A Temporal Logic for Strategic Hyperproperties

Raven Beutner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

Hyperproperties are commonly used in computer security to define information-flow policies and other requirements that reason about the relationship between multiple computations. In this paper, we study a novel class of hyperproperties where the individual computation paths are chosen by the strategic choices of a coalition of agents in a multi-agent system. We introduce `HyperATL`*, an extension of computation tree logic with path variables and strategy quantifiers. `HyperATL`* can express strategic hyperproperties, such as that the scheduler in a concurrent system has a *strategy* to avoid information leakage. `HyperATL`* is particularly useful to specify *asynchronous* hyperproperties, i.e., hyperproperties where the speed of the execution on the different computation paths depends on the choices of the scheduler. Unlike other recent logics for the specification of asynchronous hyperproperties, our logic is the first to admit decidable model checking for the full logic. We present a model checking algorithm for `HyperATL`* based on alternating word automata, and show that our algorithm is asymptotically optimal by providing a matching lower bound. We have implemented a prototype model checker for a fragment of `HyperATL`*, able to check various security properties on small programs.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking

Keywords and phrases hyperproperties, temporal logic, alternating-time temporal logic, model checking, multi-agent systems, information flow, asynchronous hyperproperties

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.24

Related Version *Full Version*: `https://arxiv.org/abs/2107.02509` [6]

Funding This work was partially supported by the German Research Foundation (DFG) as part of the Collaborative Research Center “Foundations of Perspicuous Software Systems” (TRR 248, 389792660), by the European Research Council (ERC) Grant OSARES (No. 683300) and by the Saarbrücken Graduate School of Computer Science.

1 Introduction

Hyperproperties [10] are system properties that specify a relation between the traces of the system. Such properties are of increasing importance as they can, for example, characterize the information-flow in a system [38]. Consequently, several logics for the specification of hyperproperties have been developed, including hyper variants of `CTL`*(and `LTL`) [9, 38], `PDL`-Δ [24] and `QPTL` [18]. A prominent example is the temporal hyperlogic `HyperLTL` [9], which extends linear-time temporal logic (`LTL`) [35] with explicit trace quantification. In `HyperLTL` we can, for instance, express non-interference (NI), i.e., the requirement that the observable output of a system does not depend on high-security inputs [23]. A prominent formulation of NI for non-deterministic systems is *generalized non-interference* (GNI) [31, 12], which can be expressed as the `HyperLTL` formula

$$\forall \pi_1.\ \forall \pi_2.\ \exists \pi_3.\ \Box(\bigwedge_{a \in H} a_{\pi_1} \leftrightarrow a_{\pi_3}) \land \Box(\bigwedge_{a \in O} a_{\pi_2} \leftrightarrow a_{\pi_3}),$$

where H and O are two sets of propositions, with H representing the high-security input and O the output. The formula states that for any pair of traces π_1, π_2 there exists a third trace that agrees on the high-security inputs with π_1 and on the outputs with π_2 (for simplicity we assume that no low-security inputs are present). The existence of such a trace guarantees that any observation made on the outputs is compatible with every possible sequence of high-security inputs. The non-determinism is thus the sole explanation for the system output.

In this paper, we introduce a novel class of hyperproperties that reason about *strategic behavior* in a multi-agent system. As a motivation for why strategic hyperproperties are desirable, consider *GNI* from above. As `HyperLTL` only quantifies existentially or universally over the paths in the system, the entire system is treated either as fully controllable or fully adversarial. Moreover, the witness trace π_3 can be constructed with full knowledge of both π_1 and π_2; this means that the entire output and input history can be used to resolve the non-determinism of the system appropriately. Now consider a system where the non-determinism arises from a scheduling decision between two possible subprograms P_1, P_2. Each subprogram reads the next input `h` of the system. Suppose that P_1 assumes that `h` is even and otherwise leaks information, while P_2 assumes that `h` is odd and otherwise leaks information. In the trace-based view of *GNI*, the witness trace π_3 is fixed knowing the entire *future* input sequence, allowing the construction of a leakage-avoiding path π_3; The system satisfies *GNI*. An *actual* scheduler, who chooses which of P_1, P_2 handles the next input, can only avoid a leakage if it knows what the *next* input will be, which is impossible in a real-world system. The `HyperLTL` formulation of *GNI* is, in this case, unable to express the desired property. In our scenario, we need to reason about the *strategic* behaviour of the system, i.e., we want to check if there exist a strategy for the scheduler that avoids leakage.

Strategic Hyperproperties. Reasoning about strategic behavior in multi-agent systems has been studied before. The seminal work on alternating-time temporal logic [1] introduced an extension of `CTL` (and `CTL`*[14]) that is centred around the idea of viewing paths as the outcome of a game, where some agents are controlled via a strategy. The `ATL`* quantifier $\langle\langle A\rangle\rangle\varphi$ requires the agents in A to have a strategy that enforces the path formula φ to become true. This makes `ATL`* an ideal logic for reasoning about *open* systems, where one is less interested in the pure existence of a path, but rather in the actual realizability of an outcome in a multi-agent system. `ATL` has numerous variations and extensions, which, for example, introduce knowledge modalities [42] or imperfect observation [5]. While strategy quantifiers in `ATL`* can be nested (like in `CTL`*), the logic is still unable to express hyperproperties, as the scope of each quantifier ends with the beginning of the next (see [16]).

It is very useful to reason about the strategic behaviour of the agents in a multi-agent system with respect to a hyperproperty. In the example above, one would like to ask if the scheduler has a *strategy* (based on the finite history of inputs only) such that unintended information-flow (which is a hyperproperty) is prevented (in the above example such an answer should be negative). There exist multiple angles to approach this: One could, for instance, interpret strategic hyperproperties such that a coalition of agents tries to achieve a set of outcomes satisfying some hyperproperty (expressed, for example, in `HyperLTL`). Model checking the resulting logic would then subsume realizability of `HyperLTL`, which is *undecidable* even for simple alternation-free formulas [19].

In this paper, we introduce a new temporal logic, called `HyperATL`*, that combines the strategic behaviour in multi-agent systems with the ability to express hyperproperties. Crucially, we focus on the strategic behaviour of a coalition of agents along a *single path*, i.e., we view path quantification as the outcome of a game. Syntactically, we follow a similar

$$\begin{tikzcd}
 & & \texttt{HyperATL}^* & & \\
 & \texttt{HyperCTL}^* \arrow[ur] & & \texttt{ATL}^* \arrow[ul] & \\
\texttt{HyperLTL} \arrow[ur] & & \texttt{CTL}^* \arrow[ul] \arrow[ur] & & \texttt{ATL} \arrow[ul] \\
 & \texttt{LTL} \arrow[ul] \arrow[ur] & & \texttt{CTL} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

(a)

```
while(true)
  h ← read_H()
  if (h mod 2 = 0 ) then
    o ← !o
  else
    temp ← o = 0 ? 1 : 0
    o ← temp
```

(b)

Figure 1 (a): Expressiveness of temporal logics. An arrow $A \to B$ indicates that A is a syntactic fragment of B. **(b)**: Example program that violates (synchronous) observational-determinism.

approach as alternating-time temporal logic [1]. We use the strategy quantifier $\langle\langle A \rangle\rangle \pi.\varphi$ to specify that the agents in A have a strategy such that each possible outcome, when bound to the path variable π, satisfies φ. A formula of the form $\langle\langle A_1 \rangle\rangle \pi_1.\langle\langle A_2 \rangle\rangle \pi_2.\varphi$ now requires the existence of strategy for the agents in A_1 such that for all possible outcomes of the game π_1, the agents in A_2 have a strategy such that for all possible outcomes π_2, the combination of π_1, π_2 satisfies φ (which is a formula that can refer to propositions on paths π_1, π_2). The strategic behaviour chosen by each quantifier is thus limited to the current path and can be based on the already *fixed* outcomes of outer quantifiers (i.e., the entire strategy for the agents in A_2 can depend on the full outcome of π_1). Sometimes, however, it is useful not to reason incrementally about the strategy for a single path at a time, but rather to reason about a *joint* strategy for multiple paths. To express this, we endow our logic with an explicit construct to resolve the games in parallel (syntactically, we surround quantifiers by $[\cdot]$ brackets). The formula $[\langle\langle A_1 \rangle\rangle \pi_1.\langle\langle A_2 \rangle\rangle \pi_2.]\ \varphi$ requires winning strategies for the agents in A_1 (for the first copy) and for A_2 (for the second copy) where the strategies can observe the current state of *both* copies. This enables collaboration between the agents in A_1 and A_2.

Similar to ATL*, the empty (resp. full) agent set corresponds to universal (resp. existential) quantification. HyperATL* therefore subsumes HyperCTL* (and thus HyperLTL) as well as ATL*. The logic is thus a natural extension of both the temporal logics for hyperproperties and the alternating-time logics from the non-hyper realm (see Fig. 1a).

