Hyper Strategy Logic

Raven Beutner
CISPA Helmholtz Center for
Information Security
Germany

Bernd Finkbeiner
CISPA Helmholtz Center for
Information Security
Germany

ABSTRACT

Strategy logic (SL) is a powerful temporal logic that enables strategic reasoning in multi-agent systems. SL supports explicit (first-order) quantification over strategies and provides a logical framework to express many important properties such as Nash equilibria, dominant strategies, etc. While in SL the same strategy can be used in multiple strategy profiles, each such profile is evaluated w.r.t. a path-property, i.e., a property that considers the *single* path resulting from a particular strategic interaction. In this paper, we present Hyper Strategy Logic (HyperSL), a strategy logic where the outcome of multiple strategy profiles can be compared w.r.t. a *hyperproperty*, i.e., a property that relates *multiple* paths. We show that HyperSL can capture important properties that cannot be expressed in SL, including non-interference, quantitative Nash equilibria, optimal adversarial planning, and reasoning under imperfect information. On the algorithmic side, we identify an expressive fragment of HyperSL with decidable model checking and present a model-checking algorithm. We contribute a prototype implementation of our algorithm and report on encouraging experimental results.

KEYWORDS

Strategy Logic, Hyperproperties, Model Checking, Imperfect Information, Nash Equilibrium, Information-Flow Cotrol

ACM Reference Format:
Raven Beutner and Bernd Finkbeiner. 2024. Hyper Strategy Logic. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

Two important developments in the area of reactive systems concern the study of strategic properties in multi-agent systems (MAS) and the study of hyperproperties. *Strategic properties* analyze the ability of agents to achieve a goal against (or in cooperation) with other agents. Logics such as alternating-time temporal logic (ATL*) [2] and strategy logic (SL) [25, 45] reason about the temporal interaction of such agents and allow for rigorous correctness guarantees using techniques such as model-checking. *Hyperproperties* [28] are properties that relate *multiple* executions within a system. Hyperproperties occur in many situations in computer science where traditional path properties (that refer to *individual* system execution) are not sufficient. Typical examples include **(1)** *optimality*, e.g., one path reaching a goal faster than all other paths; **(2)** *information-flow policies*, e.g., requiring that any two paths with identical low-security input should produce the same low-security output [42]; and **(3)** *robustness*, i.e., stating that similar inputs should lead to similar outputs [26].

Such hyperproperties are also of vital importance in MASs. For example, we might ask if some agent has a strategy to achieve a goal without leaking information (an information-flow property) or can achieve a goal faster than some other agent (an optimality requirement). Yet existing logics for strategic reasoning (such as variants of SL [25, 45]) cannot express such hyper-requirements (we discuss related approaches in Section 2). We illustrate this on the example of Nash equilibria:

Assume we are given a MAS with agents $\{1, \dots, n\}$ and LTL properties $\psi_1, \dots, \psi_n$ that describe the objectives of the agents. Agent i wants to make sure that $\mathsf{F}\,\psi_i$ holds, i.e., formula ψ_i *eventually* holds. We want to check whether the system admits a Nash equilibrium, i.e., there exists a strategy for each agent such that no agent has an incentive to deviate in order to fulfill her objective [48]. In SL, we can express the existence of a Nash equilibrium as follows:

$$\exists x_1, \dots, x_n. \forall y. \bigwedge_{i=1}^{n} \Big((\mathsf{F}\,\psi_i)(\vec{x}[i \mapsto y]) \to (\mathsf{F}\,\psi_i)(\vec{x}) \Big)$$

where we abbreviate the strategy profiles $\vec{x} = (x_1, \dots, x_n)$ and $\vec{x}[i \mapsto y] = (x_1, \dots, x_{i-1}, y, x_{i+1}, \dots, x_n)$. In the variant SL we consider here (similar to the SL by Chatterjee et al. [25]), atomic formulas have the form $\psi(\vec{x})$ where ψ is an LTL formula, and $\vec{x}$ is a strategy profile that assigns a strategy to each agent. Formula $\psi(\vec{x})$ holds if the unique path that results from the interaction of the strategies in $\vec{x}$ satisfies ψ. The above formula thus states that if some agent i can achieve $\mathsf{F}\,\psi_i$ by playing some deviating strategy y instead of x_i, i.e., the unique play that results from strategy profile $\vec{x}[i \mapsto y]$ satisfies $\mathsf{F}\,\psi_i$, then i could stick with strategy x_i, i.e., $\mathsf{F}\,\psi_i$ also holds under strategy profile $\vec{x}$.

In the formula, we effectively compare two plays under strategy profiles $\vec{x}$ and $\vec{x}[i \mapsto y]$. However, SL limits the comparison between multiple interactions to a boolean combination of LTL properties on their outcomes (paths). In game-theoretic terms, the above formula assumes that the reward for each agent is binary; the reward of agent i is maximal if $\mathsf{F}\,\psi_i$ holds and minimal if it does not. This fails to capture quantitative reward, for example, in a setting where agent i receives a higher reward (and thus deviates) by fulfilling ψ_i *sooner*. To express the existence of such a quantitative equilibrium, a boolean formula over individual temporal properties on strategy profiles $\vec{x}$ and $\vec{x}[i \mapsto y]$ is not sufficient. We need a more powerful mechanism that can compare the temporal behavior of *multiple* paths: a *hyperproperty*.

HyperSL. In this paper, we propose HyperSL – a new temporal logic that combines first-order strategic reasoning (as in SL) with

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand.

the ability to compare *multiple* paths w.r.t. a hyperproperty. Syntactically, we use path variables to refer to multiple paths at the same time (similar to existing hyperlogics such as HyperCTL* [27] and HyperATL* [14, 17]). In HyperSL, atomic formulas have the form $\psi[\pi_1 : \vec{x}_1, \dots, \pi_m : \vec{x}_m]$ where $\pi_1, \dots, \pi_m$ are path variables, $\vec{x}_1, \dots, \vec{x}_m$ are strategy profiles (assigning a strategy to each agent), and ψ is an LTL formula where atomic propositions are indexed by path variables from $\pi_1, \dots, \pi_m$. The formula states that the plays resulting from strategy profiles $\vec{x}_1, \dots, \vec{x}_m$, when bound to $\pi_1, \dots, \pi_m$, (together) satisfy the hyperproperty expressed by ψ.

Coming back to the Nash equilibrium example from before, we can use HyperSL to express the existence of a Nash equilibrium in a quantitative reward setting as follows:

$$\exists x_1, \dots, x_n. \forall y. \bigwedge_{i=1}^{n} \Big((\neg \psi_{i\,\pi_1} \mathbin{\mathsf{W}} \psi_{i\,\pi_2}) \begin{bmatrix} \pi_1 : \vec{x}[i \mapsto y] \\ \pi_2 : \vec{x} \end{bmatrix} \Big)$$

Here, we write $\psi_{i\,\pi_1}$ (resp. $\psi_{i\,\pi_2}$) to state that ψ_i holds on path π_1 (resp. π_2). In the formula, we again quantify over a deviating strategy y, but can compare the two paths resulting from strategy profiles $\vec{x}[i \mapsto y]$ and $\vec{x}$ within the *same* temporal formula. This formula states that path π_1 (constructed using strategy profile $\vec{x}[i \mapsto y]$) does not satisfy ψ_i strictly before ψ_i holds on path π_2 (constructed using strategy profile $\vec{x}$).[1] If the above formula holds, $\vec{x}$ thus constitutes a strategy profile such that no agent could achieve its goal strictly sooner (if at all).

Note that we can express any Nash equilibrium as long as "agent i (strictly) prefers the outcome on path π_1 over that on path π_2" is expressible using an LTL formula over π_1, π_2. Likewise, HyperSL can, e.g., express that some strategy **(1)** reaches a goal without leaking information, **(2)** is at least as fast as any other strategy, or **(3)** is robust w.r.t. the behavior of other agents.

Expressiveness of HyperSL. After we introduce HyperSL (in Section 4), we study its relation to existing logics (in Section 5). We show that HyperSL subsumes many non-hyper strategy logics as well as hyperlogics such as HyperCTL* [27], HyperATL* [14, 17], and HyperATL$^*_S$ [19] (see Section 2). Moreover, HyperSL also admits reasoning under imperfect information despite having a semantics defined under complete information. The key observation here is that "acting under imperfect information" *is a hyperproperty*: a strategy acts under imperfect information if, on any *pair* of paths with the same observation, the strategy chooses the same action. Formally, we show that HyperSL subsumes SL_{ii} [12, 13], a strategy logic centered around imperfect information.

