi,\tau}$. The intention is that *pos* is true whenever after j steps trajectory τ has dictated that trace π progresses precisely i times. Fig. 5 shows encodings t^j_π and $pos^{i,j}_{\pi,\tau}$ for the traces w.r.t. the blue trajectory, τ' in Fig. 4. We will use the auxiliary definitions (for $i \in \{0\ldots k\}$ and $j \in \{0\ldots m\}$) to force that the path π has moved to position i after j moves from the trajectory and that π has not fallen off the trace (and does not change position when the paths fall off the trace):

Encodings of t^j_π and $t^j_{\pi'}$:

$[t^0_\pi, t^1_\pi, t^2_\pi, t^3_\pi, t^4_\pi, t^5_\pi, t^6_\pi]$
$[t^0_{\pi'}, t^1_{\pi'}, t^2_{\pi'}, t^3_{\pi'}, t^4_{\pi'}, t^5_{\pi'}, t^6_{\pi'}]$

Encodings of $pos^{i,j}_{\pi,\tau'}$ and $pos^{i,j}_{\pi',\tau'}$

$[pos^{0,0}_{\pi,\tau'}, pos^{0,1}_{\pi,\tau'}, pos^{0,2}_{\pi,\tau'}, pos^{0,3}_{\pi,\tau'},$
$pos^{1,1}_{\pi,\tau'}, pos^{1,2}_{\pi,\tau'}, pos^{1,3}_{\pi,\tau'}, pos^{1,4}_{\pi,\tau'},$
$pos^{2,2}_{\pi,\tau'}, pos^{2,3}_{\pi,\tau'}, pos^{2,4}_{\pi,\tau'}, pos^{2,5}_{\pi,\tau'},$
$pos^{3,3}_{\pi,\tau'}, pos^{3,4}_{\pi,\tau'}, pos^{3,5}_{\pi,\tau'}, pos^{3,6}_{\pi,\tau'}]$

$[pos^{0,0}_{\pi',\tau'}, pos^{0,1}_{\pi',\tau'}, pos^{0,2}_{\pi',\tau'}, pos^{0,3}_{\pi',\tau'},$
$pos^{1,1}_{\pi',\tau'}, pos^{1,2}_{\pi',\tau'}, pos^{1,3}_{\pi',\tau'}, pos^{1,4}_{\pi',\tau'},$
$pos^{2,2}_{\pi',\tau'}, pos^{2,3}_{\pi',\tau'}, pos^{2,4}_{\pi',\tau'}, pos^{2,5}_{\pi',\tau'},$
$pos^{3,3}_{\pi',\tau'}, pos^{3,4}_{\pi',\tau'}, pos^{3,5}_{\pi',\tau'}, pos^{3,6}_{\pi',\tau'}]$

Fig. 5: Variables for encodings of the blue trajectory in Fig. 4, where green variables are *true* and gray variables are *false*.

$$setpos^{i,j}_{\pi,\tau} \stackrel{\text{def}}{=} pos^{i,j}_{\pi,\tau} \wedge \bigwedge_{n\in\{0..k\}\setminus\{i\}} \neg pos^{n,j}_{\pi,\tau} \wedge \neg \mathit{off}^{j}_{\pi,\tau}$$

$$nopos^{j}_{\pi,\tau} \stackrel{\text{def}}{=} \mathit{off}^{j}_{\pi,\tau} \wedge \bigwedge_{n\in\{0..k\}} \neg pos^{n,j}_{\pi,\tau}$$

Initially, $I_{pos} \stackrel{\text{def}}{=} \bigwedge_{\pi,\tau} setpos^{0,0}_{\pi,\tau}$, where $\pi \in \mathsf{Traces}(\varphi)$ and $\tau \in \mathsf{TRJ}_{Dom(\Pi)}$. I_{pos} captures that all paths are initially at position 0. Then, for every step $j \in \{0\ldots m\}$, the following formulas relate the values of *pos* and *off*, depending on whether trajectory τ moves path π or not (and on whether π has reached the end k or halted):

$$step^{j}_{\pi,\tau} \stackrel{\text{def}}{=} \bigwedge_{i\in\{0..k-1\}} \left(pos^{i,j}_{\pi,\tau} \wedge t^j_\pi \rightarrow setpos^{i+1,j+1}_{\pi,\tau}\right)$$

$$\mathit{stutters}^{j}_{\pi,\tau} \stackrel{\text{def}}{=} \bigwedge_{i\in\{0..k\}} \left(\mathit{pos}^{i,j}_{\pi,\tau} \land \neg t^{j}_{\pi} \to \mathit{setpos}^{i,j+1}_{\pi,\tau}\right)$$

$$\mathit{ends}^{j}_{\pi,\tau} \stackrel{\text{def}}{=} (\mathit{pos}^{k,j}_{\pi,\tau} \land t^{j}_{\pi}) \to \left((\neg \mathit{halt}^{k}_{\pi} \to \mathit{nopos}^{j+1}_{\pi,\tau}) \land (\mathit{halt}^{k}_{\pi} \to \mathit{setpos}^{k,j+1}_{\pi,\tau})\right)$$

