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 for Information Security, 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 (e.g., schedulers and asynchronous communication channels). In this paper, we focus on verification of confidentiality in nondeterministic environments by reasoning about *asynchronous hyperproperties*. First, 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 the generalized 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 for different fragments of A-HLTL.

1 Introduction

```
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`.

Motivation. Consider the concurrent program [11] 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` 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 [13] 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 == l) 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`.

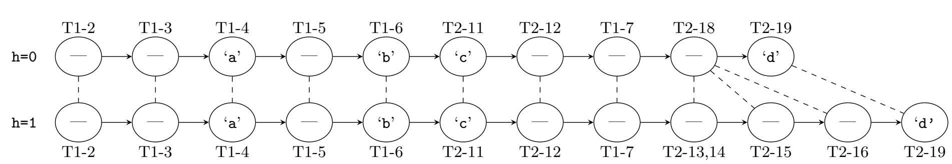

Fig. 2: Two secure interleavings for the program in Fig. 1

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
$\left[\exists(\exists/\forall)^+(\mathsf{A}/\mathsf{E})\right]^k$	Σ^p_k-complete
$\left[\forall(\exists/\forall)^+(\mathsf{E}/\mathsf{A})\right]^k$	Π^p_k-complete
Multiple Traces – Multiple Trajectories	
$\left[\exists(\exists/\forall)^+(\mathsf{E}^+\mathsf{E})\right]^k$	Σ^p_{k+1}-complete
$\left[\forall(\forall/\exists)^+(\mathsf{A}^+\mathsf{A})\right]^k$	Π^p_{k+1}-complete
$\left[\exists(\exists/\forall)^+\mathsf{A}^+\mathsf{E}^+\right]^k$	Σ^p_{k+1}-complete
$\left[\forall(\forall/\exists)^+\mathsf{E}^+\mathsf{A}^+\right]^k$	Π^p_{k+1}-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 [12]. 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.

Organization. The rest of the paper is organized as follows. We generalize A-HLTL in Section 2. Section 3 describes our bounded model checking algorithm while Section 4 is dedicated to our complexity analysis. Evaluation of our implementation results is presented in Section 5. We conclude in Section 6. Detailed proofs and descriptions of our case studies appear in the appendix.

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* $\mathit{\Gamma} : \mathsf{Trajs}(\varphi) \rightharpoonup \mathsf{TRJ}_{\mathit{Dom}(\mathit{\Gamma})}$, where $\mathit{Dom}(\mathit{\Gamma})$ is the subset of $\mathsf{Trajs}(\varphi)$ for which $\mathit{\Gamma}$ is defined. Given a trajectory assignment $\mathit{\Gamma}$, a trajectory variable τ, and a trajectory t, we denote by $\mathit{\Gamma}[\tau \mapsto t]$ the assignment that coincides with $\mathit{\Gamma}$ 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 $\mathit{\Pi} : \mathsf{Paths}(\varphi) \times \mathsf{Trajs}(\varphi) \rightarrow T \times \mathbb{N}$ which map each pair (π, τ) formed by a path variable and trajectory variable into a pointed trace. Given $(\mathit{\Pi}, \mathit{\Gamma})$ where $\mathit{\Pi}$ is an asynchronous trace assignment and $\mathit{\Gamma}$ a trajectory assignment, we use

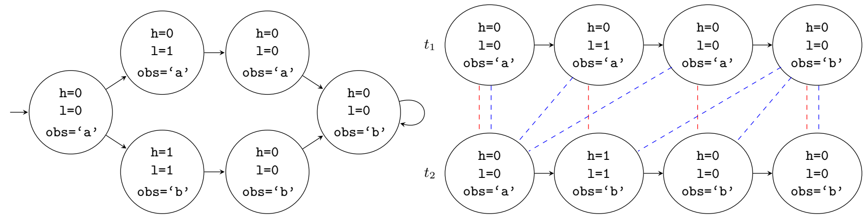

Fig. 4: Kripke structure $\mathcal{K}$ (left) and the two traces t_1 and t_2 of $\mathcal{K}$ (right), $\mathcal{K} \models \varphi_{\mathsf{NI}_{\mathsf{nd}}}$ but $\mathcal{K} \not\models \varphi_{\mathsf{NI}}$.

$(\Pi, \Gamma) + 1$ 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 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):

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

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 $pos_{\pi,\tau} : \{0 \dots m\} \rightarrow \mathbb{N}$. The meaning of $pos_{\pi,\tau}(i)$ provides how many times π has been selected in $\{\tau(0), \dots, \tau(i)\}$. 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.

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

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

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

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

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 $\bigcirc$):

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

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

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

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

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

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 $i = m$:

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

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

As the semantics introduced in [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.

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 $(x^0,\dots,x^k)$ 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 $t^0_\pi \dots t^m_\pi$, 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 $i \in \{0\dots k\}$ and $j \in \{0\dots m\}$) 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):

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

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

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

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

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

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 $step^{j}_{\pi,\tau'}$ (i.e., $pos^{0,0}_{\pi,\tau'}, pos^{1,1}_{\pi,\tau'}, pos^{2,2}_{\pi,\tau'}, pos^{3,3}_{\pi,\tau'}$); while π' stutters according to $stutters^{j}_{\pi',\tau'}$ (i.e., $pos^{0,0}_{\pi',\tau'}, pos^{0,1}_{\pi',\tau'}, pos^{0,2}_{\pi',\tau'}, pos^{0,3}_{\pi',\tau'}$). 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 $pos^{3,4}_{\pi,\tau'}, pos^{3,5}_{\pi,\tau'}, pos^{3,6}_{\pi,\tau'}$ and $pos^{1,4}_{\pi',\tau'}, pos^{2,5}_{\pi',\tau'}, pos^{3,6}_{\pi',\tau'}$. 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:

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

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 $j \in \{0 \ldots m\}$.

$$[\![p_{\pi,\tau}]\!]^{j}_{k,m} := \bigvee_{i\in\{0..k\}} (pos^{i,j}_{\pi,\tau} \wedge p^{i}_{\pi}) \tag{14}$$

$$[\![\neg p_{\pi,\tau}]\!]^{j}_{k,m} := \bigvee_{i\in\{0..k\}} (pos^{i,j}_{\pi,\tau} \wedge \neg p^{i}_{\pi}) \tag{15}$$

$$[\![\psi_1 \vee \psi_2]\!]^{j}_{k,m} := [\![\psi_1]\!]^{j}_{k,m} \vee [\![\psi_2]\!]^{j}_{k,m} \tag{16}$$

$$[\![\psi_1 \wedge \psi_2]\!]^{j}_{k,m} := [\![\psi_1]\!]^{j}_{k,m} \wedge [\![\psi_2]\!]^{j}_{k,m} \tag{17}$$

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

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

For $j = m$:

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

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

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

For $j = m$:

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

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:

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

where $\circ_A = \rightarrow$ if $\mathbb{Q}_A = \forall$ (and $\circ_A = \wedge$ 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 $\mathit{enc}(\psi)$ 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,

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

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

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

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

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

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

$$(\bigwedge_{j\in\{0\dots m\}} moves_{U_A}^j) \rightarrow (\bigwedge_{j\in\{0\dots m\}} (halted_{U_A}^j \rightarrow moves_{U_E}^j) \wedge [\![\psi]\!]_{k,m}^0)$$

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,

$$(\bigwedge_{j\in\{0\dots m\}} moves_{U_E}^j) \wedge (\bigwedge_{j\in\{0\dots m\}} (halted_{U_E}^j \rightarrow moves_{U_A}^j) \rightarrow [\![\psi]\!]_{k,m}^0)$$

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:*

– $[\![\mathcal{K}, \varphi]\!]_{K,M}^{hpes}$ *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 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.

Problem Formulation. We use MC[Fragment] 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 Fragment is one of the following for φ:

– '$[\exists(\exists/\forall)^{+}\mathsf{A}/\mathsf{E}]^{k}$', for $k \geq 0$, denotes the fragment with a lead existential trace quantifier, one outermost universal or existential trajectory quantifier, and k quantifier alternations (counting *all* quantifiers), 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. MC$[\exists^{+}\mathsf{E}]$ *and* MC$[\forall^{+}\mathsf{A}]$ *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. MC$[\exists(\exists/\forall)^{+}(\mathsf{A}/\mathsf{E})]^{k}$ *is* Σ^{p}_{k}*-complete and* MC$[\forall(\forall/\exists)^{+}(\mathsf{E}/\mathsf{A})]^{k}$ *is* Π^{p}_{k}*-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. $\mathsf{MC}\big[\exists(\exists/\forall)^{+}\mathsf{EE}^{+}\big]^{k}$ *is* Σ^{p}_{k+1}*-complete and* $\mathsf{MC}\big[\forall(\forall/\exists)^{+}\mathsf{AA}^{+}\big]^{k}$ *is* Π^{p}_{k+1}*-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^{p}_{k+1}*-complete and* $\mathsf{MC}\big[\forall(\forall/\exists)^{+}\mathsf{E}^{+}\mathsf{A}^{+}\big]^{k}$ *is* Π^{p}_{k+1}*-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 evaluated our algorithm in Section 3 on cases that require single or nested trajectories. The trajectory encoding presented in Section 3 is implemented on top of the open-source bounded model checker HyperQB [15], and the QBF solver QuABs [19]. All experiments are executed on a MacBook Pro with 2.2GHz processor and 16GB RAM[4].

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 [11] 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 Appendix B.1.