Model Checking. HyperSL's ability to compare multiple strategic interactions renders model-checking (MC) undecidable. In Section 6, we identify a fragment of our logic – called HyperSL[SPE] – for which MC is possible. Intuitively, in HyperSL[SPE], the quantifier prefix should be such that we can group it into individual "blocks" where the strategy variables from each block are used on independent path variables. HyperSL[SPE] subsumes SL[1G] (the single-goal fragment of SL) [46], HyperLTL [27], HyperATL* [14, 17], and HyperATL$^*_S$ [19], but also captures properties that cannot be expressed in existing logics. We argue that HyperSL[SPE] is the largest fragment with a decidable model-checking problem that is defined purely in terms of the quantification structure.

Implementation and Experiments. We implement our MC algorithm for HyperSL[SPE] in the `HyMASMC` tool [19] and experiment with various MAS models (in Section 7). Our experiments show that `HyMASMC` performs well on many *non*-hyper strategy logic specifications and can verify complex hyperproperties that cannot be expressed in any existing logic.

2 RELATED WORK

SL has been extended along multiple dimensions, including agent-unbinding [37], reasoning about probabilities [4], epistemic properties [7, 10, 41], and quantitative properties [21]. We refer to [45, 49] for a more in-depth discussion. The common thread in all the previous extensions is a focus on the temporal behavior on *individual* paths. HyperSL generalizes SL and is the first to compare *multiple* paths. Even quantitative extensions like SL[$\mathcal{F}$] [21] evaluate an LTL[$\mathcal{F}$]-formula on a *per-path* basis. In contrast, HyperSL can express complex relationships *between* paths.

Studying logics that can express strategic properties under *imperfect information* has attracted much attention and led to various extensions of ATL* [10, 11, 23, 30, 38] and SL [12, 36]. Berthon et al. [12] showed that their logic, SL_{ii}, subsumes most existing approaches. We show that HyperSL can also reason about imperfect information (and subsumes SL_{ii}) despite having a semantics that is defined under full information.

Logics for expressing hyperproperties in non-agent-based systems (e.g., labeled transition systems) have been obtained by extending existing temporal or first-order logics with explicit path quantification over path/trace variables or an equal-level predicate [15, 27, 29, 34, 35]. As strategic reasoning is significantly more powerful than pure path quantification, HyperSL subsumes HyperCTL* (when interpreting transition systems as single-agent MASs). HyperATL* [14, 17] and HyperATL$^*_S$ [19] extend alternating-time temporal logic (ATL*) [2] with path variables and strategy-sharing constraints, leading to a strategic hyperlogic that can express important security properties such as non-deducibility of strategies [54] and simulation security [51]. Similar to ATL*, the strategic reasoning in HyperATL* and HyperATL$^*_S$ is limited to *implicit* reasoning about the strategic ability of coalitions of agents and cannot explicitly reason about strategies as, e.g., needed to express the existence of a Nash equilibrium.

Our model-checking algorithm for HyperSL[SPE] is based on an iterative elimination of path (variables) in an automaton, similar to existing algorithms for HyperCTL* [33] and HyperATL* [17, 19]. Compared to HyperATL*, we need to eliminate paths by simulating an *arbitrary* prefix of strategy quantifiers, leading to a more involved construction and more complex correctness proof.

3 PRELIMINARIES

We let AP be a fixed finite set of atomic propositions and fix a fixed finite set of agents $Agts = \{1, \dots, n\}$. Given a set X, we write X^+ (resp. X^ω) for the set of non-empty finite (resp. infinite) sequences over X. For $u \in X^\omega$ and $j \in \mathbb{N}$, we write $x(j)$ for the ith element, $u[0, j]$ for the finite prefix up to position j (of length $j+1$), and $u[j, \infty]$ for the infinite suffix starting at position j.

[1] We make use of LTL's weak until operator W. Formula $\psi_1 \mathbin{\mathsf{W}} \psi_2$ holds if ψ_1 holds until ψ_2 holds eventually *or* ψ_1 holds at all times.

Concurrent Game Structures. As the underlying model of MASs, we use concurrent game structures (CGS) [2]. A CGS is a tuple $\mathcal{G} = (S, s_0, \mathbb{A}, \kappa, L)$ where S is a finite set of states, $s_0 \in S$ is an initial state, $\mathbb{A}$ is a finite set of actions, $\kappa : S \times (Agts \to \mathbb{A}) \to S$ is a transition function, and $L : S \to 2^{AP}$ is a labeling function. The transition function takes a state s and an *action profile* $\vec{a} : Agts \to \mathbb{A}$ (mapping each agent an action) and returns a unique successor state $\kappa(s, \vec{a})$. We write $\prod_{i \in Agts} a_i$ for the action profile where each agent i is assigned action a_i.

A strategy in $\mathcal{G}$ is a function $f : S^+ \to \mathbb{A}$, mapping finite plays to actions. We denote the set of all strategies in $\mathcal{G}$ with $Str(\mathcal{G})$. A *strategy profile* $\prod_{i \in Agts} f_i$ assigns each agent i a strategy $f_i \in Str(\mathcal{G})$. Given strategy profile $\prod_{i \in Agts} f_i$ and state $s \in S$, we can define the unique path resulting from the interaction between the agents: We define $Play_{\mathcal{G}}(s, \prod_{i \in Agts} f_i)$ as the unique path $p \in S^\omega$ such that $p(0) = s$ and for every $j \in \mathbb{N}$ we have $p(j+1) = \kappa\big(p(j), \prod_{i \in Agts} f_i(p[0,j])\big)$. That is, in every step, we construct the action profile $\prod_{i \in Agts} f_i(p[0,j])$ in which each agent i plays the action determined by f_i on the current prefix $p[0,j]$.

Alternating Automata. Our model-checking algorithm is based on alternating automata over infinite words. These automata generalize nondeterministic automata by alternating between nondeterministic and universal transitions [53]. For transitions of the former kind, we can choose *some* successor state; for transitions of the latter type, we need to consider *all* possible successor states. Formally, an *alternating parity automaton (APA)* over alphabet Σ is a tuple $\mathcal{A} = (Q, q_0, \delta, c)$ where Q is a finite set of states, $q_0 \in Q$ is an initial state, $c : Q \to \mathbb{N}$ is a color assignment, and $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is a transition function that maps each state-letter pair to a positive boolean formula over Q (denoted with $\mathbb{B}^+(Q)$). For example, if $\delta(q, l) = q_1 \lor (q_2 \land q_3)$, we can – from state $q \in Q$ and upon reading letter $l \in \Sigma$ – either move to state q_1 or move to *both* q_2 and q_3 (i.e., we spawn two copies of our automaton, one starting in state q_2 and one in q_3). We write $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ for the set of all infinite words for which we can construct a run tree that respects the transition formulas such that the *minimal* color that occurs infinitely many times (as given by c) is *even*. For space reasons, we cannot give a formal semantics of APA runs and instead refer the reader to the full version [18]. No specific knowledge about APAs is required to understand the high-level idea of our algorithm.

A special kind of APAs are *deterministic parity automata (DPA)* in which δ is a function $Q \times \Sigma \to Q$ assigning a unique successor state to each state-letter pair. We can always determinize APAs:

Proposition 1 ([44, 50]). *For any APA $\mathcal{A}$ with n states, we can effectively compute a DPA $\mathcal{A}'$ with at most $2^{2^{O(n)}}$ states such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

4 HYPER STRATEGY LOGIC

Our new logic HyperSL is centered around the idea of combining strategic reasoning (as possible in strategy logic [25, 45]) with the ability to express hyperproperties (as possible in logics such as HyperCTL* [27]). To accomplish this, we combine the ideas from both disciples. On the strategy-logic-side, we use strategy variables to quantify over strategies. On the hyper-side, we use path variables to compare multiple paths within a temporal formula.

Let $\mathcal{X}$ be a set of *strategy variables* and $\mathcal{V}$ a set of *path variables.* We typically use lowercase letters ($x, y, z, x_1, \ldots$) for strategy variables and variations of π ($\pi, \pi', \pi_1, \ldots$) for path variables. Path and state formulas in HyperSL are generated by the following grammar:

$$\psi := a_\pi \mid \varphi_\pi \mid \psi \land \psi \mid \neg\psi \mid \mathsf{X}\,\psi \mid \psi \,\mathsf{U}\, \psi$$
$$\varphi := \forall x.\varphi \mid \exists x.\varphi \mid \psi\big[\pi_1 : \vec{x}_1, \ldots, \pi_m : \vec{x}_m\big]$$

where $a \in AP$ is an atomic proposition, $\pi, \pi_1, \ldots, \pi_m \in \mathcal{V}$ are path variables, $x \in \mathcal{X}$ is a strategy variable, and $\vec{x}_1, \ldots, \vec{x}_m : Agts \to \mathcal{X}$ are *strategy profiles* that assign a strategy variable to each agent. We often write $\psi[\pi_k : \vec{x}_k]_{k=1}^m$ as a shorthand for $\psi[\pi_1 : \vec{x}_1, \ldots, \pi_m : \vec{x}_m]$. We use $\mathbb{Q}$ as a placeholder for either $\forall$ or $\exists$. We use the standard Boolean connectives $\lor, \to, \leftrightarrow$, and constants $\top, \bot$, as well as the derived LTL operators *eventually* $\mathsf{F}\,\psi := \top \,\mathsf{U}\, \psi$ and *globally* $\mathsf{G}\,\psi := \neg\,\mathsf{F}\,\neg\psi$. For each formula $\psi[\pi_k : \vec{x}_k]_{k=1}^m$, we assume that all path variables that are free in ψ belong to $\{\pi_1, \ldots, \pi_m\}$, i.e., all used path variables are bound to some strategy profile. We further assume that all nested state formulas are closed.