Then the following formula captures the correct assignment to the the *pos* variables, including the initial assignment:

$$\varphi_{pos} \stackrel{\text{def}}{=} I_{pos} \land \bigwedge_{j\in\{0..m\}} \bigwedge_{\pi,\tau} (\mathit{step}^{j}_{\pi,\tau} \land \mathit{stutters}^{j}_{\pi,\tau} \land \mathit{ends}^{j}_{\pi,\tau})$$

For example, Fig. 5 (w.r.t. Fig. 4) encodes the blue trajectory (τ') of π (i.e., t_1) and π' (i.e., t_2) as follows. First, for $j \in [0,3)$, it advances t_1 and stutters t_2. Therefore, $t^0_\pi, t^1_\pi, t^2_\pi$ are *true* and $t^0_{\pi'}, t^1_{\pi'}, t^2_{\pi'}$ are *false*. Notice that for *pos* encodings, the π position advances according to $\mathit{step}^{j}_{\pi,\tau'}$ (i.e., $\mathit{pos}^{0,0}_{\pi,\tau'}, \mathit{pos}^{1,1}_{\pi,\tau'}, \mathit{pos}^{2,2}_{\pi,\tau'}, \mathit{pos}^{3,3}_{\pi,\tau'}$); while π' stutters according to $\mathit{stutters}^{j}_{\pi',\tau'}$ (i.e., $\mathit{pos}^{0,0}_{\pi',\tau'}, \mathit{pos}^{0,1}_{\pi',\tau'}, \mathit{pos}^{0,2}_{\pi',\tau'}, \mathit{pos}^{0,3}_{\pi',\tau'}$). Then, for $j \in [3,5]$, it alternatively advances t_2 which makes $t^3_\pi, t^4_\pi, t^5_\pi$ *false* and $t^3_{\pi'}, t^4_{\pi'}, t^5_{\pi'}$ *true*. Similarly, the movements becomes $\mathit{pos}^{3,4}_{\pi,\tau'}, \mathit{pos}^{3,5}_{\pi,\tau'}, \mathit{pos}^{3,6}_{\pi,\tau'}$ and $\mathit{pos}^{1,4}_{\pi',\tau'}, \mathit{pos}^{2,5}_{\pi',\tau'}, \mathit{pos}^{3,6}_{\pi',\tau'}$. At the halting point (i.e., $j = k$), both trajectory trigger ends^{j} and do not advance anymore.

Encoding the inner LTL formula. We will use the following auxiliary predicates:

$$\mathit{halted}^{j} \stackrel{\text{def}}{=} \bigwedge_{\tau} \mathit{halted}^{j}_{\tau} \qquad\qquad \mathit{off}^{j} \stackrel{\text{def}}{=} \bigvee_{\pi,\tau} \mathit{off}^{j}_{\pi,\tau}$$

We now give the encoding for the inner temporal formulas for a fix unrolling k and m as follows. For the atomic and Boolean formulas, the following translations are performed for $j \in \{0 \dots m\}$.

$$[\![p_{\pi,\tau}]\!]^{j}_{k,m} := \bigvee_{i\in\{0..k\}} (\mathit{pos}^{i,j}_{\pi,\tau} \land p^{i}_{\pi}) \tag{14}$$
$$[\![\neg p_{\pi,\tau}]\!]^{j}_{k,m} := \bigvee_{i\in\{0..k\}} (\mathit{pos}^{i,j}_{\pi,\tau} \land \neg p^{i}_{\pi}) \tag{15}$$
$$[\![\psi_1 \lor \psi_2]\!]^{j}_{k,m} := [\![\psi_1]\!]^{j}_{k,m} \lor [\![\psi_2]\!]^{j}_{k,m} \tag{16}$$
$$[\![\psi_1 \land \psi_2]\!]^{j}_{k,m} := [\![\psi_1]\!]^{j}_{k,m} \land [\![\psi_2]\!]^{j}_{k,m} \tag{17}$$

The halting pessimistic semantics translation uses $[\![\cdot]\!]_{hpes}$, taking (14)-(17) and (18)-(21) below. For the temporal operators and $j < m$:

$$[\![\psi_1 \,\mathcal{U}\, \psi_2]\!]^{j}_{k,m} := \neg \mathit{off}^{j} \land \left([\![\psi_2]\!]^{j}_{k,m} \lor ([\![\psi_1]\!]^{j}_{k,m} \land [\![\psi_1 \,\mathcal{U}\, \psi_2]\!]^{j+1}_{k,m})\right) \tag{18}$$
$$[\![\psi_1 \,\mathcal{R}\, \psi_2]\!]^{j}_{k,m} := \neg \mathit{off}^{j} \land \left([\![\psi_2]\!]^{j}_{k,m} \land ([\![\psi_1]\!]^{j}_{k,m} \lor [\![\psi_1 \,\mathcal{R}\, \psi_2]\!]^{j+1}_{k,m})\right) \tag{19}$$