```
Thread T1(){
  while (true){
    x := 0;
    y := 0;
    if ( h == l) then
      x := 1;
      y := 1;
    else
      y := 1;
      x := 1;
  }
}
Thread T2(){
  while (true) {
    print x;
    print y;
  }
}
Thread T2(){
  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) [10] requires that two executions with the same *policy* `p` (i.e., initial configuration) can be observed differently in speculative semantics (e.g.,

[4] https://github.com/TART-MSU/async_hltl_tacas23

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:

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

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

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 in Fig. 10a , where y is the input policy p and multiple versions of x86 assembly code [10] (details in Appendix B.2). 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 [10].

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):

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

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

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 $\varphi_{\mathsf{OD}_{\mathsf{nd}}}$ introduced earlier in this section to find the potential attacks (see cases of CacheTA and $\texttt{CacheTA}_{\texttt{ndet}}$ in Table 3) The details of the case study is discussed in Appendix B.4.

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, in this paper, we directly handle the asynchrony by trajectory encoding presented in Section 3. 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].

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.

		(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**
$\texttt{ACDB}_{\texttt{ndet}}$	$\varphi_{\mathsf{NI}_{\mathsf{nd}}}$	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**
$\texttt{ConcLeak}_{\texttt{ndet}}$	$\varphi_{\mathsf{OD}_{\mathsf{nd}}}$	18	18	36	2988	2988	22274	`UNSAT`	127.09	53.14	731.48	**911.72**
$\texttt{SpecExcu}_{V1}$	φ_{SNI}	3	6	9	132	340	1112	`UNSAT`	7.45	1.72	3.07	**12.24**
$\texttt{SpecExcu}_{V2}$	φ_{SNI}	3	6	9	144	168	1112	`SAT`	5.61	1.28	2.44	**9.33**
$\texttt{SpecExcu}_{V3}$	φ_{SNI}	3	6	9	87	340	636	`UNSAT`	7.30	1.68	2.97	**11.95**
$\texttt{SpecExcu}_{V4}$	φ_{SNI}	3	6	9	93	340	636	`UNSAT`	7.37	1.71	4.50	**13.58**
$\texttt{SpecExcu}_{V5}$	φ_{SNI}	3	6	9	132	168	636	`SAT`	6.23	1.23	3.48	**10.94**
$\texttt{SpecExcu}_{V6}$	φ_{SNI}	3	7	10	132	340	766	`UNSAT`	7.47	1.82	3.26	**12.55**
$\texttt{SpecExcu}_{V7}$	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**
$\texttt{DBE}_{\texttt{ndet}}$	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	13	13	26	82	72	9414	`SAT`	1.60	0.56	9.61	**11.77**
$\texttt{DBE}_{\texttt{ndet}}$ 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**
$\texttt{LP}_{\texttt{ndet}}$	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	17	17	34	558	811	19110	`SAT`	7.37	3.86	48.15	**59.38**
$\texttt{LP}_{\texttt{ndet}}$ w/ loops	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	33	35	68	757	1591	128114	`SAT`	30.52	34.99	4165.54	**4231.05**
$\texttt{LP}_{\texttt{ndet}}$ 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**
$\texttt{EFLP}_{\texttt{ndet}}$	$\varphi_{\mathsf{SC}_{\mathsf{nd}}}$	18	22	40	582	1729	28986	`SAT`	15.92	8.90	135.48	**160.30**
$\texttt{EFLP}_{\texttt{ndet}}$ 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**
$\texttt{CacheTA}_{\texttt{ndet}}$	$\varphi_{\mathsf{OD}_{\mathsf{nd}}}$	58	58	16	16	32	16258	`UNSAT`	1.95	1.33	1.02	**4.30**
$\texttt{CacheTA}_{\texttt{ndet}}$ 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: $|T|$.

Acknowledgment

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)

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. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.
10. 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.
11. 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.
12. J. O. Gutsfeld, M. Müller-Olm, and C. Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021.
13. 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.
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. L. Tentrup. CAQE and quabs: Abstraction based QBF solvers. *Journal of Satisfiability Boolean Modeling and Computation*, 11(1):155–210, 2019.
20. 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.

A Detailed Proofs

Proof of Theorem 1

From our construction in Section 3, the following lemma follows.

Lemma 3. *Let φ be an* A-HLTL *formula with at most one trajectory quantifier alternation and k and m be unrolling bounds. Then,*

1. *If $[\![\mathcal{K}, \varphi]\!]_{k,m}^{hpes}$ is satisfiable, then $\mathcal{K} \models \varphi$.*
2. *If $[\![\mathcal{K}, \varphi]\!]_{k,m}^{hopt}$ is unsatisfiable, then $\mathcal{K} \not\models \varphi$.*

Let K be the maximum length of any path in any Kripke structure. It is easy to see that in all cases, after at most $K * |\mathsf{Paths}| * |\mathsf{Trajs}|$ steps, all paths have halted according to all trajectories because at every step there is always some trajectory moving some non-halted path. Since the halting optimistic and the pessimistic semantics only differ when the paths do not halt after the unrolling limit consider, the following result holds.

Lemma 4. *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 $M = K * |\mathsf{Paths}(\varphi)| * |\mathsf{Trajs}(\varphi)|$. Then, $[\![\mathcal{K}, \varphi]\!]_{K,M}^{hpes} = [\![\mathcal{K}, \varphi]\!]_{K,M}^{hopt}$.*

Finally, Lemmas 3 and 4 imply that after the unrolling bound $k * |\mathsf{Paths}(\varphi)| * |\mathsf{Trajs}(\varphi)|$ both the halting optimistic and pessimistic give the correct answer to the model checking problem.

Proof of Theorem 2

For the upper bound, we consider the case that the A-HLTL formula is existential, i.e., it is of the form:

$$\exists \pi_1 \ldots \exists \pi_k . \mathsf{E}\tau . \, \varphi,$$

where φ does not contain any trace quantifiers. For the case that the formula is universal, i.e., it is of the form:

$$\forall \pi_1 \ldots \forall \pi_k . \mathsf{A}\tau . \, \varphi,$$

we check the formula $\exists \pi_1 \ldots \exists \pi_k . \mathsf{E}\tau . \, \neg\varphi$ and report the complemented result.

The algorithm for the upper bound works as follows. Since the Kripke structure is acyclic, the length of the traces is bounded by the number of states of the Kripke structure. We can, therefore, nondeterministically guess the witness to traces $\pi_1 \cdots \pi_k$ and trajectory τ that satisfy the inner LTL formula φ using a counter per trace (with a logarithmic number of bits in the number of states of $\mathcal{K}$) and k bits for the trajectory. Observe that τ merely prescribes how the traces advance. That means one can obtain traces $\sigma_1 \cdots \sigma_k$ from the witnesses to $\pi_1 \cdots \pi_k$ that advance synchronously (i.e., all traces advance in a lockstep manner), where the length of traces is dictated by the guessed witness to τ. Since verifying the correctness of φ on these traces (that form a tree-shaped graph) can be done in logarithmic time [2], the upper bound remains in NL. We emphasize that the number of counters and extra k bits are in the size of the formula which is assumed to be a constant, as our complexity analysis is in the size of the input Kripke structure.