Note that our syntax does not support boolean combinations of state formulas as is usual in SL [45]. As we can evaluate a path formula on multiple paths, we can move boolean combinations within the path formulas.

Example 1. *Consider the SL formula $\exists x. (\exists y.(\mathsf{F}\, a)(x, y)) \land (\forall z. (\mathsf{G}\, b)(z, x))$, which can be expressed in HyperSL as follows: $\exists x. \exists y. \forall y. (\mathsf{F}\, a_{\pi_1} \land \mathsf{G}\, b_{\pi_2})[\pi_1 : (x, y), \pi_2 : (z, x)]$.* △

Semantics. We fix a game structure $\mathcal{G} = (S, s_0, \mathbb{A}, \kappa, L)$. A *strategy assignment* is a partial mapping $\Delta : \mathcal{X} \rightharpoonup Str(\mathcal{G})$. We write $\{\}$ for the unique strategy assignment with an empty domain. In HyperSL, a path formula ψ refers to propositions on multiple path variables. We evaluate it in the context of a *path assignment* $\Pi : \mathcal{V} \rightharpoonup S^\omega$ mapping path variables to paths (similar to the semantics of HyperCTL* [27]). Given $j \in \mathbb{N}$, we define $\Pi[j, \infty]$ as the shifted assignment defined by $\Pi[j, \infty](\pi) := \Pi(\pi)[j, \infty]$. For a path formula ψ, we then define the semantics in the context of path assignment Π:

$\Pi \models_{\mathcal{G}} a_\pi$	iff	$a \in L\big(\Pi(\pi)(0)\big)$
$\Pi \models_{\mathcal{G}} \varphi_\pi$	iff	$\Pi(\pi)(0), \{\} \models_{\mathcal{G}} \varphi$
$\Pi \models_{\mathcal{G}} \psi_1 \land \psi_2$	iff	$\Pi \models_{\mathcal{G}} \psi_1$ and $\Pi \models_{\mathcal{G}} \psi_2$
$\Pi \models_{\mathcal{G}} \neg\psi$	iff	$\Pi \not\models_{\mathcal{G}} \psi$
$\Pi \models_{\mathcal{G}} \mathsf{X}\,\psi$	iff	$\Pi[1, \infty] \models_{\mathcal{G}} \psi$
$\Pi \models_{\mathcal{G}} \psi_1 \,\mathsf{U}\, \psi_2$	iff	$\exists j \in \mathbb{N}. \Pi[j, \infty] \models_{\mathcal{G}} \psi_2$ and $\forall 0 \le k < j. \Pi[k, \infty] \models_{\mathcal{G}} \psi_1$

The semantics for path formulas synchronously steps through all paths in Π and evaluate a_π on the path bound to π. State formulas are evaluated in a state $s \in S$ and strategy assignment Δ as follows:

$s, \Delta \models_{\mathcal{G}} \forall x. \varphi$	iff	$\forall f \in Str(\mathcal{G}). s, \Delta[x \mapsto f] \models_{\mathcal{G}} \varphi$
$s, \Delta \models_{\mathcal{G}} \exists x. \varphi$	iff	$\exists f \in Str(\mathcal{G}). s, \Delta[x \mapsto f] \models_{\mathcal{G}} \varphi$
$s, \Delta \models_{\mathcal{G}} \psi\big[\pi_k : \vec{x}_k\big]_{k=1}^m$	iff	

$$\Big[\pi_k \mapsto Play_{\mathcal{G}}\Big(s, \prod_{i \in Agts} \Delta(\vec{x}_k(i))\Big)\Big]_{k=1}^m \models_{\mathcal{G}} \psi$$

To resolve a formula $\psi\big[\pi_k : \vec{x}_k\big]_{k=1}^m$, we construct m paths (bound to $\pi_1, \ldots, \pi_m$), and evaluate ψ in the resulting path assignment. The

kth path (bound to π_k) is the play where each agent i plays strategy $\Delta(\vec{x}_k(i))$, i.e., the strategy currently bound to the strategy variable $\vec{x}_k(i)$. We write $\mathcal{G} \models \varphi$ if $s_0, \{\} \models_{\mathcal{G}} \varphi$, i.e., the initial state satisfies state formula φ.

5 EXPRESSIVENESS OF HYPERSL

The ability to compare multiple paths within a temporal formula makes HyperSL a powerful formalism that subsumes many existing logics. We only briefly mention some connections to existing logics. More details can be found in the full version [18].

5.1 SL and HyperSL

HyperSL naturally subsumes many (non-hyper) strategy logics [25, 45], which evaluate temporal properties on *individual* paths. We consider SL formulas defined by the following grammar:

$$\psi := a \mid \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{X}\,\psi \mid \psi \,\mathsf{U}\, \psi$$
$$\varphi := \psi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \forall x.\, \varphi \mid \exists x.\, \varphi \mid (i, x)\varphi$$

where $a \in AP$, $x \in \mathcal{X}$, and $i \in Agts$. We assume that nested state formulas are closed. In this SL, we can quantify over strategies and *bind* a strategy x to agent i using (i, x); see the full version [18] for the full semantics. We can show the following:

Lemma 1. *For any SL formula φ there exists a HyperSL formula φ' such that for any CGS $\mathcal{G}$, $\mathcal{G} \models_{SL} \varphi$ iff $\mathcal{G} \models \varphi'$.*

Proof Sketch. We use a unique path variable $\dot{\pi}$. During translation, we track the current strategy (variable) for each agent and construct $\dot{\pi}$ using the resulting strategy profile. □

Example 2. *Consider the formula $\exists x.\, \forall y.\, (1, x)(2, y)(3, y)\, \mathsf{G}\,\mathsf{F}\, a$. We can express this formula in HyperSL as $\exists x.\, \exists y.\, \big(\mathsf{G}\,\mathsf{F}\, a_{\dot{\pi}}\big)[\dot{\pi} : (x, y, y)]$ where (x, y, y) denotes the strategy profile mapping agent 1 to x, and agents 2 and 3 to y.* △

5.2 HyperATL* and HyperSL

Compared to SL, ATL* [2] offers a weaker (implicit) form of strategic reasoning. The ATL* formula $\langle\langle A \rangle\rangle \psi$ expresses that the agents in $A \subseteq Agts$ have a joint strategy to ensure path formula ψ [2]. HyperATL* [14, 17] is an extension of ATL* that can express hyperproperties, generated by the following grammar:

$$\psi := a_\pi \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{X}\,\psi \mid \psi \,\mathsf{U}\, \psi$$
$$\varphi := \langle\langle A \rangle\rangle \pi.\, \varphi \mid [\![A]\!] \pi.\, \varphi \mid \psi$$

where $a \in AP$, $\pi \in \mathcal{V}$, and $A \subseteq Agts$. Formula $\langle\langle A \rangle\rangle \pi.\, \varphi$ states that the agents in A have a strategy such that any path under that strategy, when bound to path variable π, satisfies the remaining formula φ. Likewise, $[\![A]\!] \pi.\, \varphi$ states that, no matter what strategy the agents in A play, some compatible path, when bound to π, satisfies φ. See the full version [18] for the full HyperATL* semantics. We can show the following:

Lemma 2. *For any HyperATL* formula φ there exists a HyperSL formula φ' such that for any CGS $\mathcal{G}$, $\mathcal{G} \models_{HyperATL^*} \varphi$ iff $\mathcal{G} \models \varphi'$.*

Proof Sketch. Similar to the translation of ATL* to SL [25, 45], we translate each HyperATL* quantifier $\langle\langle A \rangle\rangle \pi$ (resp. $[\![A]\!] \pi$) using existential (resp. universal) quantification over fresh strategies for all agents in A, followed by universal (resp. existential) quantification over strategies for agents in $Agts \setminus A$ and use these strategies to construct path π. □

Example 3. *Consider the HyperATL* formula $\langle\langle \{1, 2\} \rangle\rangle \pi_1.\, \langle\langle \{3\} \rangle\rangle \pi_2.\, (a_{\pi_1} \,\mathsf{U}\, b_{\pi_2})$. We can express this in HyperSL as $\exists x_1, x_2.\, \forall x_3.\, \exists y_3.\, \forall y_1, y_2.\, (a_{\pi_1} \,\mathsf{U}\, b_{\pi_2})\big[\pi_1 : (x_1, x_2, x_3), \pi_2 : (y_1, y_2, y_3)\big]$.* △

By Lemma 2, HyperSL thus captures the various security hyperproperties (such as non-deducibility of strategies [54] and simulation security [51]) that can be expressed in HyperATL* [14]. We can extend Lemma 2 further to also capture the strategy sharing constraints found in HyperATL^*_S [19].