For $j = m$:

$$[\![\psi_1 \,\mathcal{U}\, \psi_2]\!]^{m}_{k,m} := [\![\psi_2]\!]^{m}_{k,m} \tag{20}$$
$$[\![\psi_1 \,\mathcal{R}\, \psi_2]\!]^{m}_{k,m} := \left([\![\psi_1]\!]^{m}_{k,m} \land [\![\psi_2]\!]^{m}_{k,m}\right) \lor \left(\mathit{halted}^{m} \land [\![\psi_2]\!]^{m}_{k,m}\right) \tag{21}$$

The halting optimistic semantics translation uses $[\![\cdot]\!]_{hopt}$, taking (14)-(17) and (18′)-(21′) as follows, For the temporal operators and $j < m$:

$$[\![\psi_1 \,\mathcal{U}\, \psi_2]\!]^j_{k,m} := \mathit{off}^j \lor \big([\![\psi_2]\!]^j_{k,m} \lor ([\![\psi_1]\!]^j_{k,m} \land [\![\psi_1 \,\mathcal{U}\, \psi_2]\!]^{j+1}_{k,m})\big) \tag{18′}$$
$$[\![\psi_1 \,\mathcal{R}\, \psi_2]\!]^j_{k,m} := \mathit{off}^j \lor \big([\![\psi_2]\!]^j_{k,m} \land ([\![\psi_1]\!]^j_{k,m} \lor [\![\psi_1 \,\mathcal{R}\, \psi_2]\!]^{j+1}_{k,m})\big) \tag{19′}$$

For $j = m$:

$$[\![\psi_1 \,\mathcal{U}\, \psi_2]\!]^m_{k,m} := [\![\psi_2]\!]^m_{k,m} \lor \big(\mathit{halted}^m \land [\![\psi_1]\!]^m_{k,m}\big) \tag{20′}$$
$$[\![\psi_1 \,\mathcal{R}\, \psi_2]\!]^m_{k,m} := [\![\psi_2]\!]^m_{k,m} \tag{21′}$$

Combining the encodings. Let φ be a A-HLTL formula of the form $\varphi = \mathbb{Q}_A\pi_A.\dots.\mathbb{Q}_Z\pi_Z.\mathbb{Q}_a\tau_a.\dots.\mathbb{Q}_z\tau_z.\psi$. Combining all the components, the encoding of the A-HLTL BMC problem into QBF, for bounds k and m is:

$$[\![\mathcal{K}, \varphi]\!]_{k,m} = \mathbb{Q}_A\overline{x_A}.\cdots.\mathbb{Q}_Z\overline{x_Z}.\mathbb{Q}_a\overline{t_a}.\cdots.\mathbb{Q}_z\overline{t_z}.\ \exists\overline{pos}.\ \exists\overline{\mathit{off}}.$$
$$\Big([\![\mathcal{K}]\!]_k \circ_A \cdots [\![\mathcal{K}]\!]_k \circ_Z (\varphi_{pos} \land enc(\psi))\Big)$$

where $\circ_A = \rightarrow$ if $\mathbb{Q}_A = \forall$ (and $\circ_A = \land$ if $\mathbb{Q}_A = \exists$), and $\circ_B$, ... are defined similarly. The sets $\overline{pos}$ is the set of variables $pos^{i,j}_{\pi,\tau}$ that encode the positions and $\overline{\mathit{off}}$ is the set of variables $\mathit{off}^j_{\pi,\tau}$ that encode when a trace progress has fallen off its unrolling limit. We next define the encoding $enc(\psi)$ of the temporal formula ψ.

Encoding formulas with up to 1 trajectory quantifier alternations We consider the encoding into QBF of formulas with zero and one quantifier alternation separately. In the following, we say that at position j a collection of trajectories U "moves" whenever either all trajectories have moved all their paths to the halting state, or at least one of the trajectories in U makes one of the non-halted path move at position j. Formally,

$$moves^j_U \stackrel{\text{def}}{=} halted^j_U \lor \bigvee_{\tau \in U, \pi} (t^j_\pi \land \neg halt^j_{\pi,\tau})$$

– $\mathsf{E}^+U.\psi$: In this case, the formula generated for $enc(\psi)$ is

$$(\bigwedge_{j \in \{0 \dots m\}} moves^j_U) \land [\![\psi]\!]^0_{k,m}$$

This is correct since the positions at which all trajectories stutter all paths can be removed (obtaining a satisfying path), we can restrict the search to non-stuttering trajectory steps.

– $\mathsf{A}^+U.\psi$: In this case, the formula generated for $enc(\psi)$ is

$$(\bigwedge_{j \in \{0 \dots m\}} moves^j_U) \rightarrow [\![\psi]\!]^0_{k,m}$$

The reasoning is similar as the previous case.