The lower bound follows from the NL-hardness of standard HyperLTL model checking for acyclic graphs [2]. □

Proof of Theorem 3

We show membership in Σ_k^p and Π_k^p, respectively, by induction over k. We begin with the base case ($k = 1$), that is, the fragment $\exists^+\mathsf{A}$. By nondeterministically guessing the witnesses to the existential trace quantifiers, according to Theorem 2, the model checking problem for only a universal trajectory quantifier is solvable in polynomial time. That means for $k = 1$, $\mathsf{MC}\big[\exists^+\mathsf{A}\big]$ is in $\mathsf{NP} = \Sigma_1^p$. Dually, for $k = 1$, $\mathsf{MC}\big[\forall^+\mathsf{E}\big]$ is in $\mathsf{coNP} = \Pi_1^p$.

For the inductive step, let us first focus on decision problem $\mathsf{MC}\big[\exists(\exists/\forall)^+\mathsf{A}\big]^k$. Since the Kripke structure is acyclic, the length of the traces is bounded by the number of states. We can, thus, nondeterministically guess the existentially quantified traces in polynomial time and then verify the correctness of the guess, by the induction hypothesis, in Π_{k-1}^p. Hence, the model checking problem for k alternations is in Σ_k^p. Likewise, for the decision problem $\mathsf{MC}\big[\forall(\forall/\exists)^+\mathsf{E}\big]^k$, we universally guess the universal quantified traces in polynomial time and verify the correctness of the guess, by the induction hypothesis, in Σ_k^p. Hence, the problem of determining $\mathcal{K} \models \varphi$ for k alternations in φ is in Π_k^p.

For the lower bound, we show that $\mathsf{MC}\big[\exists(\exists/\forall)^+\mathsf{A}\big]^k$ and $\mathsf{MC}\big[\forall(\forall/\exists)^+\mathsf{E}\big]^k$ are Σ_k^p-hard and Π_k^p-hard, respectively, via a reduction from the *quantified Boolean formula* (QBF) satisfiability problem [9]:

> *Given is a set of Boolean variables, $\{x_1, x_2, \ldots, x_n\}$, and a quantified Boolean formula*
>
> $$y = \mathbb{Q}_1 x_1.\mathbb{Q}_2 x_2 \ldots \mathbb{Q}_{n-1} x_{n-1}.\mathbb{Q}_n x_n.(y_1 \wedge y_2 \wedge \cdots \wedge y_m)$$
>
> *where each $\mathbb{Q}_i \in \{\forall, \exists\}$ ($i \in [1, n]$) and each clause y_j ($j \in [1, m]$) is a disjunction of three literals (3CNF). Is y true?*

If y is restricted to at most k alternations of quantifiers, then QBF satisfiability is complete for Σ_{k+1}^p if $\mathbb{Q}_1 = \exists$, and for Π_k^p if $\mathbb{Q}_1 = \forall$. We note that in the given instance of the QBF problem:

- The clauses may have more than three literals, but three is sufficient of our purpose;
- The inner Boolean formula has to be in conjunctive normal form in order for our reduction to work;
- Without loss of generality, the variables in the literals of the same clause are different (this can be achieved by a simple pre-processing of the formula), and

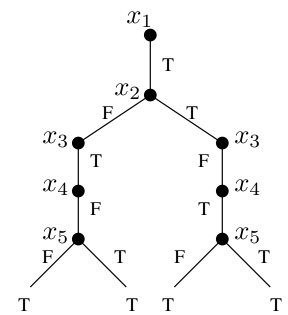

Fig. 7: Model for the QBF $y = \exists x_1.\forall x_2.\exists x_3.\exists x_4.\forall x_5.(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_3 \vee x_4 \vee \neg x_5) \wedge (x_1 \vee x_4 \vee x_5)$.

– If the formula has k alternations, then it has $k+1$ alternation *depths*. For example, formula

$$\forall x_1.\exists x_2.(x_1 \vee \neg x_2)$$

has one alternation, but two alternation depths: one for $\forall x_1$ and the second for $\exists x_2$. By $d(x_i)$, we mean the alternation depth of Boolean variable x_i.

We now present a mapping from an arbitrary instance of QBF with k alternations and where $\mathbb{Q}_1 = \exists$ to the model checking problem of an acyclic Kripke structure and a A-HLTL formula with k trace quantifier alternations and one innermost universal trajectory quantifier. Then, we show that the Kripke structure satisfies the A-HLTL formula if and only if the answer to the QBF problem is affirmative. Figures 7 and 8 show an example.

Kripke structure $\mathcal{K} = \langle S, s_{init}, \delta, L\rangle$:

– *(Atomic propositions* AP*)* For each alternation depth $d \in [1, k{+}1]$, we include an atomic proposition q^d. We furthermore include five atomic propositions: p is used to force clauses to become true if a positive literal appears in a clause; proposition $\bar{p}$ is used to force clauses to become true if a negative literal appears in a clause in our reduction; start marks the beginning of the gadget of states that represent the Boolean variables in the QBF instance; sync is used to mark the beginning a chain of states that represent a clause, and c is used to enforce lock-step synchronization of Boolean variables in their respective clauses. Thus,

$$\mathsf{AP} = \big\{c, p, \bar{p}, \mathsf{start}, \mathsf{sync}\big\} \;\cup\; \big\{q^d \mid d \in [1, k+1]\big\}.$$

– *(Set of states S)* We now identify the members of S:
  - First, we include an initial state s_{init} labeled by start.
  - For each Boolean variable x_i, where $i \in [1, n]$, we include three states s_i and $\bar{s}_i$. Each state s_i (respectively, $\bar{s}_i$) is labeled by p and $q^{d(x_i)}$ (respectively, $\bar{p}$ and $q^{d(x_i)}$).

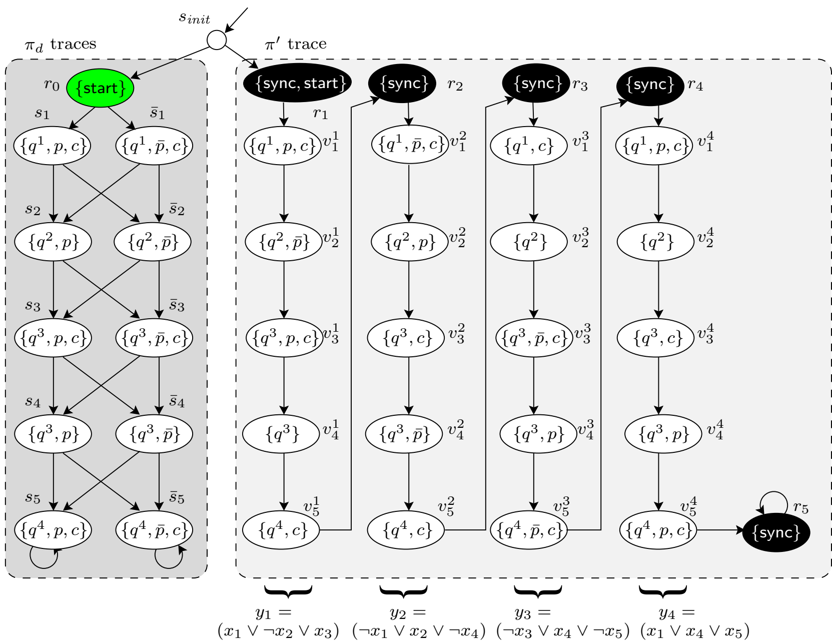

Fig. 8: Mapping quantified Boolean formula $y = \exists x_1.\forall x_2.\exists x_3.\exists x_4.\forall x_5.(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_3 \vee x_4 \vee \neg x_5) \wedge (x_1 \vee x_4 \vee x_5)$ to an instance of $\mathsf{MC}[\exists(\exists/\forall)^k\mathsf{A}]$.

- For each clause y_j, where $j \in [1, m]$, we include a state r_j, labeled by proposition sync. We also include state r_{m+1} labeled by sync that marks the end of chain of states that represent the literals in the clauses.
- For each clause y_j, where $j \in [1, m]$, we introduce the following n states:

$$\Big\{v_i^j \mid i \in [1, n]\Big\}.$$

  Each state v_i^j is labeled with propositions $q^{d(x_i)}$, and with p if x_i is a literal in y_j, or with $\bar{p}$ if $\neg x_i$ is a literal in y_j.
- Finally, we label states s_i, $\bar{s}_i$, and v_i^j by proposition c, if i is odd, for all $i \in [1, n]$.