Lemma 3. *For any HyperATL^*_S formula φ there exists a HyperSL formula φ' such that for any CGS $\mathcal{G}$, $\mathcal{G} \models_{HyperATL^*_S} \varphi$ iff $\mathcal{G} \models \varphi'$.*

Moreover, HyperSL can express properties that go well beyond the strict $\exists\forall$ and $\forall\exists$ quantifier alternation found in HyperATL* and HyperATL^*_S (as, e.g., needed for Nash equilibria).

5.3 Imperfect Information and HyperSL

In recent years, much effort has been made to study strategic behavior under *imperfect information* [9–12, 30, 36]. In such a setting, an agent acts strategically (i.e., decides on an action based on its past experience) but only observes parts of the overall system. Perhaps surprisingly, HyperSL is expressive enough to allow reasoning under imperfect information despite having a semantics with complete information (cf. Section 4). Concretely, we consider strategy logic under imperfect information (SL_{ii}), an extension of SL with imperfect information [12, 13] defined as follows:

$$\psi := a \mid \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{X}\,\psi \mid \psi \,\mathsf{U}\, \psi$$
$$\varphi := \psi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \forall x^o.\, \varphi \mid \exists x^o.\, \varphi \mid (i, x)\varphi$$

where $a \in AP$, $x \in \mathcal{X}$, $i \in Agts$, and $o \in Obs$ is an *observation* that gets attached to each strategy. SL_{ii} is evaluated on *CGSs under partial observation*, which are pairs $(\mathcal{G}, \{\sim_o\}_{o \in Obs})$ consisting of a CGS $\mathcal{G} = (S, s_0, \mathbb{A}, \kappa, L)$ and an observation relation $\sim_o \subseteq S \times S$ for each observation $o \in Obs$. If $s \sim_o s'$, then s and s' appear indistinguishable for a strategy with observation o. See the full version [18] for the full semantics.

We can effectively encode each MC instance of SL_{ii} into an equisatisfiable HyperSL instance (Note that the MAS models of SL_{ii} and HyperSL are different, so we cannot translate the formula directly but translate both the formula and the model).

Theorem 1. *For any SL_{ii} MC instance $\big((\mathcal{G}, \{\sim_o\}_{o \in Obs}), \varphi\big)$, we can effectively compute a HyperSL MC instance $\big(\mathcal{G}', \varphi'\big)$, such that $(\mathcal{G}, \{\sim_o\}_{o \in Obs}) \models_{SL_{ii}} \varphi$ iff $\mathcal{G}' \models \varphi'$.*

Proof Sketch. The key observation is that a strategy acting under imperfect information *is* a hyperproperty [20, 22]: A strategy f acts under observation $o \in Obs$ iff on any two finite paths under f the action chosen by f is the same, provided the two paths are indistinguishable w.r.t. $\sim_o$. We can extend the CGS $\mathcal{G}$ so that the above is easily expressible in HyperSL. We can then restrict quantification to strategies under an arbitrary observation and use a similar translation to the one used in Lemma 1. □

As model checking of SL_{ii} is undecidable [12], we get:

COROLLARY 1. *Model checking of HyperSL is undecidable.*

6 MODEL CHECKING OF HYPERSL

While HyperSL MC is undecidable in general (cf. Corollary 1), we can identify fragments for which MC is possible. For this, we cannot follow the approach of existing MC algorithms for (variants of) non-hyper SL, which use tree automata to summarize strategies [25, 45]. For example, given an atomic state formula $\psi[\pi_k : \vec{x}_k]_{k=1}^m$, we cannot construct a tree automaton that accepts all strategies that fulfill ψ. This automaton would need to *compare* (and thus traverse) multiple paths in a tree at the same time. Instead – given the "hyper" origins of our logic – we approach the MC problem by focusing on the interactions of its path variables and use *word* automata to summarize satisfying path assignments.

Throughout this section, we assume that all strategy variables are α-renamed such that no variable is quantified more than once.

6.1 HyperSL[SPE]

We call the fragment of HyperSL we study in this section HyperSL[SPE] – short for HyperSL with **S**ingle **P**ath **E**limination.

DEFINITION 1. *A HyperSL[SPE] formula has the form*

$$\varphi = \flat_1 \dots \flat_m.\, \psi\big[\pi_k : \vec{x}_k\big]_{k=1}^m,$$

where $\flat_1, \dots, \flat_m$ *are blocks of strategy quantifiers and for each* $1 \leq k \leq m$ *and* $i \in Agts$, *strategy variable* $\vec{x}_k(i)$ *is quantified in* $\flat_k$. *We refer to* m *as the* block-rank *of* φ.

Intuitively, the definition states that we can partition the quantifier prefix into smaller blocks where the variables quantified in each block $\flat_k$ can be used to eliminate (construct) the (unique) path variable π_k. We will exploit this restriction during model-checking: we can eliminate each block of quantifiers incrementally: as all strategies quantified in block $\flat_k$ are only needed for path π_k, we can "construct" π_k, and afterward forget about the strategies we have used. Note that the definition of HyperSL[SPE] only depends on the quantifier prefix and the path each strategy variable is used on; it does not make any assumption on the structure of ψ.

EXAMPLE 4. *Consider the following (abstract) HyperSL formula, where* ψ *is an arbitrary LTL formula over* π_1, π_2.

$$\underbrace{\exists c.}_{\flat_1}\, \underbrace{\exists z. \forall w. \exists v.}_{\flat_2}\, \psi \begin{bmatrix} \pi_1 : (c, c, c, c) \\ \pi_2 : (w, z, v, v) \end{bmatrix}$$

This formula is a HyperSL[SPE] formula: The first block $\flat_1$ *consists of strategy variable* c *and constructs* π_1, *and the second block* $\flat_2$ *constructs* π_2. *The block-rank of this formula is* 2. △

6.2 Expressiveness Of HyperSL[SPE]

Before we outline our model-checking algorithm for HyperSL[SPE] formulas, we point to some (fragments of) other logics that fall within HyperSL[SPE].

HyperATL and HyperSL[SPE].* When translating HyperATL* (or HyperATL$^*_S$) formulas into HyperSL (cf. Lemmas 2 and 3), each quantifier $\langle\langle A \rangle\rangle \pi$ (resp. $[\![A]\!]\pi$) is replaced by a $\exists^*\forall^*$ (resp. $\forall^*\exists^*$) block of strategy quantifiers that are used to construct π (and only π). The resulting formula is thus a HyperSL[SPE] formula.

SL[1G] and HyperSL[SPE]. SL[1G] is a fragment of SL that allows a prefix of strategy quantifier and agent bindings followed by a single path formula (with no nested agent binding) [24, 45–47]. When translating SL[1G] into HyperSL, we obtain a formula of the form $\mathbb{Q}_1 x_1 \dots \mathbb{Q}_m x_m.\, \psi[\pi : \vec{x}]$ (cf. Lemma 1), which is trivially HyperSL[SPE] as there is a single path variable (with block-rank 1).