- $\mathsf{A}^+ U_A \mathsf{E}^+ U_E.\psi$: In this case, the formula generated for $enc(\psi)$ is

$$\Big(\bigwedge_{j\in\{0\ldots m\}} moves^j_{U_A}\Big) \rightarrow \Big(\bigwedge_{j\in\{0\ldots m\}} (halted^j_{U_A} \rightarrow moves^j_{U_E}) \wedge [\![\psi]\!]^0_{k,m}\Big)$$

Universally quantified trajectories must explore all trajectories, which must be responded by the existential trajectories. Assume there is a strategy for U_E for the case that universal trajectories U_A never stutter at any position. This can be extended into a strategy for the case where U_A can possible stutter, by adding a stuttering step to the U_E trajectories at the same position. This guarantees the same evaluation. Therefore, we restrict our search for the outer U_A to non-stuttering trajectories. Finally, U_E is obliged to move after U_A has halted all paths to prevent global stuttering.
- $\mathsf{E}^+ U_E \mathsf{A}^+ U_A.\psi$: In this case, the formula generated for $enc(\psi)$ is similar,

$$\Big(\bigwedge_{j\in\{0\ldots m\}} moves^j_{U_E}\Big) \wedge \Big(\bigwedge_{j\in\{0\ldots m\}} (halted^j_{U_E} \rightarrow moves^j_{U_A}) \rightarrow [\![\psi]\!]^0_{k,m}\Big)$$

The rationale for this encoding is the following. It is not necessary to explore a non-moving step j for the existentially quantified trajectories U_E because if this stuttering step is successful it must work for all possible moves of the U_A trajectories at the same time step j. This includes the case that all trajectories in U_A make all paths stutter (which, if we remove j one still has all the legal trajectories for U_A). Since the logic does not contain the next operator, the evaluation for the given U_E and one of the trajectories for U_A that stutter at j will be the same as for $j+1$ for all logical formulas. Therefore, the trajectory that is obtained from removing step j from U_E is still a satisfying trajectory assignment. It follows that if there is a model for U_E there is a model that does not stutter. Finally, after all paths have halted according to the U_E trajectories, a step of U_A that stutters all paths that have not halted can be removed because, again the evaluation is the same in the previous and subsequent state. It follows that if the formula has a model, then it has a model satisfying the encoding.

Theorem 1. *Let φ be an* A-HLTL *formula with at most one trajectory quantifier alternation, let K be the maximum depth of a Kripke structure and let $M = K \times |\mathsf{Paths}(\varphi)| \times |\mathsf{Trajs}(\varphi)|$. Then, the following hold:*
- $[\![\mathcal{K},\varphi]\!]^{hpes}_{K,M}$ *is satisfiable if and only if $\mathcal{K} \models \varphi$.*
- $[\![\mathcal{K},\varphi]\!]^{hopt}_{K,M}$ *is satisfiable if and only if $\mathcal{K} \models \varphi$.*

Theorem 1 (proof in [13]) provides a model checking decision procedure. An alternative decision procedure is to iteratively increase the bound of the unrollings and invoke both semantics in parallel until the outcome coincides.

4 Complexity of A-HLTL Model Checking for Acyclic Frames

Our goal in this section is to analyze the complexity of the A-HLTL model checking problem in the size of an acyclic Kripke structure (all proofs in [13]).

Problem Formulation. We use $\mathsf{MC}\big[\mathsf{Fragment}\big]$ to distinguish different variations of the problem, where MC is the model checking decision problem, i.e., whether or not $\mathcal{K} \models \varphi$, and $\mathsf{Fragment}$ is one of the following for φ:

- '$[\exists(\exists/\forall)^{+}\mathsf{A}/\mathsf{E}]^{k}$', for $k \geq 0$, is the fragment with a lead existential trace quantifier, one outermost universal or existential trajectory quantifier, and k (counting *all*) quantifier alternations, where $k = 0$ means the existential alternation-free fragment '$\exists^{+}\mathsf{E}^{+}$'. Fragment '$[\forall(\forall/\exists)^{+}\mathsf{A}/\mathsf{E}]^{k}$' is defined similarly, where $k = 0$ is the universal alternation-free fragment '$\forall^{+}\mathsf{A}^{+}$'.
- Fragments '$[\exists(\exists/\forall)^{+}(\mathsf{E}^{+}\mathsf{A}^{+}/\mathsf{A}^{+}\mathsf{E}^{+}/\mathsf{E}\mathsf{E}^{+}/\mathsf{A}\mathsf{A}^{+})]^{k}$', for $k \geq 1$ denotes the fragment with a lead existential trace quantifier, multiple outermost trajectory quantifiers with at most one alternation, and k quantifier alternations (counting *all* quantifiers), where $k = 1$ means fragment '$\exists\mathsf{E}\mathsf{A}$'. Fragment '$[\forall(\forall/\exists)^{+}(\mathsf{E}^{+}\mathsf{A}^{+}/\mathsf{A}^{+}\mathsf{E}^{+}/\mathsf{E}\mathsf{E}^{+}/\mathsf{A}\mathsf{A}^{+})]^{k}$' is defined similarly, where $k = 1$ means fragment '$\forall\mathsf{A}\mathsf{E}$'.