Thus,

$$S = \big\{s_{init}\big\} \cup \big\{r_j \mid j \in [0, m+1]\big\} \cup \big\{v_i^j, s_i, \bar{s}_i, \mid i \in [1, n] \wedge j \in [1, m]\big\}.$$

– *(Transition relation δ)* We now identify the members of δ:

$$\begin{aligned}\delta = &\big\{(s_{init}, r_0), (s_{init}, r_1)\big\} \cup \big\{(r_0, s_1), (r_0, \bar{s}_1), (v_n^m, r_{m+1})\big\} \cup \\ &\big\{(v_i^j, v_{i+1}^j) \mid i \in [1, n) \wedge j \in [1, m]\big\} \cup \\ &\big\{(s_i, s_{i+1}), (\bar{s}_i, s_{i+1}), (s_i, \bar{s}_{i+1}), (\bar{s}_i, \bar{s}_{i+1}) \mid i \in [1, n)\big\} \cup \\ &\big\{(s_n, s_n), (\bar{s}_n, \bar{s}_n), (r_{m+1}, r_{m+1})\big\}.\end{aligned}$$

A-HLTL formula: The A-HLTL formula in our mapping is the following:

$$\varphi_{\mathsf{map}} = \overbrace{\mathbb{Q}_1\pi_1.\mathbb{Q}_2\pi_2\cdots\mathbb{Q}_n}^{\mathcal{Q}\pi'}\pi_n.\mathsf{A}\tau.$$

$$\Big(\bigwedge_{d\in\{i\mid\mathbb{Q}_i=\forall\}}\underbrace{\Diamond(\mathsf{start}_{\pi_d,\tau}\wedge\neg\mathsf{sync}_{\pi_d,\tau})}_{\psi_1}\wedge\underbrace{\Diamond(\mathsf{start}_{\pi',\tau}\wedge\mathsf{sync}_{\pi',\tau})}_{\psi_2}\Big)$$

$$\longrightarrow$$

$$\left(\begin{array}{c}\Big[\bigwedge_{d\in\{i\mid\mathbb{Q}_i=\exists\}}\underbrace{\Diamond(\mathsf{start}_{\pi_d,\tau}\wedge\neg\mathsf{sync}_{\pi_d,\tau})}_{\psi_3}\Big]\wedge\\ \Box\Big[\underbrace{\bigwedge_{d\in[1,n]}\Big(\mathsf{sync}_{\pi',\tau}\wedge\big(\mathsf{sync}_{\pi',\tau}\,\mathcal{U}\,(\neg\mathsf{sync}_{\pi',\tau}\wedge(c_{\pi_d,\tau}\leftrightarrow c_{\pi',\tau}))\,\mathcal{U}\,\mathsf{sync}_{\pi',\tau}\big)\Big)}_{\psi_4}\\ \longrightarrow\\ \bigvee_{d\in[1,n]}\underbrace{\Diamond\Big(\big(q^d_{\pi_d,\tau}\leftrightarrow q^d_{\pi',\tau}\big)\wedge\big((p_{\pi',\tau}\wedge p_{\pi_d,\tau})\vee(\bar{p}_{\pi',\tau}\wedge\bar{p}_{\pi_d,\tau})\big)\Big)}_{\psi_5}\Big]\end{array}\right)$$

where for each $i \in [1, n]$, $\mathbb{Q}_i$ is the same type of quantifier as in the input QBF instance. Observe that there is only one way to instantiate trace π', namely the path that starts s_{init} and ends in r_5. Thus, we may insert quantifier $\mathcal{Q}\pi'$ in any place such that it does not change the number of alternations of $\mathbb{Q}_1\pi_1.\mathbb{Q}_2\pi_2\cdots\mathbb{Q}_n\pi_n.\mathsf{A}\tau$. Quantifier $\mathcal{Q}$ can be either $\forall$ or $\exists$. If $\mathcal{Q} = \forall$, then A-HLTL formula is as φ_{map} above. If $\mathcal{Q} = \exists$, then in φ_{map} the sub-formula ψ_2 would have to appear as a conjunct on the right side of the first implication.

Since the input QBF has k alternations and the resulting A-HLTL formulas has $k+1$ alternations. Intuitively, this formula expresses the following. First, we limit the universal trajectory to ones that only align the states of the gadgets for Boolean variables $x_1 \cdots x_n$ and the states of the gadgets for clauses $y_1 \cdots y_m$ as aligned and they advance in a lock-step manner. Let us explain the purpose of each sub-formula in φ_{map}:

- Sub-formulas ψ_1 (for universal) and ψ_3 (for existential traces) ensure that π_d traces that range over the left substructure start from state r_0.
- Sub-formula ψ_2 moves the unique trace π' to state r_j, where $j \in [1, m]$. As mentioned earlier if $\mathcal{Q} = \exists$, then ψ_2 will appear as a conjunction with ψ_3.
- Sub-formula ψ_4 filters the universally quantified trajectory τ by allowing only those that (1) when reaching an r_j state, where sync holds; (2) move traces π_d and π' in lock-step (i.e., $(c_{\pi_d,\tau} \leftrightarrow c_{\pi',\tau})$), where π' is not in a sync state, and (3) until it reaches the last sync state.

– Sub-formula ψ_5 ensures that each instance of τ aligns with at least one of the clauses in the input QBF formula and evaluates that clause to true (either p or $\bar{p}$ agree with each other in π_d and π').

Also, formula φ_{map} forces τ to synchronize π_d at state r_0 with π' at one of the states labeled by sync (i.e., the beginning of a clause). Then, sub-formula $c_{\pi_d,\tau} \leftrightarrow c_{\pi',\tau}$ ensures that the trajectory advances π and π' in lock-step to determine whether the clause evaluates to true. Now, for all the trajectories that meet these conditions, if there exists a state where either p or $\bar{p}$ in π_d eventually matches its counterpart position in π', then clause is satisfied. The matching positions identify the assignments of Boolean variables in the corresponding clauses that make the QBF instance true.

We now show that the given quantified Boolean formula is *true* if and only if the Kripke structure obtained by our mapping satisfies the A-HLTL formula φ_{map}.

(⇒) Suppose that y is true. Then, there is an instantiation of existentially quantified trace variables for each value of universally quantified variables and the trajectory quantifier, such that each clause y_j, where $j \in [1, m]$ becomes true (see Figs. 7 and 8 for an example). We now use these instantiations to instantiate each $\exists\pi_{x_d}$ in A-HLTL formula φ_{map}, where $d \in \{i \mid \mathbb{Q}_i = \exists\}$ as follows. First, notice that π' can only be instantiated by the trace that reaches state r_1. Now, for each existentially quantified variable x_i, where $i \in [1, n]$, in depth $d \in [1, k+1]$, if $x_i = \mathtt{true}$, we instantiate π_d with a trace that includes state s_i. Otherwise, the trace will include state $\bar{s}_i$. We now show that this trace instantiation evaluates formula φ_{map} to true. Observe that the left side of the implication in the formula is basically filtering the non-legitimate trajectories and allows only those where traces π_d can synchronize with trace π'. Since each y_j is true, for any instantiation of universal quantifiers, there is at least one literal in y_j that is true. If this literal is of the form x_i, then we have $x_i = \mathtt{true}$ and trace π_d will include s_i, which is labeled by p and q^d. Hence, the values of p (respectively, q^d), in both π_d and π' instantiated by trace

$$s_{init} r_1 \cdots r_j v_1^j \cdots v_n^j \cdots r_{m+1}^{\omega}$$

are eventually equal. If the literal in y_j is of the form $\neg x_i$, then $x_i = \mathtt{false}$ and, hence, some trace π_d will include $\bar{s}_i$. Again, the values of $\bar{p}$ (respectively, q^d), in both π_d and π' are eventually equal. Finally, since all clauses are true, all traces π' reach a state where the right side of the implication becomes true.