Strategic Non-Interference. Consider again the example of *GNI* expressed in HyperLTL. In HyperATL*, we can express a more refined, strategic notion of non-interference, that postulates the existence of a *strategy* for the non-determinism. As a first step, we consider a program no longer as a Kripke structure (a standard model for temporal hyperlogics), but as a game structure played between two players. Player ξ_N is responsible for resolving the non-determinism of the system, and player ξ_H is responsible for choosing the high-security inputs to the system. We can now express that ξ_N has a strategy to produce matching outputs (without knowing the future inputs by ξ_H). Consider the following formula *stratNI*:

$$\forall \pi_1.\ \langle\langle \{\xi_N\} \rangle\rangle \pi_2.\ \Box\Big(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2} \Big)$$

This formula requires that for every possible reference path π_1, the non-determinism always has a *strategy* to produce identical outputs. One can show that *stratNI* implies *GNI*: The existence of a leakage "disproving" strategy implies the existence of a leakage "disproving" trace. A particular strength of this formulation is that we can encode additional requirements on the strategy. For example: if the internal non-determinism arises from the scheduling decisions between multiple components, we can require fairness of the scheduling strategy.

Asynchronous Hyperproperties. Strategic hyperproperties are also very natural to express *asynchronous hyperproperties.* While existing hyperlogics traverse the traces of a system synchronously, one often requires an asynchronous traversal to account, for example, for the unknown speed of execution of software that runs on some unknown platform. In a multi-agent system, the scheduling decision (i.e., whether a system progresses or remains in its current state) can then be seen as the decision made by scheduling agent (called *sched* in the following). If not already present, we can artificially add such a scheduling agent via a system transformation. By either including or excluding this agent in a strategy quantifier, we can then naturally reason about asynchronous executions of programs. Instead of reasoning about the asynchronous scheduling of a system directly, we thus reason about the existence of a strategy for the scheduling agent.

As an example consider the program in Fig. 1b, which continuously reads an input and flips the output `o` either directly, or via a temporary variable. Based on the input, the exact time point of the change in `o` differs. A synchronous formulation of *observational-determinism* (OD) [26], which requires the output to be identical on all traces, does not hold. In `HyperATL`*, we can naturally express a variant of OD where we search for a strategy for the scheduling agent *sched*, who aligns the outputs on both traces by stuttering them appropriately:

$$[\langle\!\langle \{sched\} \rangle\!\rangle \pi_1.\ \langle\!\langle \{sched\} \rangle\!\rangle \pi_2.]\ \Box(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2})$$

The program in Fig. 1b (with an added asynchronous scheduler) satisfies this variant, because *sched* can stutter the change in `o` in order to align with the second trace.

To demonstrate the expressiveness of this strategic view on asynchronous hyperproperties, we compare our approach to `AHLTL`, a recent temporal logic for asynchronous hyperproperties [4]. While `AHLTL` model checking is undecidable in general, recent work [4] has identified a large fragment for which model checking is possible. We show that this fragment can be encoded within `HyperATL`*. Every property in this (largest known) decidable fragment can thus be expressed in `HyperATL`*, for which model checking is decidable for the *full* logic.

Model Checking. We show that model checking of `HyperATL`* on concurrent game structures is decidable and present an automata-theoretic algorithm. Our algorithm incrementally reduces model checking to the emptiness of an automaton. We show that alternating automata are well suited to keep track of all possible path assignments satisfying a formula by encoding the game structure in the transition function of the automaton. We characterize the model checking complexity in terms of the number of complex quantifiers (where the agent team is non-trivial) and simple quantifiers (i.e., $\exists$ or $\forall$). We provide a lower bound, based on a novel construction that encodes a doubly exponential counter within a single strategy quantifier, that (in almost all cases) matches the upper bound from our algorithm.

Prototype Model Checker. On the practical side, we present a prototype model checker for an efficient fragment of `HyperATL`* by reducing the model checking to solving of a parity game. The fragment supported by our tool does, in particular, include all alternation free `HyperLTL` formulas [20], the $\forall^*\exists^*$-model checking approach from [12] as well as all formulas in the decidable fragment of `AHLTL` [4].

Contributions. In summary, our contributions include the following:

- We introduce a novel logic to express strategic hyperproperties and demonstrate that it is well suited to express, e.g., information-flow control and, in particular, asynchronous hyperproperties.

- We give an automata-based model checking algorithm for our logic and provide a lower bound on the model checking problem.
- We show that our logic can express all formulas in the largest known decidable fragment of the existing hyperlogic `AHLTL` [4].
- We provide a prototype-model checker for an efficiently checkable fragment of `HyperATL`* and use it to verify information-flow polices and asynchronous hyperproperties.

2 Preliminaries

In this section, we introduce some basic preliminaries needed in the following.

Concurrent Game Structure. As our model of multi-agent systems, we consider concurrent game structures (CGS) [1]. The transition relation in a CGS is based on the decision by individual agents (or players). Formally, a CGS is a tuple $\mathcal{G} = (S, s_0, \Xi, \mathcal{M}, \delta, \mathbf{AP}, L)$ where S is a finite set of states, $s_0 \in S$ the initial state, Ξ a finite set of agents and $\mathcal{M}$ a finite set of moves. We call a function $\sigma : \Xi \to \mathcal{M}$ a global move vector and for a set of agent $A \subseteq \Xi$ a function $\sigma : A \to \mathcal{M}$ a partial move vector. $\delta : S \times (\Xi \to \mathcal{M}) \to S$ is a transition function that maps states and move vectors to successor states. Finally, $\mathbf{AP}$ is a finite set of propositions and $L : S \to 2^{\mathbf{AP}}$ a labelling function. Note that every Kripke structure (a standard model for temporal logics [3]) can be seen as a 1-player CGS. For disjoint sets of agents A_1, A_2 and partial move vectors $\sigma_i : A_i \to \mathcal{M}$ for $i \in \{1, 2\}$ we define $\sigma_1 + \sigma_2$ as the move vector obtained as the combination of the individual choices. For $\sigma : A \to \mathcal{M}$ and $A' \subseteq A$, we define $\sigma_{|A'}$ as the move vector obtained by restring the domain of σ to A'.

In a concurrent game structure (as the name suggests) all agents choose their next move concurrently, i.e., without knowing what moves the other player have chosen. We introduce the concept of multi-stage CGS (MSCGS), in which the move selection proceeds in stages and agents can base their decision on the already selected moves of (some of the) other agents. This is particularly useful when we, e.g., want to base a scheduling decision on the moves selected by the other agents. Formally, a MSCGS is a CGS equipped with a function $d : \Xi \to \mathbb{N}$, that orders the agents according to informedness. Whenever $d(\xi_1) < d(\xi_2)$, ξ_2 can base its next move on the move selected by ξ_1. A CGS thus naturally corresponds to a MSCGS with $d = \mathbf{0}$, where $\mathbf{0}$ is the constant 0 function.

Alternating Automata. An alternating parity (word) automaton (APA) is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \rho, c)$ where Q is a finite set of states, $q_0 \in Q$ an initial state, Σ a finite alphabet, $\rho : Q \times \Sigma \to \mathbb{B}^+(Q)$ a transition function ($\mathbb{B}^+(Q)$ is the set of positive boolean combinations of states) and $c : Q \to \mathbb{N}$ a colouring of nodes with natural numbers. For $\Psi \in \mathbb{B}^+(Q)$, $B \subseteq Q$ we write $B \models \Psi$ if the assignment obtained from B satisfies Ψ. A tree is a set $T \subseteq \mathbb{N}^*$ that is prefixed closed, i.e., $\tau \cdot n \in T$ implies $\tau \in T$. We refer to elements in $\tau \in T$ as nodes and denote with $|\tau|$ the length of τ (or equivalently the depth of the node). For a node $\tau \in T$ we denote with $\mathit{children}(\tau)$ the immediate children of τ, i.e., $\mathit{children}(\tau) = \{\tau \cdot n \in T \mid n \in \mathbb{N}\}$. A X-labelled tree is a pair (T, r) where T is a tree and $r : T \to X$ a labelling with X. A run of an APA $\mathcal{A} = (Q, q_0, \Sigma, \rho, c)$ on a word $u \in \Sigma^\omega$ is a Q-labelled tree (T, r) that satisfies the following: **(1)** $r(\epsilon) = q_0$, **(2)** For all $\tau \in T$, $\{r(\tau') \mid \tau' \in \mathit{children}(\tau)\} \models \rho(r(\tau), u(|\tau|))$. A run (T, r) is accepting if for every infinite path π in T the minimal colour (given by c) that occurs infinitely many times is even. We denote with $\mathcal{L}(\mathcal{A})$ the set of words for which $\mathcal{A}$ has an accepting run. We call an alternating automaton $\mathcal{A}$ non-deterministic (resp. universal) if the transition function δ is a disjunction (resp. conjunction) of states. If δ is just a single state, we call $\mathcal{A}$ deterministic. Crucially alternating, non-deterministic, universal and deterministic parity automaton are all equivalent in the sense that they accept the same class of languages (namely ω-regular ones) although they can be (double) exponentially more succinct:

▶ **Theorem 1** ([33, 13]). *For every alternating parity automaton $\mathcal{A}$ with n states, there exists a non-deterministic parity automaton $\mathcal{A}'$ with $2^{\mathcal{O}(n)}$-states that accepts the same language. For every non-deterministic or universal parity automaton $\mathcal{A}$ with n states, there exists a deterministic parity automaton $\mathcal{A}'$ with $2^{\mathcal{O}(n)}$-states that accepts the same language.*

▶ **Theorem 2** ([29]). *For every alternating parity automaton $\mathcal{A}$ with n states, there exists an alternating parity automaton $\overline{\mathcal{A}}$ with $\mathcal{O}(n^2)$-states that accepts the complemented language.*

3 HyperATL*

In this section, we introduce `HyperATL`*. Our logic extends the standard temporal logic `CTL`* [14] by introducing path variables and strategic quantification [1]. Assume a countably infinite set of path variables *Var*, a set of agents Ξ and a set of atomic propositions $\mathbf{AP}$. `HyperATL`* formulas are generated by the following grammar

$$\varphi := \langle\!\langle A \rangle\!\rangle \pi.\varphi \mid a_\pi \mid \varphi \land \varphi \mid \neg\varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \bigcirc \varphi$$

where $\pi \in \mathit{Var}$ is a path variable, $a \in \mathbf{AP}$ an atomic proposition and $A \subseteq \Xi$ a set of agents. As in `HyperCTL`*, a_π means that proposition a holds in the current step on path π. Via $\langle\!\langle A \rangle\!\rangle \pi.\varphi$ we can quantify over paths in a system (which we consider as the outcome of a game). $\langle\!\langle A \rangle\!\rangle \pi.\varphi$ requires the agents in A to have a *strategy* (defined below) such that each outcome under that strategy, when bound to trace variable π, satisfies φ. We abbreviate as usual $\varphi_1 \lor \varphi_2 := \neg(\neg\varphi_1 \land \neg\varphi_2)$, and the temporal operators globally ($\square$), eventually ($\diamondsuit$) and release ($\mathcal{R}$). Trivial agent sets, i.e., $A = \emptyset$ or $A = \Xi$ correspond to classical existential or universal quantification. We therefore write $\forall\pi$ instead of $\langle\!\langle \emptyset \rangle\!\rangle \pi$ and $\exists\pi$ instead of $\langle\!\langle \Xi \rangle\!\rangle \pi$. We call a quantifier *simple* if the agent-set is trivial and otherwise *complex*. We call a formula *linear* if it consists of an initial quantifier prefix followed by a quantifier-free (`LTL`) formula.

Semantics. Let us fix a MSCGS $\mathcal{G} = (S, s_0, \Xi, \mathcal{M}, \delta, d, \mathbf{AP}, L)$ as a model. We first need to formalize the notion of a strategy in the game structure. A strategy for any agent is a function that maps finite histories of plays in the game to a move in $\mathcal{M}$. As the plays in an MSCGS progress in stages, the decision can be based not only on the past sequence of states, but also on the fixed moves of all agents in previous stages. Formally, a strategy for an agent ξ is a function $f_\xi : S^+ \times (\{\xi' \mid d(\xi') < d(\xi)\} \to \mathcal{M}) \to \mathcal{M}$. Given a set of agents A, a set of strategies $F_A = \{f_\xi \mid \xi \in A\}$ and a state $s \in S$, we define $\mathit{out}(\mathcal{G}, s, F_A)$ as the set of all runs $u \in S^\omega$ such that **(1)** $u(0) = s$ and **(2)** for every $i \in \mathbb{N}$ there exists a global move vector σ with $\delta(u(i), \sigma) = u(i+1)$ and for all $\xi \in A$ we have $\sigma(\xi) = f_\xi(u[0,i], \sigma_{|\{\xi' | d(\xi') < d(\xi)\}})$. The agents in A choose their move in each step based on the finite history of the play and the decision of all other agents in an earlier stage. Note that in case where $d = \mathbf{0}$, a strategy is just a function $S^+ \to \mathcal{M}$, ignoring the moves selected by other agents.

The semantics of a formula is now defined in terms of a path assignment $\Pi : \mathit{Var} \to S^\omega$, mapping path variables to infinite sequences of states in $\mathcal{G}$. For a path $t \in S^\omega$ we write $t[i, \infty]$ to refer to the infinite suffix of t starting at position i. We write $\Pi[i, \infty]$ to denote the path assignment defined by $\Pi[i, \infty](\pi) = \Pi(\pi)[i, \infty]$. We can then inductively define the satisfaction relation for `HyperATL`*:

$\Pi \models_\mathcal{G} a_\pi$	iff $a \in L(\Pi(\pi)(0))$
$\Pi \models_\mathcal{G} \neg\varphi$	iff $\Pi \not\models_\mathcal{G} \varphi$
$\Pi \models_\mathcal{G} \varphi_1 \land \varphi_2$	iff $\Pi \models_S \varphi_1$ and $\Pi \models_\mathcal{G} \varphi_2$
$\Pi \models_\mathcal{G} \varphi_1 \,\mathcal{U}\, \varphi_2$	iff $\exists i \geq 0 . \Pi[i, \infty] \models_\mathcal{G} \varphi_2$ and $\forall 0 \leq j < i . \Pi[j, \infty] \models_\mathcal{G} \varphi_1$
$\Pi \models_\mathcal{G} \bigcirc\varphi$	iff $\Pi[1, \infty] \models_\mathcal{G} \varphi$
$\Pi \models_\mathcal{G} \langle\!\langle A\rangle\!\rangle\pi.\, \varphi$	iff $\exists F_A : \forall t \in out(\mathcal{G}, \Pi(\epsilon)(0), F_A) : \Pi[\pi \mapsto t] \models_\mathcal{G} \varphi$

Here $\Pi(\epsilon)$ refers to the path that was last added to the assignment (similar to the HyperLTL-semantics [9]). If Π is the empty assignment, we define $\Pi(\epsilon)(0)$ as the initial state s_0 of $\mathcal{G}$. Note that the games are local to each path but based on all outer paths.: In a formula of the form $\forall\pi_1.\langle\!\langle A\rangle\!\rangle\pi_2.\varphi$ the agents in A know the already fixed, *full* trace π_1 but behave as a strategy w.r.t. π_2. We write $\mathcal{G} \models \varphi$ whenever $\emptyset \models_\mathcal{G} \varphi$ where $\emptyset$ is the empty path assignment.

▶ **Proposition 3.** *HyperATL* subsumes HyperCTL* (and thus HyperLTL) and ATL* (see Fig. 1a).*

We sometimes consider HyperATL* formulas with extend path quantification: We write $\langle\!\langle A\rangle\!\rangle_\mathcal{G}\, \pi.\varphi$ to indicate that the path π is the result of the game played in $\mathcal{G}$. We can thus refer to different structures in the same formula. For example, $\forall_{\mathcal{G}_1} \pi_1.\ \langle\!\langle A\rangle\!\rangle_{\mathcal{G}_2} \pi_2.\ \Box(o_{\pi_1} \leftrightarrow o_{\pi_2})$ states that for each path π_1 in $\mathcal{G}_1$ the agents in A have a strategy in $\mathcal{G}_2$ that produces the same outputs as on π_1. As for HyperLTL, extended quantification reduces to the standard semantics [38, §5.4].