The Complexity of A-HLTL Model Checking. We first show the A-HLTL model checking problem for the alternation-free fragment with only one trajectory quantifier is NL-complete. For example, verification of information leak in speculative execution in sequential programs renders a formula of the form $\forall^{4}\mathsf{A}$, which belongs to the alternation-free fragment (more details in Section 5).

Theorem 2. *$\mathsf{MC}\big[\exists^{+}\mathsf{E}\big]$ and $\mathsf{MC}\big[\forall^{+}\mathsf{A}\big]$ are NL-complete.*

We now switch to formulas with alternating trace quantifiers. The significance of the next theorem is that a single trajectory quantifier does not change the complexity of model checking as compared to the classic HyperLTL verification [2]. It is noteworthy to mention that several important classes of formulas belong to this fragment. For example, according to Theorem 3 while model checking *observational determinism* [20] ($\forall\forall\mathsf{E}$), *generalized noninference* [16] ($\forall\forall\exists\mathsf{E}$), and *non-inference* [5] ($\forall\exists\mathsf{E}$) with a single initial input are all coNP-complete.

Theorem 3. *$\mathsf{MC}\big[\exists(\exists/\forall)^{+}(\mathsf{A}/\mathsf{E})\big]^{k}$ is Σ_{k}^{p}-complete and $\mathsf{MC}\big[\forall(\forall/\exists)^{+}(\mathsf{E}/\mathsf{A})\big]^{k}$ is Π_{k}^{p}-complete in the size of the Kripke structure.*

We now focus on formulas with multiple trajectory quantifiers. We first show that alternation-free multiple trajectory quantifiers bumps the class of complexity by one step in the polynomial hierarchy.

Theorem 4. *$\mathsf{MC}\big[\exists(\exists/\forall)^{+}\mathsf{E}\mathsf{E}^{+}\big]^{k}$ is Σ_{k+1}^{p}-complete and $\mathsf{MC}\big[\forall(\forall/\exists)^{+}\mathsf{A}\mathsf{A}^{+}\big]^{k}$ is Π_{k+1}^{p}-complete in the Kripke structure.*

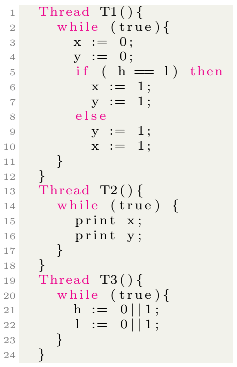
Theorem 5. *For $k \geq 1$, $\mathsf{MC}\big[\exists(\exists/\forall)^{+}\mathsf{A}^{+}\mathsf{E}^{+}\big]^{k}$ is Σ_{k+1}^{p}-complete and $\mathsf{MC}\big[\forall(\forall/\exists)^{+}\mathsf{E}^{+}\mathsf{A}^{+}\big]^{k}$ is Π_{k+1}^{p}-complete in the size of the Kripke structure.*

Finally, Theorems 3, 4, and 5 imply that the model checking problem for acyclic Kripke structures and A-HLTL formulas with an arbitrary number of trace quantifier alternation and only one trajectory quantifier is in PSPACE.

5 Case Studies and Evaluation

We now evaluate our technique. The encoding in Section 3 is implemented on top of the open-source bounded model checker HyperQB [15]. All experiments are executed on a MacBook Pro with 2.2GHz processor and 16GB RAM (https://github.com/TART-MSU/async_hltl_tacas23).

Non-interference in Concurrent Programs. We first consider the programs presented earlier in Figs. 1 and 3 together with A-HLTL formulas φ_{NI} and $\varphi_{\mathsf{NI}_{\mathsf{nd}}}$ from Section 1. We receive UNSAT (for the original formula and not its negation), which indicates that violations have been spotted. Indeed, our implementation successfully finds a counterexample with a specific trajectory that prints out `'acdb'` when the high-security value `h` is equal to zero (entries of `ACDB` and $\texttt{ACDB}_{\texttt{ndet}}$ in Table 3). Our other experiment is an extension of the example in [10] for multiple asynchronous channels (see Fig. 6) and the following formula: $\varphi_{\mathsf{OD}_{\mathsf{nd}}} = \forall\pi.\forall\pi'.\mathsf{A}\tau.\ \mathsf{E}\tau'.\ \square\,(\mathsf{l}_{\pi,\tau} \leftrightarrow \mathsf{l}_{\pi',\tau}) \rightarrow \square\,(\mathsf{obs}_{\pi,\tau'} \leftrightarrow \mathsf{obs}_{\pi',\tau'})$. The results for this case are entries of `ConcLeak` and $\texttt{ConcLeak}_{\texttt{ndet}}$ in Table 3. Details of the counterexample can be found in [13].



```
Thread T1(){
  while (true){
    x := 0;
    y := 0;
    if ( h == 1) then
      x := 1;
      y := 1;
    else
      y := 1;
      x := 1;
  }
}
Thread T2(){
  while (true) {
    print x;
    print y;
  }
}
Thread T3(){
  while (true){
    h := 0||1;
    l := 0||1;
  }
}
```

Fig. 6: Program with nondeterministic sequence of inputs.