(⇐) Suppose our mapped Kripke structure satisfies the A-HLTL formula φ_{map}. This means that for each instantiation of the trajectory τ, since trace π' is of the form $s_{init} r_1 \cdots r_j v_1^j \cdots v_n^j \cdots r_{m+1}^{\omega}$, then there exists a state v_i^j, where the values of q^d and either p or $\bar{p}$ are eventually equal to their counterparts in some trace π_d. If this trace is existentially quantified and includes s_i, then we assign $x_i = \mathtt{true}$ for the preceding quantifications. If the trace includes $\bar{s}_i$, then

$x_i = \texttt{false}$. Observe that since in no state p and $\bar{p}$ are simultaneously true and no trace includes both s_i and $\bar{s}_i$, variable x_i will have only one truth value. This way, a model similar to Fig. 7 can be constructed. Similar to the forward direction, it is straightforward to see that this valuation makes every clause y_j of the QBF instance true.

In our mapping if $\mathbb{Q}_1 = \exists$, the hardness of model checking for A-HLTL formulas is Σ_k^p. If $\mathbb{Q}_1 = \forall$, then analogously the problem becomes Π_k^p-hard. □

Proof of Theorem 4

We show membership to Σ_{k+1}^p and Π_{k+1}^p, respectively, by induction over k. We begin with the base case ($k = 0$), that is, the fragment $\exists^+\mathsf{EE}$. By guessing the witnesses to the existential and trajectory quantifiers, we can verify the correctness of the inner LTL formula in polynomial time. Thus, $\mathsf{MC}[\exists^+\mathsf{EE}]$ is in NP and dually $\mathsf{MC}[\forall^+\mathsf{AA}]$ is in coNP. For k quantifier alternations, let us first focus on the decision problem $\mathsf{MC}[\exists(\exists/\forall)^+\mathsf{EE}]^k$. Since the Kripke structure is acyclic, the length of the traces is bounded by the number of states. We can, thus, nondeterministically guess the existentially quantified traces in polynomial time and then verify the correctness of the guess, by the induction hypothesis, in Π_k^p. Hence, the model checking problem for k is in Σ_{k+1}^p. Likewise, for the decision problem $\mathsf{MC}[\forall(\forall/\exists)^+\mathsf{AA}]^k$, we universally guess the universally quantified traces in polynomial time and verify the correctness of the guess, by the induction hypothesis, in Σ_{k+1}^p. Hence, the problem of determining $\mathcal{K} \models \varphi$ for k alternations in φ is in Π_{k+1}^p.

For the lower bound, similar to the proof of Theorem 3, we show that $\mathsf{MC}[\exists(\exists/\forall)^+\mathsf{EE}]^k$ and $\mathsf{MC}[\forall(\forall\exists)^+\mathsf{AA}]^k$ are Σ_{k+1}^p-hard and Π_{k+1}^p-hard, respectively, via a reduction from the QBF satisfiability problem (see the description of the problem in the proof of Theorem 3). We now present a mapping from an arbitrary instance of QBF with k alternations to the model checking problem of an acyclic

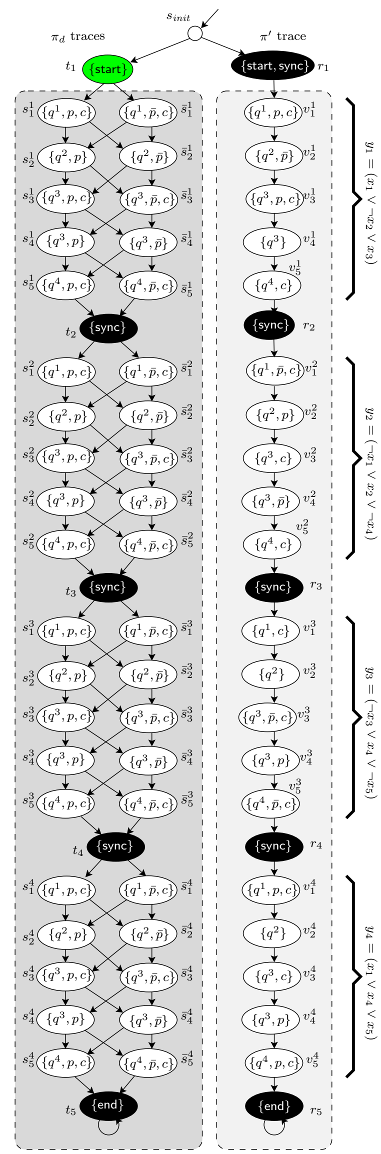

Fig. 9: Mapping the QBF formula as in Fig. 8 to an instance of $\mathsf{MC}[\exists(\exists/\forall)^k\mathsf{EE}]$.

Kripke structure and a A-HLTL formula with k quantifier alternations and two innermost existential trajectory quantifiers. Then, we show that the Kripke structure satisfies the A-HLTL formula if and only if the answer to the QBF problem is affirmative. Figure 9 shows an example of our mapping.

Kripke structure $\mathcal{K} = \langle S, s_{init}, \delta, L\rangle$**:**

- *(Atomic propositions* AP*)* For each alternation depth $d \in [1, k+1]$, we include an atomic proposition q^d. We furthermore include five atomic propositions: p is used to force clauses to become true if a Boolean variable appears in a clause; proposition $\bar{p}$ is used to force clauses to become true if the negation of a Boolean variable appears in a clause in our reduction; start marks the beginning of the gadget of states that represent the Boolean variables in the QBF instance; sync is used to mark the beginning a chain of states that represent a clause; end marks the terminal states, and c is used to enforce lock-step synchronization of Boolean variables in their respective clauses. Thus,
$$\mathsf{AP} = \big\{c, p, \bar{p}, \mathsf{start}, \mathsf{sync}, \mathsf{end}\big\} \ \cup\ \big\{q^d \mid d \in [1, k+1]\big\}.$$
- *(Set of states S)* We now identify the members of S:
  - First, we include an initial state s_{init}.
  - For each Boolean variable x_i, where $i \in [1, n]$ and clause y_j, where $j \in [1, m]$, we include two states s_i^j and $\bar{s}_i^j$. Each state s_i^j (respectively, $\bar{s}_i^j$) is labeled by p and $q^{d(x_i)}$ (respectively, $\bar{p}$ and $q^{d(x_i)}$), and
  - For each clause y_j, where $j \in [1, m]$:
    - ∗ We include states r_j and t_j, labeled by proposition sync. We also include states r_{m+1} and t_{m+1} labeled by end that marks the end of chain of states that represent the literals in the clauses.
    - ∗ We introduce the following n states:
$$\Big\{v_i^j \mid i \in [1, n]\Big\}.$$
      Each state v_i^j is labeled with propositions $q^{d(x_i)}$, and with p if x_i is a literal in y_j, or with $\bar{p}$ if $\neg x_i$ is a literal in y_j.
  - Finally, we label states s_i^j, $\bar{s}_i^j$, and v_i^j by proposition c, if i is odd, for all $i \in [1, n]$ and $j \in [1, m]$.

  Thus,
$$S = \big\{s_{init}\big\} \cup \big\{r_j, t_j \mid j \in [0, m+1]\big\} \ \cup\ \big\{v_i^j, s_i^j, \bar{s}_i^j \mid i \in [1, n] \wedge j \in [1, m]\big\}.$$
- *(Transition relation δ)* We now identify the members of δ:
$$\begin{aligned}\delta = &\big\{(s_{init}, r_1), (s_{init}, t_1)\} \ \cup\ \big\{(v_i^j, v_{i+1}^j) \mid i \in [1, n) \ \wedge\ j \in [1, m]\big\} \cup \\ &\big\{(s_i^j, s_{i+1}^j), (\bar{s}_i^j, s_{i+1}^j), (s_i^j, \bar{s}_{i+1}^j), (\bar{s}_i^j, \bar{s}_{i+1}^j) \mid i \in [1, n) \wedge j \in [1, m]\big\} \cup \\ &\big\{(s_n^j, t_{j+1}), (\bar{s}_n^j, t_{i+1}), (t_j, s_1^j), (t_j, \bar{s}_1^j) \mid i \in [1, n) \wedge j \in [1, m]\big\} \cup \\ &\big\{(v_j^n, r_{j+1}) \mid i \in [1, n] \wedge j \in [1, m)\big\} \cup \\ &\big\{(t_{m+1}, t_{m+1}), (r_{m+1}, r_{m+1})\big\}.\end{aligned}$$