Parallel-Composition. We extend HyperATL* with a *syntactic* construct that allows multiple traces to be resolved in a single bigger game, where individual copies of the system progress in parallel. Consider the following modification to the HyperATL* syntax, where $k \geq 1$:

$$\varphi := [\langle\!\langle A_1\rangle\!\rangle\pi_1.\ \cdots\ \langle\!\langle A_k\rangle\!\rangle\pi_k.]\ \varphi \mid a_\pi \mid \neg\varphi \mid \varphi \land \varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \bigcirc\varphi$$

When surrounding strategy quantifiers by $[\cdot]$ the resulting traces are the outcome of a game played on a bigger, parallel game of the structure. This way, the agents in each copy can base their decisions not only on the current state of their copy but on the combined state of all k copies (which allows for a coordinated behaviour among the copies). For a player ξ, and a CGS $\mathcal{G} = (S, s_0, \Xi, \mathcal{M}, \delta, \mathbf{AP}, L)$, a k-strategy for ξ is a function $f_\xi : (S^k)^+ \to \mathcal{M}$. The strategy can thus base its decision on a finite history for each of the k copies. For a system $\mathcal{G}$, sets of k-strategies strategies $F_{A_1}, \cdots, F_{A_k}$ and states $s_1, \cdots, s_k$, we define $out(\mathcal{G}, (s_1, \cdots, s_k), F_{A_1}, \cdots, F_{A_k})$ as all plays $u \in (S^k)^\omega$ such that **(1)** $u(0) = (s_1, \cdots, s_k)$ and **(2)** for every $i \in \mathbb{N}$ there exist move vectors $\sigma_1, \cdots, \sigma_k$ such that $u(i+1) = (\delta(t_1, \sigma_1), \cdots, \delta(t_k, \sigma_k))$ where $u(i) = (t_1, \cdots, t_k)$ and for every $j \in \{1, \cdots, k\}$, agent $\xi \in A_j$ and strategy $f_\xi \in F_{A_j}$, $\sigma_j(\xi) = f_\xi(u[0, i])$. Agents can thus control the individual progress of their system and base their decision on the history of the other quantifiers. Note that in the case where $k = 1$ this is identical to the construction seen above. For simplicity, we gave the semantics for a CGS (i.e., a MSCGS without stages), it can be generalized easily. We can now extend our semantics by

$$\Pi \models_\mathcal{G} [\langle\!\langle A_1\rangle\!\rangle\pi_1.\ \cdots\ \langle\!\langle A_k\rangle\!\rangle\pi_k.]\ \varphi \text{ iff}$$
$$\exists F_{A_1}, \cdots, F_{A_k} : \forall(t_1, \cdots, t_k) \in out(\mathcal{G}, (\Pi(\epsilon)(0), \cdots, \Pi(\epsilon)(0)), F_{A_1}, \cdots, F_{A_k}) : \Pi[\pi_i \mapsto t_i]_{i=1}^k \models_\mathcal{G} \varphi$$

Note that $[\langle\!\langle A\rangle\!\rangle\pi.]\ \varphi$ is equivalent to $\langle\!\langle A\rangle\!\rangle\pi.\varphi$. This does, of course, not hold once we consider multiple strategy quantifiers grouped together by $[\cdot]$.

Comparison with $\forall\exists$-HyperLTL model checking [12]. To give some more intuition for the parallel composition, we can compare our $[\cdot]$-construct with the model checking algorithm introduced in [12]. The idea of the method from [12] is to check a $\forall\exists$-formula by viewing the existential quantifier as a player who has to decide on a next state (in her copy) by reacting to the moves of the universal quantifier. If such a winning strategy exists, the $\forall\exists$-formula also holds, whereas the absence of a winning strategy does, in general, *not* imply that the formula is invalid (as the strategy bases its decision on finite plays whereas the existential path is chosen with the universally quantified path already fixed). This game based view of the existential player can be natively expressed in HyperATL*: While the HyperATL*-formula $\forall\pi_1.\exists\pi_2.\varphi$ is equivalent to the same HyperLTL-formula (i.e., the existential trace π_2 is chosen knowing the entire trace π_1), model checking of the formula $[\forall\pi_1.\exists\pi_2].\varphi$ corresponds to the strategy search for the existential player that is only based on finite prefixes of π_1 (which directly corresponds to [12]). We can actually show that if any MSCGS $\mathcal{G}$ satisfies $[\forall\pi_1.\langle\langle A\rangle\rangle\pi_2.]\varphi$ then it also satisfies $\forall\pi_1.\ \langle\langle A\rangle\rangle\pi_2.\varphi$ (see [6]). This gives a more general proof of the soundness of [12]. As our prototype implementation supports $[\langle\langle A_1\rangle\rangle\pi_1.\langle\langle A_2\rangle\rangle\pi_2.]$-formulas, our tool subsumes the algorithm in [12] (see Sec. 8).

4 Examples of Strategic Hyperproperties

After having introduced the formal semantics of HyperATL*, we now demonstrate how the strategic quantification can be useful for expressing hyperproperties. We organize our example in two categories. We begin with examples from information-flow control and highlight the correspondence with existing properties and security paradigms. Afterwards, we show how the strategic hyperproperties are naturally well suited to express asynchronous hyperproperties.

4.1 Strategic Information-Flow Control

We focus our examples on game structures that result from a reactive system. Let H (resp. L) be the set of atomic propositions forming the high-security (resp. low-security) inputs of a system (we assume $H \cap L = \emptyset$). The game structure then comprises 3-players ξ_N, ξ_H, ξ_L, responsible for resolving non-determinism and selecting high-security and low-security inputs. In particular, the move from ξ_H (resp. ξ_L) determines the values of the propositions in H (resp. L) in the next step. We call a system *input-total*, if in each state, ξ_H and ξ_L can choose all possible valuations for the input propositions.

Strategic Non-Interference. In the introduction, we already saw that *GNI* [31] is (in some cases) a too relaxed notion of security, as it can base the existence of a witness trace on knowledge of the entire input-sequence. Note that *GNI* can be extended to also allow for input from a low-security source that may affect the output. The HyperATL* formula *stratNI* (given in the introduction) instead postulates a *strategy* for ξ_N that incrementally constructs a path that “disproves“ information leakage. We can show that *stratNI* implies *GNI*. Loosely speaking, whenever there is a strategy for the non-determinism based on the finite history of inputs, there also exists a path when given the full history of inputs (as in *GNI*).

▶ **Lemma 4.** *For any system $\mathcal{G}$ that is input-total, we have that if $\mathcal{G} \models$ stratNI then $\mathcal{G} \models$ GNI.*

Simulation-based Non-Interference. Other attempts to non-interference are based on the existence of a bisimulation (or simulation) [40, 39, 30]. While trace-based notions of non-interference (such as *GNI*) only require the existence of a path that witnesses the absence of a leak, simulation based properties require a lock-step relation in which this holds. Given a system $\mathcal{G}$ with initial states s_0. For states s, s' and evaluations $i_L \in 2^L$ and $i_H \in 2^H$, we write $s \Rightarrow^{i_L}_{i_H} s'$ if $L(s') \cap L = i_L$ and $L(s') \cap H = i_H$ and s' is a possible successor of s. A *security simulation* is a relation R on the states of S such that whenever sRt, we have **(1)** s and t agree on the output propositions, and **(2)** for any $i_L \in 2^L$ and $i_H, i'_H \in 2^H$ if $s \Rightarrow^{i_L}_{i_H} s'$ then there exists a t' with $t \Rightarrow^{i_L}_{i'_H} t'$ and $s'Rt'$. Note that this is not equivalent to the fact that $\Rightarrow$ is a simulation in the standard sense [32]. While a standard simulation relation is always reflexive, reflexivity of security simulations guarantees the security of the system [39, 40]. A system is thus called *simulation secure* if there exists a security simulation R with $s_0 R s_0$. It is easy to see that every input-total system that is *simulation secure* already satisfies *GNI*. The converse does, in general, not hold. We can show that HyperATL* can express *simulation security* by using the parallel-composition of quantifiers.

$$[\forall_{\mathcal{G}}\, \pi_1.\ \langle\!\langle \{\xi_N\} \rangle\!\rangle_{\mathcal{G}_{shift}}\, \pi_2.]\ \Box\Big(\bigwedge_{a \in L} a_{\pi_1} \leftrightarrow \bigcirc a_{\pi_2}\Big) \rightarrow \Box\Big(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow \bigcirc a_{\pi_2}\Big)$$

Here we consider HyperATL* with extended quantifier, where we annotate each quantifier with the game structure it is resolved on. $\mathcal{G}_{shift}$ is the structure $\mathcal{G}$ where we added a dummy initial state, that shifts the behaviour of the system by one position, which is again corrected via the next operator in the LTL formula. This allows the strategy for ξ_N in the second copy to base its decision on an already fixed step in the first copy, i.e., it corresponds to a strategy with a fixed lookahead of 1 step. We can show:

▶ **Lemma 5.** *An input-total system $\mathcal{G}$ is* simulation secure *if and only if it satisfies simNI.*