Beyond HyperATL$^*_S$ and SL[1G]. Additionally, HyperSL[SPE] captures interesting hyperproperties that could not be captured in existing logics:

EXAMPLE 5. *Assume a MAS with* $Agts = \{r, a, ndet\}$ *describing a planning task between a robot* r *that wants to reach a state where AP goal* $\in$ *AP holds, and an adversary* a *that wants to prevent the robot from reaching the goal. In each step, agent* r *can select a direction to move in, and* a *can choose a direction it wants to push the robot to. Each combination of actions of* r *and* a *results in a set of potential successor locations, and the nondeterminism agent* ndet *decides which of those locations the robot actually moves to. We want to check if agent* r *has a winning strategy that* can *reach the goal against all possible behaviors of agent* a, *i.e.,* r *needs to reach the goal under favorable non-deterministic outcomes. We can express this (non-hyper) property in HyperSL[SPE] as*

$$\exists x. \forall y. \exists z. \big(\mathsf{F}\, goal_\pi\big)\big[\pi : (x, y, z)\big],$$

where we write (x, y, z) *for the strategy profile that assigns agent* r *to* x, *agent* a *to* y, *and agent* ndet *to* z. *In HyperSL[SPE], we can additionally state that* r *should reach the goal* as fast as possible, *i.e., at least as fast as any path in the MAS:*

$$\exists x. \forall y. \exists z. \forall a. \forall b. \forall c. (\neg goal_{\pi'}) \,\mathsf{U}\, goal_\pi \begin{bmatrix} \pi : (x, y, z) \\ \pi' : (a, b, c) \end{bmatrix}$$

Here, we quantify over any potential different path π' *and state that* π *is at least as fast as* π'. *Such requirements cannot be expressed in SL (even in quantitative versions like SL[$\mathcal{F}$]), nor can they be expressed in HyperATL* or HyperATL$^*_S$.* △

6.3 Summarizing Path Assignments

In the remainder of this section, we prove the following:

THEOREM 2. *Model checking for HyperSL[SPE] is decidable.*

We fix a CGS $\mathcal{G} = (S, s_0, \mathbb{A}, \kappa, L)$ and state $\dot{s} \in S$, and let $\varphi = \flat_1 \dots \flat_m.\, \psi\big[\pi_k : \vec{x}_k\big]_{k=1}^m$ be a HyperSL[SPE] formula. We want to check if $\dot{s}, \{\} \models_{\mathcal{G}} \varphi$, i.e., φ holds in state $\dot{s}$.

Zipping Path Assignments. The main idea of our algorithm is to summarize path assignments that satisfy subformulas of φ, similar to MC algorithms for HyperLTL, HyperCTL*, and HyperATL* [16, 17, 19, 33]. To enable automata-based reasoning about path assignments, i.e., mappings $\Pi : V \to S^\omega$ for some $V \subseteq \mathcal{V}$, we *zip* such an assignment into an infinite word. Concretely, given $\Pi : V \to S^\omega$ we define $zip(\Pi) \in (V \to S)^\omega$ as the infinite word of functions where $zip(\Pi)(j)(\pi) := \Pi(\pi)(j)$ for every $j \in \mathbb{N}$, i.e., the function in the jth step maps each path variable $\pi \in V$ to the jth state on the path bound to π.

Algorithm 1 Simulation construction for block elimination.

```
1 def simulate(𝒢 = (S, s₀, 𝔸, κ, L), ṡ, π, x⃗, ♭ = ℚ₁x₁ … ℚₙxₙ, 𝒜):
2   𝒜_det = (Q, q₀, δ, c) = toDPA(𝒜) // Using Proposition 1
3   ℬ = (Q × S, (q₀, ṡ), δ′, c′) where
4     c′(q, s) := c(q)
5     δ′((q, s), t⃗) := ⩕^{ℚ₁}_{a_{x₁} ∈ 𝔸} ⋯ ⩕^{ℚₙ}_{a_{xₙ} ∈ 𝔸} (δ(q, t⃗[π ↦ s]), κ(s, ∏_{i∈Agts} a_{x⃗(i)}))
6   return ℬ
```

Summary Automaton. Given a quantifier block $\flat = \mathbb{Q}_1 x_1 \ldots \mathbb{Q}_n x_n$ over strategy variables $x_1, \ldots, x_n$, we define $\widetilde{\flat}$ as the analogous block of quantification of strategies $f_{x_1}, \ldots, f_{x_n}$, i.e., $\widetilde{\flat} := \mathbb{Q}_1 f_{x_1} \in \mathit{Str}(\mathcal{G}) \ldots \mathbb{Q}_n f_{x_n} \in \mathit{Str}(\mathcal{G})$. At the core of our model-checking algorithm, we construct automata that accept (zippings of) partial satisfying path assignments. Formally:

Definition 2. *For $1 \leq k \leq m+1$, we say an automaton $\mathcal{A}$ over alphabet $(\{\pi_1, \ldots, \pi_{k-1}\} \to S)$ is a $(\mathcal{G}, \dot{s}, k)$-summary if for every path assignment $\Pi : \{\pi_1, \ldots, \pi_{k-1}\} \to S^\omega$ we have $\mathit{zip}(\Pi) \in \mathcal{L}(\mathcal{A})$ if and only if*

$$\widetilde{\flat_k} \cdots \widetilde{\flat_m}. \Pi\Big[\pi_j \mapsto \mathit{Play}_{\mathcal{G}}(\dot{s}, \prod_{i \in \mathit{Agts}} f_{\vec{x}_j(i)})\Big]_{j=k}^{m} \models_{\mathcal{G}} \psi.$$

That is, a $(\mathcal{G}, \dot{s}, k)$-summary accepts (the zipping of) a path assignment Π over paths $\pi_1, \ldots, \pi_{k-1}$ if – when simulating the quantification over strategies needed to construct paths $\pi_k, \ldots, \pi_m$ and adding them to Π – the body ψ of the formula is satisfied.

Example 6. *We illustrate the concept using the abstract formula from Example 4. A $(\mathcal{G}, \dot{s}, 3)$-summary is an automaton $\mathcal{A}_3$ over alphabet $(\{\pi_1, \pi_2\} \to S)$ such that for every $\Pi : \{\pi_1, \pi_2\} \to S^\omega$ we have $\mathit{zip}(\Pi) \in \mathcal{L}(\mathcal{A}_3)$ iff $\Pi \models_{\mathcal{G}} \psi$. A $(\mathcal{G}, \dot{s}, 2)$-summary is an automaton $\mathcal{A}_2$ over alphabet $(\{\pi_1\} \to S)$ such that for every $\Pi : \{\pi_1\} \to S^\omega$ we have $\mathit{zip}(\Pi) \in \mathcal{L}(\mathcal{A}_2)$ iff*

$$\exists f_z. \forall f_w. \exists f_v. \Pi\big[\pi_2 \mapsto \mathit{Play}_{\mathcal{G}}(\dot{s}, (f_w, f_z, f_v, f_v))\big] \models_{\mathcal{G}} \psi,$$

i.e., we mimic the quantification of block $\flat_2$ to construct path π_2 (using the quantified strategies $f_z, f_w, f_v \in \mathit{Str}(\mathcal{G})$) and add this path to Π (which already contains π_1). △

6.4 Constructing $(\mathcal{G}, \dot{s}, k)$-Summaries

We write $⩕^{\mathbb{Q}}$ for a conjunction ($\bigwedge$) if $\mathbb{Q} = \forall$ and a disjunction ($\bigvee$) if $\mathbb{Q} = \exists$. The backbone of our model-checking algorithm (which we present in Section 6.5) is an effective construction of a $(\mathcal{G}, \dot{s}, k)$-summary $\mathcal{A}_k$ for each $1 \leq k \leq m+1$. To construct these summaries, we simulate quantification over strategies. We describe this simulation construction in Algorithm 1. Before explaining the construction, we state the result of Algorithm 1 as follows:

Proposition 2. *Given $\dot{s} \in S, \pi \in \mathcal{V}$, a strategy profile $\vec{x} : \mathit{Agts} \to \mathcal{X}$, a quantifier block $\flat$ such that for every $i \in \mathit{Agts}$, $\vec{x}(i)$ is quantified in $\flat$, and an APA $\mathcal{A}$ over alphabet $(V \uplus \{\pi\} \to S)$. Let $\mathcal{B}$ be the results of* `simulate`$(\mathcal{G}, \dot{s}, \pi, \vec{x}, \flat, \mathcal{A})$. *Then for any path assignment $\Pi : V \to S^\omega$, we have $\mathit{zip}(\Pi) \in \mathcal{L}(\mathcal{B})$ iff*

$$\widetilde{\flat}. \mathit{zip}\Big(\Pi\big[\pi \mapsto \mathit{Play}_{\mathcal{G}}(\dot{s}, \prod_{i \in \mathit{Agts}} f_{\vec{x}(i)})\big]\Big) \in \mathcal{L}(\mathcal{A}). \qquad (1)$$

That is, the automaton $\mathcal{B}$ accepts the zipping of an assignment $\Pi : V \to S^\omega$ iff by simulating the quantifier prefix in $\flat$, we construct a path for π that, when added to Π, is accepted by $\mathcal{A}$. Note the similarity to Definition 2: In Definition 2 we simulate multiple quantifier blocks to construct paths $\pi_k, \ldots, \pi_m$ that, when added to Π, should satisfy the body ψ. In Proposition 2, we simulate a single path that, when added to Π, should be accepted by automaton $\mathcal{A}$. We will later use Proposition 2 to simulate one quantifier block at a time, eventually reaching an automaton required by Definition 2.