A-HLTL formula: The A-HLTL formula in our mapping is the following:

$$\varphi_{\mathsf{map}} = \overbrace{\mathbb{Q}_1\pi_1.\mathbb{Q}_2\pi_2\cdots\mathbb{Q}_n}^{\mathcal{Q}\pi'}\pi_n.\mathsf{E}\tau.\mathsf{E}\tau'.$$

$$\bigwedge_{d\in[1,n]}\left(\underbrace{\Diamond(\mathsf{start}_{\pi_d,\tau}\ \wedge\ \mathsf{start}_{\pi',\tau}\ \wedge\ \mathsf{start}_{\pi_d,\tau'}\ \wedge\ \mathsf{start}_{\pi',\tau'})}_{\psi_1}\ \mathcal{U}\Big[\underbrace{(\mathsf{start}_{\pi_d,\tau}\ \wedge\ \mathsf{start}_{\pi',\tau}\ \wedge\ \neg\mathsf{start}_{\pi_d,\tau'}\ \wedge\ \neg\mathsf{start}_{\pi',\tau'})}_{\psi_2}\ \mathcal{U}\Big(\underbrace{\Big(\mathsf{start}_{\pi_d,\tau}\ \wedge\ \mathsf{start}_{\pi',\tau}\ \wedge\ \mathsf{sync}_{\pi_d,\tau'}\ \wedge\ \mathsf{sync}_{\pi',\tau'}\ \wedge (c_{\pi_d,\tau}\leftrightarrow c_{\pi_d,\tau'})\wedge(c_{\pi',\tau}\leftrightarrow c_{\pi',\tau'})\wedge (c_{\pi_d,\tau'}\leftrightarrow c_{\pi',\tau'})\wedge(c_{\pi_d,\tau}\leftrightarrow c_{\pi',\tau})\wedge (p_{\pi_d,\tau}\leftrightarrow p_{\pi_d,\tau'})\ \wedge(\bar{p}_{\pi_d,\tau}\leftrightarrow \bar{p}_{\pi_d,\tau'})\Big)}_{\psi_3}\ \mathcal{U}\ \underbrace{(\mathsf{end}_{\pi_d,\tau'}\wedge \mathsf{end}_{\pi',\tau'})}_{\psi_4}\Big)\Big]\right)$$

$$\bigwedge$$

$$\underbrace{\Box\left(\mathsf{sync}_{\pi',\tau'}\rightarrow\Big[\mathsf{sync}_{\pi',\tau}\ \mathcal{U}\ \Big((\neg\mathsf{sync}_{\pi',\tau}\ \wedge\ \neg\mathsf{end}_{\pi',\tau'})\ \mathcal{U}\ \bigvee_{d\in[1,n]}\Big(\big(q^d_{\pi_d,\tau'}\leftrightarrow q^d_{\pi',\tau'}\big)\wedge \big((p_{\pi',\tau'}\wedge p_{\pi_d,\tau'})\ \vee\ (\bar{p}_{\pi',\tau'}\wedge\bar{p}_{\pi_d,\tau'})\big)\Big)\Big)\Big]\right)}_{\psi_5}$$

Similar to the proof of Theorem 3 we add the quantifier on π' so that the number of alternations in $\mathbb{Q}_1\pi_1.\mathbb{Q}_2\pi_2\cdots\mathbb{Q}_n\pi_n.\mathsf{E}\tau.\mathsf{E}\tau'$ does not change. Again, note that since π' can only be instantiated with one path, namely, $s_{init}\cdots r_m$ the choice of the quantifier does not matter. Intuitively, this formula expresses the following. The structure for the π_d traces (see Fig. 9) is for choosing truth values `false` and `true`. Unlike in the proof of Theorem 3, the structure is now repeated for each clause. In principle, this allows for different choices in each clause; however, the A-HLTL formula ensures that the choices are consistent across all clauses. For this purpose, the A-HLTL formula uses the two existential trajectories $\mathsf{E}\tau$ and $\mathsf{E}\tau'$. Trajectory τ' steps through the π_d traces and the π' trace (representing the clauses) in lock step. Trajectory τ trails behind by exactly one clause. Let us explain the purpose of each sub-formula next:

- Sub-formula ψ_1 ensures that the both trajectories τ and τ' are initially positioned in states labeled by start.
- Sub-formula ψ_2 holds trajectory τ in states t_1 and r_1 until trajectory τ' advances to states t_2 and r_2.
- Once ψ_2 holds, sub-formula ψ_3 forces both trajectories to move in lock-step until trajectory τ' reaches end states; i.e., until sub-formula ψ_4 holds. Sub-formula ψ_3 also ensures that the values of p and $\bar{p}$ are chosen consistently between τ and τ'.
- Sub-formula ψ_5 requires trajectory τ' to make all clauses true. Notice that τ' visits all states (τ does not visit states of the last clause).

We now show that the given quantified Boolean formula is *true* if and only if the Kripke structure obtained by our mapping satisfies the A-HLTL formula φ_{map}.

(⇒) Suppose that y is true. Then, there is an instantiation of existentially quantified trace variables for each value of universally quantified variables and the trajectory quantifier, such that each clause y_j, where $j \in [1, m]$ becomes true (see Figs. 7 and 9 for an example). We now use these instantiations to instantiate each $\exists \pi_{x_d}$ in A-HLTL formula φ_{map}, where $d \in \{i \mid \mathbb{Q}_i = \exists\}$ as well as trajectories τ and τ' as follows. First, as mentioned earlier, π' can only be instantiated by the trace that reaches state r_{m+1}. Now, for each existentially quantified variable x_i, where $i \in [1, n]$, in depth $d \in [1, k+1]$, if $x_i = \mathtt{true}$, we instantiate π_d with a trace that includes state s_i^j for the clause y_j that contains literal x_i. Otherwise, the trace will include state $\bar{s}_i^j$. We now show that this trace instantiation evaluates formula φ_{map} to true. The trajectories can also be instantiated so that they synchronize the evaluation and consistency of truth values. Since each y_j is true, for any instantiation of universal quantifiers, there is at least one literal in y_j that is true. If this literal is of the form x_i, then we have $x_i = \mathtt{true}$ and trace π_d will include s_i^j, which is labeled by p and q^d. Hence, the values of p (respectively, q^d), in both π_d and π' instantiated by trace

$$s_{init} r_1 \cdots r_j v_1^j \cdots v_n^j \cdots r_{m+1}^{\omega}$$

are eventually equal. This is ensured by the instantiated trajectory τ. Similarly, if the literal in y_j is of the form $\neg x_i$, then $x_i = \mathtt{false}$ and, hence, some trace π_d will include $\bar{s}_i$. Again, the values of $\bar{p}$ (respectively, q^d), in both π_d and π' are eventually equal. Finally, since all clauses are true, all traces π' reach a state where the right side of the implication becomes true.

(⇐) Suppose our mapped Kripke structure satisfies the A-HLTL formula φ_{map}. This means that for each instantiation of the trajectories τ and τ, since trace π' is of the form $s_{init} r_1 \cdots r_j v_1^j \cdots v_n^j \cdots r_{m+1}^{\omega}$, then there exists a state v_i^j, where the values of q^d and either p or $\bar{p}$ are eventually equal to their counterparts in some trace π_d. This of course happens in some j gadget. If this trace is existentially quantified and includes s_i^j, then we assign $x_i = \texttt{true}$ for the preceding quantifications. If the trace includes $\bar{s}_i^j$, then $x_i = \texttt{false}$. Observe that since in no state p and $\bar{p}$ are simultaneously true and no trace includes both s_i and $\bar{s}_i$, variable x_i will have only one truth value. This is further ensured by the existence of trajectories τ and τ' that guarantee the consistency of truth values.

The argument to establish Π_{k+1}^p- and Π_{k+1}^p-hardness is similar to that of the proof of Theorem 3. □

Proof of Theorem 5

First, observe that $\mathsf{MC}[\exists\mathsf{EA}]$ is NP-complete. The upper bound is a trivial consequence of Theorem 2. The upper is also a trivial consequence of Theorem 4. Likewise, $\mathsf{MC}[\forall\mathsf{AE}]$ is coNP-complete.

In the following, we show that the base case $\mathsf{MC}[\forall^+\mathsf{E}^+\mathsf{A}^+]$ is Π_2^p-complete and, by duality, $\mathsf{MC}[\exists^+\mathsf{A}^+\mathsf{E}^+]^i$s Σ_2^p-complete. The complexity for formulas with additional path quantifiers then follows analogously to the previous theorems. The upper bounds for $\mathsf{MC}[\forall^+\mathsf{E}^+\mathsf{A}^+]$ and $\mathsf{MC}[\exists^+\mathsf{A}^+\mathsf{E}^+]$ follow from Theorem 1 (i.e., the soundness of our BMC algorithm for formulas with one alternation for trajectory quantifiers).