Non-Deducibility of Strategies. Lastly, we consider the notion of *non-deducibility of strategies* (*NDS*) [45]. *NDS* requires not only that each output is compatible with each sequence of inputs, but also with each input-*strategy*. This becomes important as a high-security input player who can observe the internal state of a system might be able to leak information deliberately. As a motivating example, consider the following (first introduced in [45]): Suppose we have a system that reads a binary input `h` from a high-security source and outputs `o`. The system maintains a bit b of information in its state, initially chosen non-deterministically. In each step, the system reads the input `h`, outputs `h` $\oplus\, b$ (where $\oplus$ is the xor-operation), non-deterministically chooses a new value for b and then repeats. As $\oplus$ essentially encodes a one-time pad it is not hard to see, that this system is secure from a purely trace-based point of view (as expressed in e.g. *GNI*): Any possible combination of input and output can be achieved when resolving the non-deterministic choice of b appropriately. If the high-input player is, however, able to observe the system (in the context of [45] the system shares the internal bit on a private channel), she can communicate arbitrary sequence of bits to the low-security environment. Whenever she wants to send bit c, she inputs $c \oplus b$ where b is the internal bit she has access to (note that $(c \oplus b) \oplus b = c$). For such system system we therefore do no want to specify that every possible output sequence is compatible with all possible inputs, but instead compatible with all possible input-*strategies* (based on the state of the system). Phrased differently, there should *not* be a output sequence such that a strategy can reliably *avoid* this output. We can express *NDS* in HyperATL* as follows:

$$\neg\left(\exists \pi_1.\ \langle\!\langle \xi_H \rangle\!\rangle \pi_2.\ \Box\Big(\bigwedge_{a \in L} a_{\pi_1} \leftrightarrow a_{\pi_2}\Big) \rightarrow \Diamond\Big(\bigvee_{a \in O} a_{\pi_1} \not\leftrightarrow a_{\pi_2}\Big)\right)$$

This formula states that there does not exist a trace π_1 such that ξ_H has a strategy to avoid the output of π_1 (provided with the same low-security inputs). *NDS* is a stronger requirement than *GNI*, as shown by the following Lemma:

▶ **Lemma 6.** *For any system $\mathcal{G}$, if $\mathcal{G} \models NDS$ then $\mathcal{G} \models GNI$.*

4.2 Asynchronous Hyperproperties

Reasoning about the strategic behaviour of agents is particularly useful when reasoning about asynchronous hyperproperties, as each asynchronous execution can be considered the result of the decision of an asynchronous scheduler. We call a player an asynchronous scheduler if it can decide whether the system progresses (as decided by the other agents) or stutters. Note that this differs from the asynchronous turn-based games defined in [1]. In our setting, the scheduler does not control which of the player controls the next move, but rather decides if the system as a whole progresses or stutters. In cases where the system does not already include such an asynchronous scheduler (if we e.g. use a Kripke structure interpreted as a 1-player CGS), we can include a scheduler via a simple system transformation:

▶ **Definition 7.** *Given a MSCGS $\mathcal{G} = (Q, q_0, \Xi, \mathcal{M}, \delta, d, \mathbf{AP}, L)$ and a fresh agent* sched *not already included in the set of agents of Ξ. We define the stutter version of $\mathcal{G}$, denoted $\mathcal{G}_{stut}$, by $\mathcal{G}_{stut} := (Q \times \{0,1\}, (q_0, 0), \Xi \uplus \{sched\}, \mathcal{M} \times \{0,1\}, \delta', d', \mathbf{AP} \uplus \{stut\}, L')$ where*

$$\delta'((s,b),\sigma) = \begin{cases} (\delta(s, proj_1 \circ \sigma_{|\Xi}), 0) & \text{if } (proj_2 \circ \sigma)(sched) = 0 \\ (s, 1) & \text{if } (proj_2 \circ \sigma)(sched) = 1 \end{cases}$$

$L'((s,0)) = L(s)$ and $L'(s,1) = L(s) \cup \{stut\}$. Finally $d'(\xi) = d(\xi)$ for $\xi \in \Xi$ and $d'(sched) = m+1$ where m is the maximal element in the codomain of d.

Here $proj_i$ is the projection of the ith element in a tuple and $\circ$ denotes function composition, i.e., $(f \circ g)(x) := f(g(x))$. $\mathcal{G}_{stut}$ thus progresses as $\mathcal{G}$ with the exception of the additional scheduling player. In each step, the $\{0,1\}$-decision of *sched*, which can be based on the decision by the other agents (as *sched* is in the last stage of the game), decides if the system progresses or remains in its current state. The extended state-space $Q \times \{0,1\}$ is used to keep track of the stuttering which becomes visible via the new atomic proposition *stut*. Our construction will be particularly useful when comparing our logic to AHLTL [4].

Observational Determinism. As an example we consider observational-determinism which states that the output along all traces is identical (in HyperLTL $OD := \forall \pi_1. \forall \pi_2.\ \square(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2})$). The example in Fig. 1b does not satisfy this property, as the output is changed at different timepoints. If we consider any system as a multi-agent system including the scheduler *sched*, we can use HyperATL* to naturally express an asynchronous version of OD via:

$$OD_{asynch} := [\langle\langle \{sched\} \rangle\rangle \pi_1. \langle\langle \{sched\} \rangle\rangle \pi_2.]\ fair_{\pi_1} \wedge fair_{\pi_2} \wedge \square(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2})$$

where $fair_{\pi_i} := \square \Diamond \neg stut_{\pi_i}$, asserts that the system may not be stuttered forever. Note that we encapsulated the quantifiers by $[\cdot]$ thus resolving the games in parallel. The schedulers for both copies of the system can thus observe the current state of the other copy. The example from Fig. 1b, after a transformation via Definition 7, satisfies this formula, as the output can be aligned by the scheduling player.

One-Sided Stuttering. By resolving the stuttered traces incrementally (i.e., omitting the $[\cdot]$-brackets) we can also express *one-sided stuttering*, i.e., allow only the second copy to be stuttered. As an example assume P^o is a program written in the high-level programming language and P^a the complied program into binary code. Let S^o and S^a be the state systems of both programs. Using HyperATL* we can now verify that the compiler did not

$\substack{a_{\pi_i} \\ \neg a_{\pi_i}}$	$\mathcal{A}_\varphi = (\{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, \mathbf{0})$ $\rho(q_{init}, [s_1, \cdots, s_n]) = \begin{cases} \substack{\top \\ \bot} & \text{if } a \in L(s_i) \\ \substack{\bot \\ \top} & \text{if } a \notin L(s_i) \end{cases}$
$\varphi_1 \substack{\lor \\ \land} \varphi_2$	$\mathcal{A}_\varphi = (Q_1 \cup Q_2 \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c_1 \uplus c_2 \uplus [q_{init} \mapsto 0])$ $\rho(q, [s_1, \cdots, s_n]) = \begin{cases} \rho_1(q_{0,1}, [s_1, \cdots, s_n]) \substack{\lor \\ \land} \rho_2(q_{0,2}, [s_1, \cdots, s_n]) & \text{if } q = q_{init} \\ \rho_i(q, [s_1, \cdots, s_n]) & \text{if } q \in Q_i \end{cases}$
$\bigcirc \varphi_1$	$\mathcal{A}_\varphi = (Q_1 \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c_1 \uplus [q_{init} \mapsto 0])$ $\rho(q, [s_1, \cdots, s_n]) = \begin{cases} q_{0,1} & \text{if } q = q_{init} \\ \rho_1(q, [s_1, \cdots, s_n]) & \text{if } q \in Q_1 \end{cases}$
$\varphi_1 \substack{\mathcal{U} \\ \mathcal{R}} \varphi_2$	$\mathcal{A}_\varphi = (Q_1 \cup Q_2 \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c_1 \uplus c_2 \uplus [q_{init} \mapsto \substack{1 \\ 0}])$ $\rho(q, [s_1, \cdots, s_n]) = \begin{cases} \rho_2(q_{0,2}, [s_1, \cdots, s_n]) \substack{\lor \\ \land} (\rho_1(q_{0,1}, [s_1, \cdots, s_n]) \substack{\land \\ \lor} q_{init}) & \text{if } q = q_{init} \\ \rho_i(q, [s_1, \cdots, s_n]) & \text{if } q \in Q_i \end{cases}$

Figure 2 APA construction for `LTL` temporal operators. $\mathcal{A}_{\varphi_i} = (Q_i, q_{0,i}, \Sigma_{\varphi_i}, \rho_i, c_i)$ is the inductively constructed automaton for φ_i.

leak information, i.e., the assembly code does provide the same outputs as the original code. As the compiler breaks each program statement into multiple assembly instructions, we can not require the steps to match in a synchronous manner. Instead, the system S^o should be allowed to stutter for the assembly program to catch up. We can express this as follows:

$$\forall_{S^a} \pi_1. \langle\langle \{sched\} \rangle\rangle_{S^o_{stut}} \pi_2. \, fair_{\pi_2} \land \Box(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2})$$

I.e., for every execution of the assembly code we can stutter the program such that the observations align. Here, S^o_{stut} denotes the modified version obtained by introducing an explicit scheduler (Definition 7). Note that we use extend path quantification by annotating a quantifier with the game structure, thereby effectively comparing both systems with respect to a hyperproperty.