Before proving Proposition 2, let us explain the automaton construction in `simulate` (Algorithm 1). In Eq. (1), $\widetilde{\flat}$ quantifies over strategies in $\mathcal{G}$, which are infinite objects (function $S^+ \to \mathbb{A}$). The crucial point that we will exploit is that the underlying game the strategies operate on is *positionally determined*. The automaton we construct can, therefore, *simulate* the path π in $\mathcal{G}$ and select fresh actions in each step (instead of fixing strategies globally) [14, 17, 19]. To do this, we first translate the APA $\mathcal{A}$ to a DPA $\mathcal{A}_{det} = (Q, q_0, \delta, c)$ (in line 2). The new automaton $\mathcal{B}$ then simulates path π by tracking its current state in $\mathcal{G}$ and simultaneously tracks the current state of $\mathcal{A}_{det}$, thus operating on states in $Q \times S$. We start in state $(q_0, \dot{s})$, i.e., the initial state of $\mathcal{A}_{det}$ and the designed state $\dot{s}$ from which we want to start the simulation of π. The color of each state is simply the color of the automaton we are tracking, i.e., $c'(q, s) = c(q)$ (line 4). During each transition, we then update the current state of $\mathcal{A}_{det}$ and the state of the simulation (defined in line 5). Concretely, when in state (q, s), we read a letter $\vec{t} : V \to S$ that assigns states to all path variables in V (recall that the alphabet of $\mathcal{A}$ is $V \cup \{\pi\} \to S$ and the alphabet of $\mathcal{B}$ is $V \to S$). We update the state of $\mathcal{A}_{det}$ to $\delta(q, \vec{t}[\pi \mapsto s])$, i.e., we extend the input letter $\vec{t}$ with the current state s of the simulation of path π (note that $\vec{t}[\pi \mapsto s] : V \cup \{\pi\} \to S$). To update the simulation state s, we make use of the positional determinacy of the game: Instead of quantifying over strategies (as in Eq. (1)), we can quantify over actions in each step of the automaton. Concretely, for each universally quantified strategy variable in $\flat$, we pick an action conjunctively, and for each existentially quantified variable, we pick an action disjunctively. After we have picked actions $a_{x_1}, \ldots, a_{x_n}$ for all strategies quantified in $\flat$, we can update the state of the π-simulation by constructing the action assignment $\prod_{i \in \mathit{Agts}} a_{\vec{x}(i)}$, i.e., assign each agent the corresponding action, and obtain the next state using $\mathcal{G}$'s transition function κ.

Example 7. *Let us use Example 4 to illustrate the construction in Algorithm 1. Assume we are given an $(\mathcal{G}, \dot{s}, 3)$-summary $\mathcal{A}_3$ over alphabet $(\{\pi_1, \pi_2\} \to S)$, i.e., for every $\Pi : \{\pi_1, \pi_2\} \to S^\omega$, we have $\mathit{zip}(\Pi) \in \mathcal{L}(\mathcal{A}_3)$ iff $\Pi \models_{\mathcal{G}} \psi$ (cf. Example 6). We invoke* `simulate`$(\mathcal{G}, \dot{s}, \pi_2, \vec{x}, \flat_2, \mathcal{A}_3)$ *where $\vec{x} = (w, z, v, v)$ and $\flat_2 = \exists z \forall w \exists v$, and let (Q, q_0, δ, c) be the DPA equivalent to $\mathcal{A}_3$ (computed in line 2). In this case,* `simulate` *computes the APA $\mathcal{B} = (Q \times S, (q_0, \dot{s}), \delta', c')$ over alphabet $\{\pi_1\} \to S$ where $\delta'\big((q, s), \vec{t}\big)$ is defined as*

$$\bigvee_{a_z \in \mathbb{A}} \bigwedge_{a_w \in \mathbb{A}} \bigvee_{a_v \in \mathbb{A}} \Big(\delta(q, \vec{t}[\pi_2 \mapsto s]), \kappa\big(s, (a_w, a_z, a_v, a_v)\big)\Big).$$

That is, in each step, we disjunctively choose an action a_z (corresponding to the action selected by existentially quantified strategy z), conjunctively pick an action a_w (corresponding to the action selected by universally quantified strategy w), and finally disjunctively select

Algorithm 2 Model-checking algorithm for HyperSL[SPE].

```
1  def modelCheck(𝒢, ṡ, φ = 𝕓₁ ⋯ 𝕓ₘ. ψ[πₖ : x⃗ₖ]ᵐₖ₌₁):
2    // Assume ψ contains no nested state formulas
3    𝒜ₘ₊₁ = LTLtoAPA(ψ)
4    // 𝒜ₘ₊₁ is a (𝒢, ṡ, m + 1)-summary
5    for k from m to 1:
6      𝒜ₖ = simulate(𝒢, ṡ, πₖ, x⃗ₖ, 𝕓ₖ, 𝒜ₖ₊₁)
7      // 𝒜ₖ is a (𝒢, ṡ, k)-summary
8    if ℒ(𝒜₁) ≠ ∅ then
9      return SAT // ṡ, {} ⊨𝒢 φ
10   else
11     return UNSAT  // ṡ, {} ⊭𝒢 φ
```

action a_v. After we have fixed actions a_z, a_w and a_v, we take a step in $\mathcal{G}$ by letting each agent i play action $a_{\vec{x}(i)}$, i.e., agent 1 chooses action a_w, agent 2 chooses a_z, and agents 3 and 4 pick a_v. By Proposition 2, every $\Pi : \{\pi_1\} \to S^\omega$ satisfies $zip(\Pi) \in \mathcal{L}(\mathcal{B})$ iff

$$\exists f_z. \forall f_w. \exists f_v. zip\big(\Pi[\pi_2 \mapsto Play_{\mathcal{G}}(\dot{s}, (f_w, f_z, f_v, f_v))]\big) \in \mathcal{L}(\mathcal{A}_3)$$

which (by assumption on $\mathcal{A}_3$) holds iff

$$\exists f_z. \forall f_w. \exists f_v. \Pi[\pi_2 \mapsto Play_{\mathcal{G}}(\dot{s}, (f_w, f_z, f_v, f_v))] \models_{\mathcal{G}} \psi.$$

We have thus used `simulate` *(Algorithm 1) to compute a $(\mathcal{G}, \dot{s}, 2)$-summary from a $(\mathcal{G}, \dot{s}, 3)$-summary (cf. Example 6).* △

We can now formally prove Proposition 2:

Proof Sketch of Proposition 2. The idea of automaton $\mathcal{B}$ constructed in Algorithm 1 is to simulate the path that corresponds to path variable π. To argue that $\mathcal{B}$ expresses the desired language, we make use of the positional determinacy of concurrent parity games (CPG) [40]. A CPG is a simple multi-player game model where we can quantify over strategies for each of the players. For any fixed Π, we design an (infinite-state) CPG, that is won iff Eq. (1) holds. We then exploit the fact that CPGs are determined (cf. [40, Thm. 4.1]), i.e., instead of quantifying over entire strategies in the CPG, we can quantify over Skolem functions for actions in each step. This allows us to show that the CPG is won iff $\mathcal{B}$ has an accepting run (on the fixed Π), giving us the desired result. We refer the interested reader to the full version [18] for details. □

6.5 Model-Checking Algorithm

Equipped with the concept of $(\mathcal{G}, \dot{s}, k)$-summary and the simulation construction, we can now present our MC algorithm for HyperSL[SPE] in Algorithm 2. The `modelCheck` procedure is given a CGS $\mathcal{G}$, a state $\dot{s}$, and a HyperSL[SPE] formula φ, and checks if $\dot{s}, \{\} \models_{\mathcal{G}} \varphi$. Our algorithm assumes, w.l.o.g., that the path formula ψ contains no nested state formulas. In case there are nested state formulas, we can eliminate them iteratively: We recursively check each nested state formula on all states of the CGS, and label all states where the state formula holds with a fresh atomic proposition. In the path formula, we can then replace each state formula with a reference to the fresh atomic proposition. See, e.g., [19, 32] for details.