For the lower bound, we encode the satisfiability of a QBF formula $\mathbb{Q}_1 x_1.\mathbb{Q}_2 x_2 \ldots \mathbb{Q}_{n-1} x_{n-1}.\mathbb{Q}_n x_n.(y_1 \wedge y_2 \wedge \cdots \wedge y_m)$ with a *single* quantifier alternation such that $\mathbb{Q}_1 = \mathbb{Q}_2 = \ldots = \mathbb{Q}_k = \forall$ and $\mathbb{Q}_k = \mathbb{Q}_{k+1} = \ldots = \mathbb{Q}_n = \exists$ as a model checking problem of a $\forall^+\mathsf{E}^+\mathsf{A}^+$ A-HLTL formula. We choose a single alternation for simplicity. More alternations will follow in the same fashion as Theorems 3 and 4.

We modify the Kripke structure from the proof of Theorem 3 by adding a fresh path that starts with a state labeled $\{\mathsf{start}^2\}$ and then alternates for $n-k$ times between states labeled $\{p\}$ and states labeled $\{\bar{p}\}\}$. The role of this path is to encode the values for the existentially chosen variables $x_{k+1}, \ldots, x_n$. By aligning $\{p\}$ and $\{\bar{p}\}\}$ positions with the positions of the paths representing the truth assignments of the preceding quantifiers, the trajectory quantifier picks a truth assignment for the innermost quantifiers. This is reflected in the new A-HLTL formula φ_{map}:

$$\varphi_{\mathsf{map}} = \forall \pi_a \forall \pi_e \forall \pi'.\mathsf{E}\tau.\mathsf{A}\tau'.$$

$$\begin{aligned}
&\underbrace{\Big(\Diamond\, \mathsf{start}_{\pi_a,\tau} \wedge \Diamond\, \mathsf{start}^2_{\pi_e,\tau} \;\wedge\; \Diamond\, \mathsf{sync}_{\pi',\tau} \Big)}_{\psi_1} \\
&\rightarrow \\
&\Bigg(\underbrace{\Big((\neg \mathsf{start}_{\pi_a,\tau} \wedge \neg \mathsf{start}^2_{\pi_e,\tau})\, \mathcal{U}\, (\Box \mathsf{sync}_{\pi',\tau}) \Big)}_{\psi_2} \wedge \\
&\psi_3 \begin{cases}
\Box \Big(\neg \mathsf{start}_{\pi_a,\tau} \wedge c_{\pi_a,\tau} \wedge p_{\pi_e,\tau} \rightarrow (p_{\pi_e,\tau}\, \mathcal{U}\, \neg c_{\pi_a,\tau}) \Big) \wedge \\
\Box \Big(\neg \mathsf{start}_{\pi_a,\tau} \wedge \neg c_{\pi_a,\tau} \wedge p_{\pi_e,\tau} \rightarrow (p_{\pi_e,\tau}\, \mathcal{U}\, c_{\pi_a,\tau}) \Big) \wedge \\
\Box \Big(\neg \mathsf{start}_{\pi_a,\tau} \wedge c_{\pi_a,\tau} \wedge \bar{p}_{\pi_e,\tau} \rightarrow (\bar{p}_{\pi_e,\tau}\, \mathcal{U}\, \neg c_{\pi_a,\tau}) \Big) \wedge \\
\Box \Big(\neg \mathsf{start}_{\pi_a,\tau} \wedge \neg c_{\pi_a,\tau} \wedge \bar{p}_{\pi_e,\tau} \rightarrow (\bar{p}_{\pi_e,\tau}\, \mathcal{U}\, c_{\pi_a,\tau}) \Big) \wedge
\end{cases} \\
&\Box \Bigg[\underbrace{\Big(\mathsf{sync}_{\pi',\tau'} \wedge \big(\mathsf{sync}_{\pi',\tau'}\, \mathcal{U}\, (\neg \mathsf{sync}_{\pi',\tau'} \wedge (c_{\pi_a,\tau} \leftrightarrow c_{\pi',\tau'}))\, \mathcal{U}\, \mathsf{sync}_{\pi',\tau'} \big) \Big)}_{\psi_4} \\
&\rightarrow \\
&\psi_5 \begin{cases}
\Diamond \Big(\big(q^1_{\pi',\tau'} \wedge \big((p_{\pi',\tau'} \leftrightarrow p_{\pi_e,\tau}) \vee (\bar{p}_{\pi',\tau'} \wedge \bar{p}_{\pi_e,\tau}) \big) \big) \vee \\
\big(q^2_{\pi',\tau'} \wedge \big((p_{\pi',\tau'} \leftrightarrow p_{\pi_a,\tau}) \vee (\bar{p}_{\pi',\tau'} \leftrightarrow \bar{p}_{\pi_a,\tau}) \big) \big) \Big) \Bigg]
\end{cases} \Bigg)
\end{aligned}$$

Trace π_a holds the valuation of the universal variables, trace π_e together with trajectory τ the valuation of the existential variables. Trajectory τ aligns the clauses with the valuation (analogously to τ in the proof of Theorem 3). The intended purpose of the sub-formulas are as follows:

- Sub-formula ψ_1 initialize the paths at the right initial place to go through the clauses and propositional variables.
- The role of sub-formula ψ_2 is to create enough "slack" in the trajectory τ so that τ' can align the variables with every possible clause. For this purpose, τ waits until π' has reached the terminal state before advancing π_a and π_e.
- Sub-formula ψ_3 ensures that at all times, traces π_a and π_e toggle between c and $\neg c$. Furthermore, the four conjuncts ensure that trajectory τ only assigns a single valuation to each variable.
- Sub-formulas ψ_4 and ψ_5 have exactly the same role as in the mapping in Theorem 3: paths π_a (and π_e) and π' advance in lock step in trajectories τ and τ' to evaluate clauses.

The (if and only if) reduction are similar to that of Theorem 3. The complexity for formulas with additional path quantifiers then follows analogously to the previous theorems.

```
  if (y < size)
    temp &= B[A[y]*512]
```

(a) C program.

```
mov   size, %rax
mov   y, %rbx
cmp   %rbx, %rax
jbe   END
mov   A(%rbx), %rax
shl   $9, %rax
mov   B(%rax), %rax
and   %rax, temp
```

(b) Insecure translation.

```
mov   size, %rax
mov   y, %rbx
mov   $0, %rdx
cmp   %rbx, %rax
jbe   END
cmovbe   $-1, %rdx
mov   A(%rbx), %rax
shl   $9, %rax
or    %rdx,%rax
mov   B(%rax), %rax
or    %rdx,%rax
and   %rax, temp
```

(c) Secure translation.

Fig. 10: The speculative execution example, where the secure translation is applied load hardening technique to avoid information leaks.

B Details of our Case Studies

B.1 Counterexample for the Program in Fig. 6

One possible interleaving is as follows. First, `T3` executes lines 1–3 and set both `h` and `l` to 1. Next, since `h = l = 1`, `T1` runs lines 1–5, changes `x` to `1` while `y` remains `0`. At this moment, `T2` executes lines 1–3, which gives a sequence of outputs { `1`, `0` }. Then, `T1` resumes and changes `y` to `1`, so in the next iteration `T2` prints another sequence of outputs { `1`, `1` }, and all threads halt.

This sequence of outputs { `1`, `0`, `1`, `1` } printed by this specific scenario leaks the information that `h = 1`, because when `h = 0`, `x` can be set to `1` only if `y` has been set to `1` too. Since `T2` always prints `x` before `y`, so the output sequence { `1`, `0`, `1`, `1` } is not reproducible; hence, leaks information

Note that this specific information leak happens when the sequence of public inputs can be observed by the attacker. In other words, an attacker can correctly guess the value of `h` by observing this particular sequence of public inputs and outputs. That is, in order to correctly detect information leakage, one has to use a formula that aligns the public inputs and the observable outputs separately because the two alignments may "cross" with each other as illustrated in Section 1. As a result, two trajectory variables are needed in this case, as presented in the formula $\varphi_{\mathsf{OD}_{\mathsf{nd}}}$ below. Our encoding allows correct detection for this information leak due to concurrency (as shown in Table 3). The returned counterexample presents the concurrent bug where the secret inputs can be guessed by a malicious attacker from observing the sequences of public inputs and outputs.