Speculative Information Flow. *Speculative execution* is a standard optimization technique that allows branch prediction by the processor. *Speculative non-interference* (SNI) [9] requires that two executions with the same *policy* p (i.e., initial configuration) can be observed differently in speculative semantics (e.g., a possible branch), if and only if their non-speculative semantics with normal condition checks are also observed differently; i.e., the following A-HLTL formula:

$$\varphi_{\mathsf{SNI}} = \underbrace{\forall\pi_1.\forall\pi_2.}_{\text{speculative}}\ \underbrace{\forall\pi'_1.\forall\pi'_2}_{\text{nonspeculative}}\ .\ \mathsf{A}\tau.\Big(\square(\mathsf{obs}_{\pi_1,\tau} \leftrightarrow \mathsf{obs}_{\pi_2,\tau}) \wedge$$

$$(\mathsf{p}_{\pi_1,\tau} \leftrightarrow \mathsf{p}_{\pi_2,\tau}) \wedge (\mathsf{p}_{\pi_1,\tau} \leftrightarrow \mathsf{p}_{\pi'_1,\tau}) \wedge (\mathsf{p}_{\pi_2,\tau} \leftrightarrow \mathsf{p}_{\pi'_2,\tau})\Big) \rightarrow \square\big(\mathsf{obs}_{\pi'_1,\tau} \leftrightarrow \mathsf{obs}_{\pi'_2,\tau}\big)$$

where obs is the memory footprint, traces π_1 and π_2 range over the (nonspeculative) `C` code and traces π'_1 and π'_2 range over the corresponding (speculative) assembly code. We evaluate SNI on the translation from a `C` program (details in [13]), where `y` is the input policy `p` and multiple versions of `x86` assembly code [9]. The results of model checking speculative execution are in Table 3 (see entries from $\texttt{SpecExcu}_{V1}$ to $\texttt{SpecExcu}_{V7}$). Additional versions from $\texttt{SpecExcu}_{V3}$ to $\texttt{SpecExcu}_{V7}$ are under different compilation options. Our method correctly identify all the insecure and secure ones as stated in [9].

Compiler Optimization Security. Secure compiler optimization [17] aims at preserving input-output behaviors of a *source* program (original implementation) and a *target* program (after applying optimization), including security policies. We investigate the following optimization strategies: Dead Branch Elimination (DBE), Loop Peeling (LP), and Expression Flattening (EF). To verify a secure optimization, we consider two scenarios: (1) one single I/O event (one trajectory, similar to [1]), and (2) a sequences of I/O events (two trajectories):

$$\varphi_{\mathsf{SC}} = \forall\pi.\forall\pi'.\mathsf{E}\tau.\ (\mathsf{in}_{\pi,\tau} \leftrightarrow \mathsf{in}_{\pi',\tau}) \rightarrow \Box\,(\mathsf{out}_{\pi,\tau} \leftrightarrow \mathsf{out}_{\pi',\tau})$$
$$\varphi_{\mathsf{SC_{nd}}} = \forall\pi.\forall\pi'.\mathsf{A}\tau.\ \mathsf{E}\tau'.\Box\,(\mathsf{in}_{\pi,\tau} \leftrightarrow \mathsf{in}_{\pi',\tau}) \rightarrow \Box\,(\mathsf{out}_{\pi,\tau'} \leftrightarrow \mathsf{out}_{\pi',\tau'}),$$

where in is the set of inputs and out is the set of outputs. Table 3 (cases DBE – $\texttt{EFLP}_{\texttt{ndet}}$) shows the verification results of each optimization strategy and different combination of the strategies (details in [13]).

Cache-Based Timing Attacks. Asynchrony also leads to attacks when system executions are confined to a single CPU and its cache [18]. A cache-based timing attack happens when an attacker is able to guess the values of high-security variables when cache operations (i.e., evict, fetch) influence the scheduling of different threads. Our case study is inspired by the cache-based timing attack example in [18] and we use the formula of observational determinism $\varphi_{\mathsf{OD_{nd}}}$ introduced earlier in this section to find the potential attacks (see cases of CacheTA and $\texttt{CacheTA}_{\texttt{ndet}}$ in Table 3 with details in [13]).