The main idea of our MC algorithm is to iteratively construct a $(\mathcal{G}, \dot{s}, k)$-summary $\mathcal{A}_k$ for each $1 \leq k \leq m + 1$. Initially, in line 4, we construct a $(\mathcal{G}, \dot{s}, m + 1)$-summary $\mathcal{A}_{m+1}$ using a standard construction to translate the LTL formula ψ to an APA over alphabet $(\{\pi_1, \dots, \pi_m\} \to S)$, as is, e.g., standard for HyperCTL* [33]. For each k from m to 1, we then use the $(\mathcal{G}, \dot{s}, k + 1)$-summary $\mathcal{A}_{k+1}$ to compute a $(\mathcal{G}, \dot{s}, k)$-summary $\mathcal{A}_k$ using the `simulate` construction from Algorithm 1 (similar to what we illustrated in Example 7). From Proposition 2, we can conclude the following invariant:

Table 1: We compare HyMASMC and MCMAS-SL[1G] on the scheduler problem from [24]. We give the size of the system ($|S|$), the size of the reachable fragment ($|S_{reach}|$), and the times in seconds (t). The timeout (TO) is set to 1 h.

n	$\lvert S \rvert$	$\lvert S_{reach} \rvert$	$t_{\text{MCMAS-SL[1G]}}$	t_{HyMASMC}
2	72	9	**0.1**	0.4
3	432	21	6.71	**1.9**
4	2592	49	313.7	**24.5**
5	15552	113	TO	**332.1**

Lemma 4. *In line 7, $\mathcal{A}_k$ is a $(\mathcal{G}, \dot{s}, k)$-summary.*

After the loop, we are thus left with a $(\mathcal{G}, \dot{s}, 1)$-summary $\mathcal{A}_1$ (over the simpleton alphabet $(\emptyset \to S)$) and can check if $\dot{s}, \{\} \models_{\mathcal{G}} \varphi$ by testing $\mathcal{A}_1$ for emptiness (line 8):

Lemma 5. *For any $(\mathcal{G}, \dot{s}, 1)$-summary $\mathcal{A}$, we have that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ if and only if $\dot{s}, \{\} \models_{\mathcal{G}} \varphi$.*

From Lemmas 4 and 5, it follows that `modelCheck`$(\mathcal{G}, \dot{s}, \varphi)$ returns SAT iff $\dot{s}, \{\} \models_{\mathcal{G}} \varphi$, proving Theorem 2.

6.6 Model-Checking Complexity

The determinization in line 2 of Algorithm 1 results in a DPA $\mathcal{A}_{det}$ of doubly exponential size (cf. Proposition 1). The size of $\mathcal{B}$ is then linear in the size of $\mathcal{A}_{det}$ and $\mathcal{G}$. In the worst case, each call of `simulate` thus increases the size of the automaton by two exponents. For a HyperSL[SPE] formula with block-rank m, `simulate` is called m times, so the final automaton $\mathcal{A}_1$ has, in the worst case, $2m$-exponential many states (in the size of ψ and $\mathcal{G}$). As we can check emptiness of APAs over the singleton alphabet $(\emptyset \to S)$ in polynomial time, we get:

Theorem 3. *Model checking for a HyperSL[SPE] formula with block-rank m is in $2m$-EXPTIME.*

From Lemma 2 and the lower bounds known for HyperATL* [17], it follows that our algorithm is asymptotically almost optimal:

Lemma 6. *Model checking for a HyperSL[SPE] formula with block-rank m is $(2m - 1)$-EXPSPACE-hard.*

6.7 Beyond HyperSL[SPE]

HyperSL[SPE] is defined purely in terms of the structure of the quantifier prefix. As soon as strategy variables are quantified in an order such that they cannot be grouped together, MC becomes, in general, undecidable: Already the simplest such property $\mathbb{Q}x.\mathbb{Q}y.\mathbb{Q}z.\mathbb{Q}w.\psi[\pi_1 : (x, z), \pi_2 : (y, w)]$, leads to undecidable MC (see the full version [18]). The fragment we have identified is thus the largest possible (when only considering the quantifier prefix). Any further study into decidable fragments of HyperSL needs to impose

Table 2: We check random formulas from the (Sec), (GE), and (Rnd) templates on the ISPL models from [39]. For each model and template, we sample 10 formulas and report the average time (in seconds).

Model	Sec	GE	$\mathbf{Rnd}_2$	$\mathbf{Rnd}_3$	$\mathbf{Rnd}_4$
BIT-TRANSMISSION	0.6	0.7	0.8	0.8	2.7
BOOK-STORE	0.4	0.4	0.4	0.5	0.5
CARD-GAME	0.4	0.5	0.4	0.5	0.5
DINING-CRYPTOGRAPHERS	0.6	2.7	11.4	22.6	10.3
MUDDY-CHILDREN	0.4	3.0	1.7	0.8	0.9
SIMPLE-CARD-GAME	0.3	3.4	2.9	25.3	32.6
SOFTWARE-DEVELOPMENT	-	-	-	-	-
STRONGLY-CONNECTED	0.6	0.8	0.8	1.7	3.2
TIANJI-HORSE-RACING	0.4	0.5	0.4	0.5	0.5

restrictions beyond the prefix and, e.g., analyze how different path variables are related within an LTL path formula (see also Section 8).

7 IMPLEMENTATION AND EXPERIMENTS

We have implemented our HyperSL[SPE] model-checking algorithm in the `HyMASMC` tool [19].

7.1 Model-Checking For Strategy Logic

We compare `HyMASMC` against `MCMAS-SL[1G]` [24] on (non-hyper) SL[1G] properties (cf. Section 6.2). In Table 1, we depict the verification times for the scheduling problem from [24] (which can be expressed in SL[1G] and ATL*). As in [19], we observe that `HyMASMC` performs much faster than `MCMAS-SL[1G]`, which we largely accredit to `HyMASMC`'s efficient automata backend using `spot` [31]. Note that we use `MCMAS-SL[1G]` and `HyMASMC` directly on the original model, i.e., we did not perform any prepossessing using, e.g., abstraction techniques [5, 6, 8] (which would reduce the system size and make the verification more scaleable for both tools).

7.2 Model-Checking For Hyperproperties

In a second experiment, we demonstrate that `HyMASMC` can verify hyperproperties on various MASs from the literature. We use the ISPL models from the `MCMAS` benchmarks suit [39], and generate random HyperSL[SPE] formulas from various property templates:

- **(Sec):** We check if some agent i can reach some target state without leaking information about some secret AP via some observable AP. Concretely, we check if i can play such that on some other path, the same observation sequence is coupled with a different high-security input, a property commonly referred to as *non-inference* [43] or *opacity* [52, 55].
- **(GE):** We check if a given SL[1G] formula holds on all input sequences for which *some* winning output sequence exists, as is, e.g., required in *good-enough* synthesis [1, 3].
- **(Rnd):** We randomly generate HyperSL[SPE] formulas with block-rank 2, 3, and 4 (called $\mathbf{Rnd}_2$, $\mathbf{Rnd}_3$, and $\mathbf{Rnd}_4$, respectively).

We depict the results in Table 2, demonstrating that `HyMASMC` can handle most instances. The only exception is the SOFTWARE-DEVELOPMENT model, which includes ≈15k states and is therefore too large for an automata-based representation.

Table 3: We use `HyMASMC` to solve the optimal adversarial planning problem (cf. Example 5) for varying sizes. Times are given in seconds, and the TO is set to 120 sec.

Size	40	50	60	70	80	90	100	110	120
t	14.2	22.0	31.2	42.5	57.6	70.1	86.8	104.6	TO

We stress that we do not claim that all formulas in each of the templates model realistic properties in each of the systems. Rather, our evaluation **(1)** demonstrates that HyperSL[SPE] can express interesting properties, and **(2)** empirically shows that `HyMASMC` can check such properties in existing ISPL models (confirming this via further real-world scenarios is interesting future work).

7.3 Model-Checking For Optimal Planning

In our last experiment, we challenge `HyMASMC` with planning examples as those outlined in Example 5. We randomly generate planning instances between the robot r, adversary a, and $ndet$, and check if robot r *can* reach the goal following some shortest path in the problem. For a varying size n, we randomly create 10 planning instances with n states. We report the verification times in Table 3. With increasing size, the running time of `HyMASMC` clearly increases, but the increase seems to be quadratic rather than exponential.

8 CONCLUSION AND FUTURE WORK

We have presented HyperSL, a new temporal logic that extends strategy logic with the ability to reason about hyperproperties. HyperSL can express complex properties in MASs that require a combination of strategic reasoning and hyper-requirements (such as optimalilty, GE, non-interference, and quantitative Nash equilibria); many of which were out of reach of existing logics. As such, HyperSL can serve as a unifying foundation for an exact exploration of the interaction of strategic behavior with hyperproperties, and provides a formal language to express (un)decidability results. Moreover, we have taken a first step towards the ambitious goal of automatically model-checking HyperSL. Our fragment HyperSL[SPE] subsumes many relevant other logics and captures unique properties not expressible in existing frameworks. Our implementation in `HyMASMC` shows that our MC approach is practical in small MASs.