5 Automata-Theoretic Model Checking

In this section we present an automata-based algorithm for `HyperATL`* model checking. The crucial insight in our algorithm is how to deal with the strategic quantification. Let us briefly recall `ATL`* model checking [1]: In `ATL`*, checking if $\langle\langle A \rangle\rangle \varphi$ holds in some state s, can be reduced to the non-emptiness check of the intersection of two tree automata. One accepting all possible trees that can be achieved via a strategy for players in A, and one accepting all trees whose paths satisfy the path formula φ [1]. In our hyperlogic this is not possible. When checking $\langle\langle A \rangle\rangle \pi.\varphi$ we can not construct an automaton accepting all trees that satisfy φ, as the satisfaction of φ depends on the paths assigned to the outer path-quantifiers (that are not yet fixed). Instead, we construct an automaton that accepts all *path assignments* for the outer quantifiers such that there exists a winning strategy for the agents in A. We show that alternating automata are well suited to keep track of all path assignments for which a strategy exists, as they allow us to encode the strategic behaviour of $\mathcal{G}$ within the transition function of the automaton.

We assume that the formula φ to be checked is given in *negation normal form*, i.e., negations only occur directly in front of atomic propositions or in front of a path quantifier. By including conjunction ($\wedge$) and release ($\mathcal{R}$) every formula can be translated into a negation normal form of linear size. We, furthermore, assume that if $\langle\!\langle A \rangle\!\rangle \pi.\varphi$ occurs in the formula, we have $A \neq \emptyset$. Note that in this case where $A = \emptyset$ we can use that $\forall \pi.\varphi \equiv \neg \exists \pi.\neg \varphi$. For infinite words $t_1, \cdots, t_n \in \Sigma^\omega$ we define $\mathit{zip}(t_1, \cdots, t_n) \in (\Sigma^n)^\omega$ as the word obtained by combining the traces pointwise, i.e., $\mathit{zip}(t_1, \cdots, t_n)(i) := (t_1(i), \cdots, t_n(i))$. Our algorithm now progresses in a bottom-up manner. Assume that some subformula φ occurs under quantifiers binding path variables $\pi_1, \cdots, \pi_n$. We say that an automaton $\mathcal{A}$ over S^n is $\mathcal{G}$-equivalent to φ, if for any paths $t_1, \cdots, t_n$ it holds that $[\pi_i \mapsto t_i]_{i=1}^n \models_{\mathcal{G}} \varphi$ if any only if $\mathit{zip}(t_1, \cdots, t_n) \in \mathcal{L}(\mathcal{A})$. $\mathcal{G}$-equivalence thus means that an automaton accepts a zipping of traces exactly if the trace assignment constructed from those traces satisfies the formula. By induction on the structure of the formula, we construct an automaton that is $\mathcal{G}$-equivalent to each sub formula.

For the standard boolean combinators and `LTL` temporal operators our construction follows the typical translation from `LTL` to alternating automata [34, 43] given in Fig. 2. The interesting case is now the elimination of a strategy quantifier of the form $\varphi = \langle\!\langle A \rangle\!\rangle \pi.\psi$. Given an inductively constructed APA $\mathcal{A}_\psi$ over $\Sigma_\psi = S^{n+1}$. We aim for an automaton $\mathcal{A}_\varphi$ over $\Sigma_\varphi = S^n$. The automata should accept all traces t over S^n such that there exist a strategy for agents in A such that all traces compatible with this strategy t' when added to t (the trace $t \times t' \in (S^{n+1})^\omega$) is accepted by $\mathcal{A}_\psi$. Let $\mathcal{G} = (S, s_0, \Xi, \mathcal{M}, \delta, d, \mathbf{AP}, L)$ be the given MSCGS. We distinguish between the cases where $A = \Xi$ (i.e., existential quantification) and $A \neq \Xi$.

Existential Quantification. We first consider the case where $A = \Xi$, i.e., $\varphi = \exists \pi.\psi$. Model checking can be done similar to [20]. Let $\mathcal{A}_\psi = (Q, q_0, \Sigma_\psi, \lambda : Q \times \Sigma_\psi \to 2^Q, c)$ be the inductively constructed automaton, translated into a non-deterministic automaton of exponential size via Theorem 1. We then construct $\mathcal{A}_\varphi := (S \times Q \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c')$ where $c'(s, q) = c(q)$ ($c'(q_{init})$ can be chosen arbitrarily) and ρ is defined via

$$\rho(q_{init}, [s_1, \cdots, s_n]) = \{(s', q') \mid q' \in \lambda(q, [s_1, \cdots, s_n, s_n^\circ]) \wedge \exists \sigma : \Xi \to \mathcal{M}.\delta(s_n^\circ, \sigma) = s'\}$$
$$\rho((s, q), [s_1, \cdots, s_n]) = \{(s', q') \mid q' \in \lambda(q, [s_1, \cdots, s_n, s]) \wedge \exists \sigma : \Xi \to \mathcal{M}.\delta(s, \sigma) = s'\}$$

where we define $s_n^\circ = s_n$ if $n \geq 1$ and $s_n^\circ = s_0$ (the initial state) otherwise. Note that $\mathcal{A}_\varphi$ is again a non-deterministic automaton. Every accepting run of $\mathcal{A}_\varphi$ on $\mathit{zip}(t_1, \cdots, t_n)$ now corresponds to a path t in $\mathcal{G}$ such that $\mathcal{A}_\psi$ accepts $\mathit{zip}(t_1, \cdots, t_n, t)$.

(Complex) Strategic Quantification. We now consider the case where $A \neq \Xi$. Our automaton must encode the strategic behaviour of the agents. We achieve this, by encoding the strategic play in the game structure within the transition function of an automaton. Let $\mathcal{A}_\psi^{det} = (Q, q_0, \Sigma_\psi, \lambda : Q \times \Sigma_\psi \to Q, c)$ be a *deterministic* automaton obtained from the inductively constructed $\mathcal{A}_\psi$ via Theorem 1. Note that $\mathcal{A}_\psi^{det}$ is, in the worst case, of double exponential size (in the size of $\mathcal{A}_\psi$). To encode the strategic behaviour in $\mathcal{G}$ we use the alternation available in an automaton by disjunctively choosing moves for controlled players in A, followed by a conjunctive treatment of all adversarial player. The stages of a game, naturally correspond to the order of the move selection. Define the set $A_i := A \cap d^{-1}(i)$ and $\overline{A}_i := (\Xi \setminus A) \cap d^{-1}(i)$ and let m be the maximal element in the codomain of d. The choice of each agent in A followed by those not in A can be encoded into a boolean formula. We define $\mathcal{A}_\varphi := (S \times Q \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c')$ where ρ is defined by

$$\rho(q_{init},[s_1, \cdots, s_n]) = \bigvee_{\sigma_1 : A_1 \to \mathcal{M}} \bigwedge_{\sigma'_1 : \overline{A}_1 \to \mathcal{M}} \cdots \bigvee_{\sigma_m : A_m \to \mathcal{M}} \bigwedge_{\sigma'_m : \overline{A}_m \to \mathcal{M}} \left(\delta(s_n^\circ, \sum_{i=1}^{m}(\sigma_i + \sigma'_i)), \lambda(q_0, [s_1, \cdots, s_n, s_n^\circ])\right)$$

$$\rho((s,q),[s_1, \cdots, s_n]) = \bigvee_{\sigma_1 : A_1 \to \mathcal{M}} \bigwedge_{\sigma'_1 : \overline{A}_1 \to \mathcal{M}} \cdots \bigvee_{\sigma_m : A_m \to \mathcal{M}} \bigwedge_{\sigma'_m : \overline{A}_m \to \mathcal{M}} \left(\delta(s, \sum_{i=1}^{m}(\sigma_i + \sigma'_i)), \lambda(q, [s_1, \cdots, s_n, s])\right)$$

and $c'(s,q) = c(q)$ (we can again define $c'(q_{init})$ arbitrarily). In case $n = 0$, we again define s_n° as the initial state s_0, otherwise $s_n^\circ = s_n$. Note that in the case where the MSCGS is a CGS, i.e., $d = \mathbf{0}$ the transition function has the form $\vee\wedge$, where the choices in A are considered disjunctively and the choices by all other agents conjunctively. Our construction can be extended to handle the self composition $[\langle\!\langle A_1 \rangle\!\rangle \pi_1. \cdots \langle\!\langle A_k \rangle\!\rangle \pi_k]$ (see [6] for details).

Negated Quantification. We extend our construction to handle negation outside of quantifiers, i.e., a formula $\varphi = \neg\langle\!\langle A \rangle\!\rangle \pi.\psi$ via $\mathcal{A}_\varphi := \overline{\mathcal{A}_{\langle\!\langle A \rangle\!\rangle \pi.\psi}}$ by using Theorem 2.