B.2 Detail Explanation of Speculative Information Flows

The `C` program in Fig. 10a simply performs array data accessing with a given input index `y`, where the `if` statement checks if `y` stays in the ideal bound of array size or not. However, since low level assembly code requires more steps on memory storing and accessing using registers, an attacker might be able to guess the high-security values by comparing the speculative and non-speculative executions after translation (see Fig. 10b). In this case study, we evaluate 7 different versions of code where some of them are secure and some of them are insecure when doing speculative runs.

- $\texttt{SpecExcu}_{V1}$: *Insecure array value accessing.* The first version considers naive translation as shown in Fig. 10b. In non-speculative cases (i.e., $\texttt{y} < \texttt{size}$), the observations are identical. However, in speculative executions (i.e., $\texttt{y} \geq \texttt{size}$), lines 5 – 8 in the assembly code (see Fig. 10b) leak the value of the critical information (i.e., line 2 in the original `C` program). Thus, it violates SNI. Our BMC technique successfully returns UNSAT in this case, indicating a counterexample has been spotted.
- $\texttt{SpecExcu}_{V2}$: *Secure array value accessing with masking.* The second version shown in Fig. 10c overcomes the leaks from $\texttt{SpecExcu}_{V1}$ by applying the countermeasure of *speculative load hardening.* By adding an extra variable *mask* which the value is assigned on line 3 and 6, followed by the two `or` operations on lines 9 and 11, *mask* successfully hides the real value in a speculative execution which an attacker might try to access. Hence, it satisfies SNI. Our BMC algorithm returns SAT and shows the absence of a counterexample.

$\texttt{SpecExcu}_{V1}$: Insecure array value accessing

We first consider naive translation as showed in Fig. 10b. In non-speculative cases (i.e., $\texttt{y} < \texttt{size}$), the observations of executions are identical. However, when for speculative executions (i.e., $\texttt{y} \geq \texttt{size}$), line 5 – 8 will leak value of the critical information (i.e., line 2 in the c program). Thus it violates SNI and our solver successfully returns SAT, indicating a counterexample.

$\texttt{SpecExcu}_{V2}$: Secure array value accessing with masking

The second version showed in Fig. 10c overcomes the leaks from SE_{V1} by apply the countermeasure of *speculative load hardening.* By adding an extra variable *mask* which the value is assigned on line 3 and 6, where the `or` operations on line 9 and 11 hides the real value that a speculative execution might try to access. Hence, it satisfies SNI, where our solver returns UNSAT.

$\texttt{SpecExcu}_{V3}$: Speculative Load Hardening (insecure)

The third version refines V_1 with another if-statement. However, even with this extra conditional checking, the translation still violates speculative non-interference because the attacker can now know secret information about the content of `array A` in index `y`. (i.e., If it is equivalent to `k`) or not.

$\texttt{SpecExcu}_{V4}$: Branching (insecure)

The fourth version is investigating the usage of conditional operator. The compilation here translates the conditional operator directly into a branch instruction. This additional branching creates extra source for speculative execution and hence, leads to a potential speculative leak.

```
while (true){
  int x = 0;
  int k = 0;
  for k in range(8):
    if (k == 0 || k == ndet) then
      x = secret_in;
    else:
      x = x + x;
      k = k + 1;
    public_out(x % k)
  }
```

(a) Before loop peeling.

```
while (true) {
  int x = 0;
  int k = 0;
  if (k == 0 || k == ndet) then
    x = secret_in;
  else:
    x = x + x;
  k = k + 1;
  for k in range(8):
    if (k == 0 || k == ndet) then
      x = secret_in;
    else:
      x = x + x;
      k = k + 1;
    public_out(x % k)
  }
```

(b) After loop peeling.

Fig. 11: A compiler optimization example of loop peeling with nondeterministic input sequence.

$\texttt{SpecExcu}_{V5}$: Conditional Move (secure)
The fifth version conquers the problem in $\texttt{SpecExcu}_{V4}$. In this case, the conditional operators are always translated into conditional moves, instead of creating harmful branching. As a result, $\texttt{SpecExcu}_{V5}$ is secure under the speculative runs.

$\texttt{SpecExcu}_{V6}$: Pointer (insecure)
The last two versions $\texttt{SpecExcu}_{V6}$ and $\texttt{SpecExcu}_{V7}$ are considering how the input pointer provided by an attacker could cause speculative leaks. Essentially, $\texttt{SpecExcu}_{V6}$ assume the attacker specify the input as a pointer value, the memory access in the speculative run would be exploited by the attacker due to secondary memory access. In other words, the attacker is able to obtain the sensitive information by assigning a pointer input. As a result, insecure.

$\texttt{SpecExcu}_{V7}$: Pointer with Load Hardening (secure)
On the other hand, $\texttt{SpecExcu}_{V7}$ resolves this problem by performing hardening. In this way, no harmful information flow would happen in the speculative runs. Hence, the program is proved secure.

B.3 Secure Compiler Optimization

In Fig. 11, if the outermost while-loop executes only one time, the alignment of public outputs requires only one existential trajectory (i.e., φ_{SC}). Figure 11 peels off the first iteration of a for-loop (LP).

However, when the outermost while loop execute several times and the inputs are read from asynchronous channels, two trajectories are required to check conformance of source and target (i.e., $\varphi_{\mathsf{SC}_{\mathsf{nd}}}$). We also notice that our trajectory encoding is able to reproduce the same verification outcomes as introduced in case studies in [1] with better performance in time

T1

```
1 while true do:
2   if (secret == low):
3     highArray = new Array[M]
4     fillArray(highArray)
5   else:
6     for i in [1..m+p]:
7       skip
```

T2

```
1 while true do:
2   for i in [1..n]:
3     skip
4   readArray(lowArray)
5   out(1)
```

T3

```
1 while true do:
2   for i in [1..n+m]:
3     skip
4   readArray(lowArray)
5   out(0)
```

T4

```
1 while true do:
2   secret = 0 || 1
3   low    = 0 || 1
```

Fig. 12: Cache-based timing attack with nondeterministic inputs from thread T4.

B.4 Cache-based Timing Attacks

We consider a 4-threaded program as presented in Fig. 12 The three threads `T1`, `T2`, and `T3` are interacting with the cache when they execute `fillArray()` or `readArray()`. The counters n, m, and p are representing the numbers of execution steps for threads `T1`, `T2`, and `T3`, respectively, to schedule the threads in the order of `T1`, `T2`, and `T3`.

Assume the cache size is $M = 2$ and is originally filled with the value of the low array. When `secret` does not equal to `low`, the cache will remain with low array data (i.e., `T1` doesn't execute line 3–4). Hence, when `T2` starts executing line 4, it only needs to read the data which takes two steps (since $M = 2$), and then outputs 1. Afterward, `T3` reads the data in line 4 again, and outputs 0, which leads to the final sequence of outputs $\{1, 0\}$. Now, consider when `secret` equals to `low`, which makes the cache filled with high-array data (i.e., `T1` executes line 3–4). In this case, `T2` will take longer time to finish line 4, because of the necessary operations of evicting the high-array data, which adds extra steps for `T2` to finish line 4 compare to the previous scenario. Hence, `T3` will start before `T2` finished, and output 0 before `T2` eventually output 1. That is, an output sequence $\{0, 1\}$. The two scenarios are presented in Fig. 13

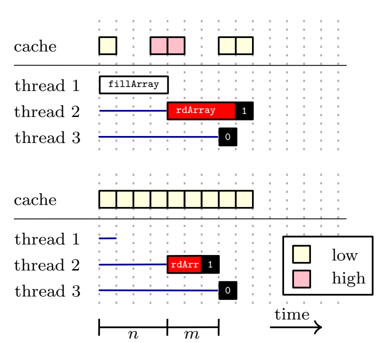

Fig. 13: A Cache-based timing attack.

We modify this original example in [18] with another source of nondeterminism, the input channel `T4`, to show that two trajectories are needed in this case in order to correctly identify a cache-based timing attack. With `T4`, one has to align the inputs first before aligning the outputs from `T2` and `T3` when there are sequences of inputs and outputs (due to the loops).