5.1 Analysis of Experimental Results

Table 3 presents the diameter of the transition relation, length of trajectories m, state spaces, and the number of trajectory variables. We also present the total solving time of our algorithm as well as the break down: generating models (genQBF), building trajectory encodings (buildTr), and final QBF solving (solveQBF). Our two most complex cases are concurrent leak ($\texttt{ConcLeak}_{\texttt{ndet}}$) and loop peeling ($\texttt{LP}_{\texttt{ndet}}$). For concurrent leak, it is because there are three threads with many interleavings (i.e., asynchronous composition), takes longer time to build. For loop peeling, although there is no need to consider interleavings except for the nondeterministic inputs; however, the diameters of traces ($D_{\mathcal{K}_1}$, $D_{\mathcal{K}_2}$) are longer than other cases, which makes the length and size of trajectory variables (i.e., m and $|T|$) grow and increases the total solving time. Our encoding is able to handle a variety of cases with one or more trajectories, depending on whether multiple sources of non-determinism is present. To see efficiency, we compare the solving time for cases of compiler optimization with one trajectory with the results in [1]. This method reduces A-HLTL model checking to HyperLTL model checking for limited fragments and utilizes the model checker MCHyper. On the other hand, we directly handle asynchrony by trajectory encoding. Table 2 shows our algorithm considerably outperforms the approach in [1] in larger cases.

	MCHyper [1]	This paper	
Case	**Total**[s]	genQBF/ buildTr/ solveQBF[s]	**Total**[s]
DBE	**0.8**	0.9 / 0.07 / 0.01	**0.98**
LP	**365.9**	1.37 / 1.40 / 1.13	**3.90**
EFLP	**1315.2**	5.11 / 8.12 / 9.35	**22.58**

Table 2: Comparison of model checking compiler optimization with [1].

		(model checking spec and data)							(time took for solving)			
Models	φ	$D_{\mathcal{K}_1}$	$D_{\mathcal{K}_2}$	m	$\lvert S_{\mathcal{K}_1}\rvert$	$\lvert S_{\mathcal{K}_2}\rvert$	$\lvert T\rvert$	QBF	`genQBF`[s]	`buildTr`[s]	`solveQBF`[s]	**Total**[s]
`ACDB`	φ_{NI}	6	6	12	109	109	1378	`UNSAT`	2.80	0.32	0.23	**3.35**
`ACDB`$_{\mathtt{ndet}}$	$\varphi_{\mathsf{NI}_{\mathsf{nd}}}$	8	8	16	696	696	2754	`UNSAT`	7.74	2.54	3.73	**14.01**
`ConcLeak`	φ_{OD}	11	11	22	597	597	6118	`UNSAT`	14.85	7.10	8.29	**30.24**
`ConcLeak`$_{\mathtt{ndet}}$	$\varphi_{\mathsf{OD}_{\mathsf{nd}}}$	18	18	36	2988	2988	22274	`UNSAT`	127.09	53.14	731.48	**911.72**
`SpecExcu`$_{V1}$	φ_{SNI}	3	6	9	132	340	1112	`UNSAT`	7.45	1.72	3.07	**12.24**
`SpecExcu`$_{V2}$	φ_{SNI}	3	6	9	144	168	1112	`SAT`	5.61	1.28	2.44	**9.33**
`SpecExcu`$_{V3}$	φ_{SNI}	3	6	9	87	340	636	`UNSAT`	7.30	1.68	2.97	**11.95**
`SpecExcu`$_{V4}$	φ_{SNI}	3	6	9	93	340	636	`UNSAT`	7.37	1.71	4.50	**13.58**
`SpecExcu`$_{V5}$	φ_{SNI}	3	6	9	132	168	636	`SAT`	6.23	1.23	3.48	**10.94**
`SpecExcu`$_{V6}$	φ_{SNI}	3	7	10	132	340	766	`UNSAT`	7.47	1.82	3.26	**12.55**
`SpecExcu`$_{V7}$	φ_{SNI}	2	5	7	144	168	352	`SAT`	5.83	1.28	2.58	**9.69**
`DBE`	φ_{SC}	4	4	8	8	6	546	`SAT`	0.9	0.07	0.01	**0.98**
`DBE`$_{\mathtt{ndet}}$	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	13	13	26	82	72	9414	`SAT`	1.60	0.56	9.61	**11.77**
`DBE`$_{\mathtt{ndet}}$ w/ bugs	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	13	13	26	82	72	9414	`UNSAT`	1.36	0.49	2.05	**3.90**
`LP`	φ_{SC}	22	22	44	80	76	3870	`SAT`	1.37	1.40	1.13	**3.90**
`LP`$_{\mathtt{ndet}}$	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	17	17	34	558	811	19110	`SAT`	7.37	3.86	48.15	**59.38**
`LP`$_{\mathtt{ndet}}$ w/ loops	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	33	35	68	757	1591	128114	`SAT`	30.52	34.99	4165.54	**4231.05**
`LP`$_{\mathtt{ndet}}$ w/ bugs	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	17	17	34	558	661	19110	`UNSAT`	6.51	3.60	20.75	**30.86**
`EFLP`	φ_{SC}	32	32	64	80	248	108290	`SAT`	5.11	8.12	9.35	**22.58**
`EFLP`$_{\mathtt{ndet}}$	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	18	22	40	582	1729	28986	`SAT`	15.92	8.90	135.48	**160.30**
`EFLP`$_{\mathtt{ndet}}$ w/ loops	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	33	45	78	295	1996	178894	`SAT`	36.98	62.89	121.60	**221.47**
`CacheTA`	φ_{OD}	13	13	26	48	48	9414	`UNSAT`	1.49	0.53	0.38	**2.40**
`CacheTA`$_{\mathtt{ndet}}$	$\varphi_{\mathsf{OD}_{\mathsf{nd}}}$	58	58	16	16	32	16258	`UNSAT`	1.95	1.33	1.02	**4.30**
`CacheTA`$_{\mathtt{ndet}}$ w/ loops	$\varphi_{\mathsf{OD}_{\mathsf{nd}}}$	35	35	70	88	88	139302	`UNSAT`	5.50	27.65	125.92	**159.07**