▶ **Proposition 8.** *$\mathcal{A}_\varphi$ is $\mathcal{G}$-equivalent to φ.*

By following our inductive construction, we obtain an automaton over the singleton alphabet (empty state sequences) that is non-empty iff the model satisfies the formula. Emptiness of an alternating parity automaton can then be checked in polynomial size (assuming a fixed number of colours) [28].

We can observe a gap in complexity of algorithm between simple and complex quantification. The former case requires a translation of an alternating automaton to a non-deterministic one whereas the latter requires a full determinisation. To capture the complexity of our algorithm we define $\mathcal{T}_c(k, n)$ as a tower of k exponents in n, i.e., $\mathcal{T}_c(0, n) = n^c$ and $\mathcal{T}_c(k + 1, n) = c^{\mathcal{T}_c(k,n)}$. For $k \geq 0$ we define k-EXPSPACE as the class of languages recognised by a deterministic Turing machine ([41]) with space $\mathcal{T}_c(k, \mathcal{O}(n))$ for some c (and similarly for time). We define (-1)-EXPSPACE as NLOGSPACE. Note that 0-EXPSPACE = PSPACE.

▶ **Theorem 9.** *Model checking of a HyperATL* formula with k complex and l simple quantifiers is in $(2k+l)$-EXPTIME. If $l \geq 1$ and the formula is linear, it is also in $(2k+l-1)$-EXPSPACE (both in size of the formula).*

The fact that we can derive a better upper bound when $l > 0$ follows from the fact that we can determine the emptiness of a non-deterministic automaton in NLOGSPACE [44] and for an alternating automaton only in polynomial time (for a fixed number of colours) [28]. Note that for the syntactic fragment of HyperCTL* our algorithm matches the algorithm in [20].

6 Lower Bounds for Model Checking

Theorem 9 gives us an upper bound on the model checking problem for HyperATL*. We can show the following lower bound

▶ **Theorem 10.** *Model checking of a linear HyperATL* formula with k complex and l simple quantifiers is $(2k + l - 1)$-EXPSPACE-hard in the size of the formula, provided $l \geq 1$.*

The proof of Theorem 10 proceeds by encoding space-bounded Turing machines into HyperATL*. We show that (complex) strategic quantification can be used to encode an incremental counter that grows by *two exponents* with each quantifier, opposed to the increment by a single exponent for simple quantification [38]. This is possible, as we can design a formula that requires the first player to enumerate a counter, while the second play can challenge its correctness. The only winning strategy for the former player is then to output a correct counter that holds up against all scrutiny by the latter player. As the construction of the counter is rather complex, we refer the interested reader to a detailed proof in the full version [6].

Note that Theorem 10 is conditioned on $l \geq 1$. This gives an interesting complexity landscape: In cases where $l \geq 1$, model checking is $(2k+l-1)$-EXPSPACE-complete (irrespective of k). If $l = 0$ we get an upper bound of $2k$-EXPTIME (Theorem 9). In the special case where $l = 0$ and $k = 1$ we get a matching lower bound from the ATL* model-checking problem [1] (subsuming LTL realizability [36, 37]) and thus 2-EXPTIME-completeness. If $k > 1$ the best lower bound is $(2k-2)$-EXPSPACE. The exact complexity for the case where $k > 1$ and $l = 0$ is thus still open.

7 HyperATL* vs. asynchronous HyperLTL

We have seen that our strategic logic can naturally express asynchronous hyperproperties. In this section, we compare our logic to AHLTL [4], a recent extension of HyperLTL specifically designed to express such asynchronous properties. AHLTL is centred around the idea of a trajectory, which, informally speaking, is the stuttering of traces in a system. In AHLTL an initial trace quantifier prefix is followed by a quantification over such a trajectory. For example, a formula of the form $\forall \pi_1. \cdots \forall \pi_n.\mathbf{E}.\varphi$ means that for all paths $\pi_1, \cdots, \pi_n$ in the system there exists some stuttering of the paths, such that φ is satisfied. AHLTL follows a purely trace-based approach where the stuttering is fixed, knowing the full paths $\pi_1, \cdots, \pi_n$. In comparison, in our logic a *strategy* must decide if to stutter based on finite a prefix in the system. Model checking AHLTL is, in general, undecidable [4]. The largest known fragment for which an algorithm is known are formulas of the form $\forall \pi_1. \cdots \forall \pi_n.\mathbf{E}.\varphi$ where φ is an *admissible formula* [4] which is a conjunction of formulas of the form $\Box \bigwedge_{a \in P} a_{\pi_i} \leftrightarrow a_{\pi_j}$ (where P is a set of atomic propositions) and stutter-invariant formulas over a single path variable. We can show the following (where $\mathcal{G}_{stut}$ is the stutter transformation from Definition 7):

▶ **Theorem 11.** *For any Kripke structure $\mathcal{G}$ and AHLTL formula of the form $\forall \pi_1. \cdots \forall \pi_n.\mathbf{E}.\varphi$ it holds that if $\mathcal{G}_{stut} \models [\langle\langle\{sched\}\rangle\rangle \pi_1. \cdots \langle\langle\{sched\}\rangle\rangle \pi_n]\ \varphi \wedge \bigwedge_{i \in \{1,\cdots,n\}} fair_{\pi_i}$ (1) then $\mathcal{G} \models_{AHLTL} \forall \pi_1. \cdots \forall \pi_n.\mathbf{E}.\varphi$ (2). If φ is an admissible formula, (1) and (2) are equivalent.*

Theorem 11 gives us a sound approximation of the (undecidable) AHLTL model checking. Furthermore, for admissible formulas, AHLTL can be truthfully expressed in our logic. As shown in [4], many interesting properties can be expressed using an admissible formula and can thus be (truthfully) checked in our logic. Our framework can therefore express many interesting properties, is fully decidable, and also subsumes the largest known decidable fragment of AHLTL.

8 Experimental Evaluation

While MC for the full logic is very expensive (Theorem 10) and likely not viable in practice, formulas of the form $[\langle\langle A_1\rangle\rangle \pi_1. \ \cdots \langle\langle A_n\rangle\rangle \pi_n.]\varphi$ where φ is quantifier free, can be checked efficiently via a reduction to a parity game (see the full version [6] for details). Note that

Table 1 Validity of various `HyperATL`* formulas on small benchmark programs. A ✓(resp. ✗) means that the formula is satisfied (resp. not satisfied). The time consumption is given in milliseconds.

	(OD)	**(NI)**	**(simSec)**	**(sGNI)**
`P1`	✓(15)	✓(16)	✓(16)	✓(46)
`P2`	✗(112)	✓(80)	✓(83)	✓(432)
`P3`	✗(70)	✗(44)	✓(54)	✓(112)
`P4`	✗(73)	✗(64)	✗(70)	✓(191)

(a) Examples for Information-Flow Policies.

	(OD)	**(OD$_{asynch}$)**	**(NI$_{asynch}$)**
`Q1`$_i$	✗(112)	✓(788)	✓(812)
`Q1`$_{ii}$	✗(281)	✓(3372)	✓(3516)
`Q1`$_{iii}$	✗(1680)	✓(20756)	✓(24078)
`Q2`	✗(985)	✗(18141)	✓(6333)

(b) Examples for Asynchronous Hyperproperties.

all alternation-free `HyperLTL` formulas, the reduction from the MC approach from [12] and the reduction in Theorem 11 fall in this fragment. We implemented a prototype model checker for this fragment to demonstrate different security notions (both synchronous and asynchronous) on small example programs. Our tool uses `rabinizer 4` [27] to convert a `LTL` formula into a deterministic automaton and `pgsolver` [22] to solve parity games. Our tool is publicly available (see the full version [6]).

Information-Flow Policies. We have created a small benchmark of simple programs that distinguish different information-flow policies. We checked the following properties: **(OD)** is the standard (alternation-free) formula of observational determinism, **(NI)** is a simple formulation of non-interference due to [23], **(simSec)** is simulation security [39] as expressed in Sec. 4.1. Finally, **(sGNI)** is the simple game based definition of GNI resolved on the parallel-composition (as used in [12]). We designed small example programs that demonstrate the difference between security guarantees and present the results in Table 1a. Note that the model checking algorithm for $\forall^*\exists^*$ formulas from [12] is subsumed by our approach. As we reduce the search for a strategy for the existential player to a parity game opposed to a SMT constraint, we can handle much bigger systems.

Asynchronous Hyperproperties. To showcase the expressiveness of our framework to handle asynchronous properties, we implemented the stuttering transformation from Definition 7. We evaluated our tool by checking example programs both on synchronous observational-determinism **(OD)** and asynchronous versions of OD **(OD$_{asynch}$)** and non-interference **(NI)**$_{asynch}$. Note that while **(OD$_{asynch}$)** can also express in the decidable fragment of `AHLTL`, **(NI$_{asynch}$)** is not an admissible formula (and can not be handled in [4]). As non-interference only requires the outputs to align provided the inputs do, one needs to take care that the asynchronous scheduler does not “cheat“ by deliberately missaligning inputs and thereby invalidating the premise of this implication. Our results are given in Table 1b. To demonstrate the state-explosion problem we tested the same program (`Q1`) with different bit-widths (programs `Q1`$_i$, `Q1`$_{ii}$, `Q1`$_{iii}$), causing an artificial blow-up in the number of states.