Table 3: Case studies break down for Kripke structures: $\mathcal{K}_1, \mathcal{K}_2$ (all case studies have two, e.g.,one for high-level and one for assembly code), formula: φ, diameter: D, state space: $|S|$, trajectory depth: m, and size of trajectory variables: $|T|$.

6 Conclusion and Future Work

In this paper, we focused on the problem of A-HLTL model checking for *terminating* programs. We generalized A-HLTL to allow nested *trajectory* quantification, where a trajectory determines how different traces may advance and stutter. We rigorously analyzed the complexity of A-HLTL model checking for acyclic Kripke structures. The complexity grows in the polynomial hierarchy with the number of quantifier alternations, and, it is either aligned with that of HyperLTL or is one step higher in the polynomial hierarchy. We also proposed a BMC algorithm for A-HLTL based on QBF-solving and reported successful experimental results on verification of information flow security in concurrent programs, speculative execution, compiler optimization, and cache-based timing attacks.

Asynchronous hyperproperties enable logic-based verification for software programs. Thus, future work includes developing different abstraction techniques such as predicate abstraction, abstraction-refinement, etc, to develop software model checking techniques. We also believe developing synthesis techniques for A-HLTL creates opportunities to automatically generate secure programs and assist in areas such as secure compilation.

References

1. J. Baumeister, N. Coenen, B. Bonakdarpour, B. Finkbeiner, and C. Sánchez. A temporal logic for asynchronous hyperproperties. In *Proc. of the 33rd Int'l Conf. on Computer Aided Verification (CAV'21), Part I*, volume 12759 of *LNCS*, pages 694–717. Springer, 2021.
2. B. Bonakdarpour and B. Finkbeiner. The complexity of monitoring hyperproperties. In *Proceedings of the 31st IEEE Computer Security Foundations Symposium CSF*, pages 162–174, 2018.
3. L. Bozzelli, A. Peron, and C. Sánchez. Asynchronous extensions of HyperLTL. In *Proc. of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'21)*, pages 1–13. IEEE, 2021.
4. M. R. Clarkson, F. Finkbeiner, K. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez. Temporal logics for hyperproperties. In *Proceedings of the 3rd International Conference on Principles of Security and Trust (POST)*, pages 265–284, 2014.
5. M. R. Clarkson and F. B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.
6. N. Coenen, B. Finkbeiner, C. Hahn, and J. Hofmann. The hierarchy of hyperlogics. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019.
7. N. Coenen, B. Finkbeiner, C. Sánchez, and L. Tentrup. Verifying hyperliveness. In I. Dillig and S. Tasiran, editors, *Computer Aided Verification*, pages 121–139, Cham, 2019. Springer International Publishing.
8. B. Finkbeiner, M. Rabe, and C. Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *In Proc. of the 27th Int'l Conf. on Computer Aided Verification (CAV'15)*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.
9. M. Guarnieri, B. Köpf, J. F. Morales, J. Reineke, and A. Sánchez. SPECTECTOR: Principled detection of speculative information flows. In *Proceedings of the 41st IEEE Symposium on Security and Privacy*, S&P 2020. IEEE, 2020.
10. G. L. Guernic. Automaton-based confidentiality monitoring of concurrent programs. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF)*, pages 218–232, 2007.
11. J. O. Gutsfeld, M. Müller-Olm, and C. Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021.
12. M. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, 1990.
13. T. Hsu, B. Bonakdarpour, B. Finkbeiner, and C. Sánchez. Bounded model checking for asynchronous hyperproperties. *CoRR*, abs/2301.07208, 2023.
14. T. Hsu and C. Sánchez. Hyperqube: A qbf-based bounded model checker for hyperproperties. *CoRR*, abs/2109.12989, 2021.
15. T.-H. Hsu, C. Sánchez, and B. Bonakdarpour. Bounded model checking for hyperproperties. In *Proceedings of the 27th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, pages 94–112, 2021.
16. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 79–93, Apr. 1994.

17. K. S. Namjoshi and L. M. Tabajara. Witnessing secure compilation. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 1–22. Springer, 2020.
18. D. Stefan, P. Buiras, E. Z. Yang, A. Levy, D. Terei, A. Russo, and D. Mazières. Eliminating cache-based timing attacks with instruction-based scheduling. In *European Symposium on Research in Computer Security*, pages 718–735. Springer, 2013.
19. S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, page 29, 2003.