9 Related Work

There has been a lot of recent interest in logics for hyperproperties. Most logics are obtained by extending standard temporal or first-order/second-order logics with either path quantification or by a special equal-level predicate [21]. See [11] for an overview. To the best of our knowledge, none of these logics can express strategic hyperproperties.

Alternating-time Temporal Epistemic Logic. The relationship between epistemic logics and hyperlogic is interesting, as both reason about the flow of information in a system. As shown in [7], HyperLTL and $\mathtt{LTL}_{\mathcal{K}}$ (LTL extended with a knowledge operator [15]) have incomparable expressiveness. In HyperQPTL, which extends HyperLTL with propositional quantification [18], the knowledge operator can be encoded by explicitly marking the knowledge positions via propositional quantification [38, §7]. Alternating-time temporal logic has also been extended with knowledge operators [42]. The resulting logic, ATEL, can express properties of the form "if ξ knows ϕ, then she can enforce ψ via a strategy." The natural extension of the logic in [42], which allows for arbitrary nesting of quantification and operators (i.e., an extension of $\mathtt{ATL}^*$ instead of ATL) is incomparable to $\mathtt{HyperATL}^*$.

Model Checking. Decidable model checking is a crucial prerequisite for the effective use of a logic. Many of the existing (synchronous) hyperlogics admit decidable model checking, although mostly with non-elementary complexity (see [17] for an overview). For alternating-time temporal logic (in the non-hyper realm), model checking is efficient (especially when one prohibits arbitrary nesting of temporal operators and quantifiers as in ATL) [1, 2]. If one allows operators and quantifiers to be nested arbitrarily ($\mathtt{ATL}^*$), model checking subsumes LTL satisfiability and realizability. This causes a jump in the model checking complexity to 2-EXPTIME-completeness. As our lower bound demonstrates, the combination of strategic quantification and hyperproperties results in a logic that is algorithmically harder (for model checking) than non-strategic hyperproperties (as HyperLTL) or non-hyper strategic logics (as $\mathtt{ATL}^*$). The fragment of $\mathtt{HyperATL}^*$ implemented in our prototype model checker subsumes alteration-free HyperLTL (see MCHyper [20]), model checking via explicit strategies [12] and the (known) decidable fragment of AHLTL [4].

Asynchronous Hyperproperties. Extending hyperlogics to express asynchronous properties has only recently started to gain momentum [25, 4, 8]. In [4] they extend HyperLTL with explicit trajectory quantification. [25] introduced a variant of the polyadic μ-calculus, H_μ, able to express hyperproperties. In [8] they extended HyperLTL with new modalities that remove redundant (for example stuttering) parts of a trace. Model checking is undecidable for all three logics. The (known) decidable fragment of [4] can be encoded into $\mathtt{HyperATL}^*$. The only known decidable classes for H_μ [25] and $\mathtt{HyperLTL}_S$ [8] are obtained by bounding the asynchronous offset by a constant k, i.e., asynchronous execution may not run apart ("diverge") for more than k steps. For actual software, this is a major restriction.

10 Conclusion

We have introduced $\mathtt{HyperATL}^*$, a temporal logic for strategic hyperproperties. Besides the obvious benefits of simultaneously reasoning about strategic choice and information flow, $\mathtt{HyperATL}^*$provides a natural formalism to express *asynchronous* hyperproperties, which has been a major challenge for previous hyperlogics. Despite the added expressiveness, $\mathtt{HyperATL}^*$ model checking remains decidable, with comparable cost to logics for synchronous hyperproperties (cf. Theorem 9). $\mathtt{HyperATL}^*$ is the first logic for asynchronous hyperproperties where model checking is decidable for the entire logic. Its expressiveness and decidability, as well as the availability of practical model checking algorithms, make it a very promising choice for model checking tools for hyperproperties.

References

1 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

2 Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. MOCHA: modularity in model checking. In *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525. Springer, 1998. doi:10.1007/BFb0028774.

3 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

4 Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A temporal logic for asynchronous hyperproperties. In *Computer Aided Verification - 33nd International Conference, CAV 2021, Los Angeles, CA, USA, July 18-24, 2021*, Lecture Notes in Computer Science. Springer, 2021.

5 Raphaël Berthon, Bastien Maubert, and Aniello Murano. Decidability results for atl* with imperfect information and perfect recall. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 1250–1258. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3091299.

6 Raven Beutner and Bernd Finkbeiner. A temporal logic for strategic hyperproperties. *CoRR*, abs/2107.02509, 2021. arXiv:2107.02509.

7 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11.

8 Laura Bozzelli, Adriano Peron, and César Sánchez. Asynchronous extensions of hyperltl. In *36nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021.* ACM, 2021.

9 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

10 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

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

12 Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2019. doi:10.1007/978-3-030-25540-4_7.

13 Doron Drusinsky and David Harel. On the power of bounded concurrency I: finite automata. *J. ACM*, 41(3):517–539, 1994. doi:10.1145/176584.176587.

14 E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.

15 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge.* MIT Press, 1995. doi:10.7551/mitpress/5803.001.0001.

16 Bernd Finkbeiner. Temporal hyperproperties. *Bull. EATCS*, 123, 2017. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/514.

17 Bernd Finkbeiner. Model checking algorithms for hyperproperties (invited paper). In *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings*, volume 12597 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2021. doi:10.1007/978-3-030-67067-2_1.

18 Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing omega-regular hyperproperties. In *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 40–63. Springer, 2020. doi:10.1007/978-3-030-53291-8_4.

19 Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesizing reactive systems from hyperproperties. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2018. doi:10.1007/978-3-319-96145-3_16.

20 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking hyperltl and hyperctl*. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2015. doi:10.1007/978-3-319-21690-4_3.

21 Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.30.

22 Oliver Friedmann and Martin Lange. Solving parity games in practice. In *Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings*, volume 5799 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2009. doi:10.1007/978-3-642-04761-9_15.

23 Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.

24 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Propositional dynamic logic for hyperproperties. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 50:1–50:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.50.

25 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021. doi:10.1145/3434319.

26 Marieke Huisman, Pratik Worah, and Kim Sunesen. A temporal logic characterisation of observational determinism. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*, page 3. IEEE Computer Society, 2006. doi:10.1109/CSFW.2006.6.

27 Jan Kretínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 567–577. Springer, 2018. doi:10.1007/978-3-319-96145-3_30.

28 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 224–233. ACM, 1998. doi:10.1145/276698.276748.

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

30 Heiko Mantel and Henning Sudbrock. Flexible scheduler-independent security. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security,*

Athens, Greece, September 20-22, 2010. Proceedings, volume 6345 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2010. doi:10.1007/978-3-642-15497-3_8.

31 Daryl McCullough. Noninterference and the composability of security properties. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 18-21, 1988*, pages 177–186. IEEE Computer Society, 1988. doi:10.1109/SECPRI.1988.8110.

32 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. doi:10.1007/3-540-10235-3.

33 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984. doi:10.1016/0304-3975(84)90049-5.

34 David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS '88), Edinburgh, Scotland, UK, July 5-8, 1988*, pages 422–427. IEEE Computer Society, 1988. doi:10.1109/LICS.1988.5139.

35 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

36 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.

37 Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989. doi:10.1007/BFb0035790.

38 Markus N. Rabe. *A temporal logic approach to information-flow control.* PhD thesis, Saarland University, 2016. URL: http://scidok.sulb.uni-saarland.de/volltexte/2016/6387/.

39 Andrei Sabelfeld. Confidentiality for multithreaded programs via bisimulation. In *Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference, PSI 2003, Akademgorodok, Novosibirsk, Russia, July 9-12, 2003, Revised Papers*, volume 2890 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2003. doi:10.1007/978-3-540-39866-0_27.

40 Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00, Cambridge, England, UK, July 3-5, 2000*, pages 200–214. IEEE Computer Society, 2000. doi:10.1109/CSFW.2000.856937.

41 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

42 Wiebe van der Hoek and Michael J. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Stud Logica*, 75(1):125–157, 2003. doi:10.1023/A:1026185103185.

43 Moshe Y. Vardi. Alternating automata and program verification. In *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1995. doi:10.1007/BFb0015261.

44 Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi:10.1006/inco.1994.1092.

45 J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 7-9, 1990*, pages 144–161. IEEE Computer Society, 1990. doi:10.1109/RISP.1990.63846.