A particularly interesting future direction is to search for further fragments of HyperSL with decidable model checking. As argued in Section 6.7, any such fragment needs to take the structure of the LTL-formula(s) into account. For example, Mogavero et al. [46] showed that SL[CG] (a fragment of SL that only allows conjunctions of goal formulas) still admits behavioral strategies (i.e., strategies that do not depend on future or counterfactual decisions of other strategies). When extending this to our hyper setting, it seems likely that if a strategy is used on multiple path variables, but these paths occur in disjoint conjuncts of path formulas, MC remains decidable. We leave such extensions as future work.

ACKNOWLEDGMENTS

This work was supported by the European Research Council (ERC) Grant HYPER (101055412), and by the German Research Foundation (DFG) as part of TRR 248 (389792660).

REFERENCES

[1] Shaull Almagor and Orna Kupferman. 2020. Good-Enough Synthesis. In *International Conference on Computer Aided Verification, CAV 2020*.

[2] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. 2002. Alternating-time temporal logic. *J. ACM* (2002).

[3] Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. 2021. Best-Effort Synthesis: Doing Your Best Is Not Harder Than Giving Up. In *International Joint Conference on Artificial Intelligence, IJCAI 2021*.

[4] Benjamin Aminof, Marta Kwiatkowska, Bastien Maubert, Aniello Murano, and Sasha Rubin. 2019. Probabilistic Strategy Logic. In *International Joint Conference on Artificial Intelligence, IJCAI 2019*.

[5] Thomas Ball and Orna Kupferman. 2006. An Abstraction-Refinement Framework for Multi-Agent Systems. In *Symposium on Logic in Computer Science LICS 2006*.

[6] Francesco Belardinelli, Angelo Ferrando, Wojciech Jamroga, Vadim Malvone, and Aniello Murano. 2023. Scalable Verification of Strategy Logic through Three-Valued Abstraction. In *International Joint Conference on Artificial Intelligence, IJCAI 2023*.

[7] Francesco Belardinelli, Sophia Knight, Alessio Lomuscio, Bastien Maubert, Aniello Murano, and Sasha Rubin. 2021. Reasoning About Agents That May Know Other Agents' Strategies. In *International Joint Conference on Artificial Intelligence, IJCAI 2021*.

[8] Francesco Belardinelli and Alessio Lomuscio. 2017. Agent-based Abstractions for Verifying Alternating-time Temporal Logic with Imperfect Information. In *Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*.

[9] Francesco Belardinelli, Alessio Lomuscio, and Vadim Malvone. 2019. An Abstraction-Based Method for Verifying Strategic Properties in Multi-Agent Systems with Imperfect Information. In *Conference on Artificial Intelligence, AAAI 2019*.

[10] Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. 2017. Verification of Multi-agent Systems with Imperfect Information and Public Actions. In *Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*.

[11] Raphaël Berthon, Bastien Maubert, and Aniello Murano. 2017. Decidability Results for ATL* with Imperfect Information and Perfect Recall. In *Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*.

[12] Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Y. Vardi. 2017. Strategy logic with imperfect information. In *Symposium on Logic in Computer Science, LICS 2017*.

[13] Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Y. Vardi. 2021. Strategy Logic with Imperfect Information. *ACM Trans. Comput. Log.* (2021).

[14] Raven Beutner and Bernd Finkbeiner. 2021. A Temporal Logic for Strategic Hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2021*.

[15] Raven Beutner and Bernd Finkbeiner. 2022. Software Verification of Hyperproperties Beyond k-Safety. In *International Conference on Computer Aided Verification, CAV 2022*.

[16] Raven Beutner and Bernd Finkbeiner. 2023. AutoHyper: Explicit-State Model Checking for HyperLTL. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2023*.

[17] Raven Beutner and Bernd Finkbeiner. 2023. HyperATL*: A Logic for Hyperproperties in Multi-Agent Systems. *Log. Methods Comput. Sci.* (2023).

[18] Raven Beutner and Bernd Finkbeiner. 2024. Hyper Strategy Logic. *CoRR* (2024).

[19] Raven Beutner and Bernd Finkbeiner. 2024. On Alternating-Time Temporal Logic, Hyperproperties, and Strategy Sharing. In *Conference on Artificial Intelligence, AAAI 2024*.

[20] Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. 2023. Second-Order Hyperproperties. In *International Conference on Computer Aided Verification, CAV 2023*.

[21] Patricia Bouyer, Orna Kupferman, Nicolas Markey, Bastien Maubert, Aniello Murano, and Giuseppe Perelli. 2019. Reasoning about Quality and Fuzziness of Strategic Behaviours. In *International Joint Conference on Artificial Intelligence, IJCAI 2019*.

[22] Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. 2015. Unifying Hyper and Epistemic Temporal Logics. In *International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2015*.

[23] Nils Bulling and Wojciech Jamroga. 2014. Comparing variants of strategic ability: how uncertainty and memory influence general properties of games. *Auton. Agents Multi Agent Syst.* (2014).

[24] Petr Cermák, Alessio Lomuscio, and Aniello Murano. 2015. Verifying and Synthesising Multi-Agent Systems against One-Goal Strategy Logic Specifications. In *Conference on Artificial Intelligence, AAAI 2015*.

[25] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. 2010. Strategy logic. *Inf. Comput.* (2010).

[26] Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. 2012. Continuity and robustness of programs. *Commun. ACM* (2012).

[27] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. 2014. Temporal Logics for Hyperproperties. In *International Conference on Principles of Security and Trust, POST 2014*.

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

[29] Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. 2019. The Hierarchy of Hyperlogics. In *Symposium on Logic in Computer Science, LICS 2019*.

[30] Catalin Dima and Ferucio Laurentiu Tiplea. 2011. Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. *CoRR* (2011).

[31] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. 2022. From Spot 2.0 to Spot 2.10: What's New?. In *International Conference on Computer Aided Verification, CAV 2022*.

[32] E. Allen Emerson and Joseph Y. Halpern. 1986. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *J. ACM* (1986).

[33] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. 2015. Algorithms for Model Checking HyperLTL and HyperCTL*. In *International Conference on Computer Aided Verification, CAV 2015*.

[34] Bernd Finkbeiner and Martin Zimmermann. 2017. The First-Order Logic of Hyperproperties. In *Symposium on Theoretical Aspects of Computer Science, STACS 2017*.

[35] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. 2020. Propositional Dynamic Logic for Hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2020*.

[36] Sophia Knight and Bastien Maubert. 2019. Dealing with imperfect information in Strategy Logic. *CoRR* (2019).

[37] François Laroussinie and Nicolas Markey. 2015. Augmenting ATL with strategy contexts. *Inf. Comput.* (2015).

[38] François Laroussinie, Nicolas Markey, and Arnaud Sangnier. 2015. ATLsc with partial observation. In *International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2015*.

[39] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2009. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *International Conference on Computer Aided Verification, CAV 2009*.

[40] Vadim Malvone, Aniello Murano, and Loredana Sorrentino. 2016. Concurrent Multi-Player Parity Games. In *International Conference on Autonomous Agents & Multiagent Systems, AAMAS 2016*.

[41] Bastien Maubert and Aniello Murano. 2018. Reasoning about Knowledge and Strategies under Hierarchical Information. In *International Conference on Principles of Knowledge Representation and Reasoning, KR 2018*.

[42] Daryl McCullough. 1988. Noninterference and the composability of security properties. In *Symposium on Security and Privacy, SP 1988*.

[43] John McLean. 1994. A general theory of composition for trace sets closed under selective interleaving functions. In *Symposium on Research in Security and Privacy, SP 1994*.

[44] Satoru Miyano and Takeshi Hayashi. 1984. Alternating Finite Automata on omega-Words. *Theor. Comput. Sci.* (1984).

[45] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM Trans. Comput. Log.* (2014).

[46] Fabio Mogavero, Aniello Murano, and Luigi Sauro. 2013. On the Boundary of Behavioral Strategies. In *Symposium on Logic in Computer Science, LICS 2013*.

[47] Fabio Mogavero, Aniello Murano, and Luigi Sauro. 2014. A Behavioral Hierarchy of Strategy Logic. In *International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2014*.

[48] John F Nash Jr. 1950. Equilibrium points in n-person games. *Proceedings of the national academy of sciences* (1950).

[49] Marc Pauly and Rohit Parikh. 2003. Game Logic - An Overview. *Stud Logica* (2003).

[50] Nir Piterman. 2007. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. *Log. Methods Comput. Sci.* (2007).

[51] Andrei Sabelfeld. 2003. Confidentiality for Multithreaded Programs via Bisimulation. In *International Conference on Perspectives of Systems Informatics, PSI 2003*.

[52] Anooshiravan Saboori and Christoforos N. Hadjicostis. 2013. Verification of initial-state opacity in security applications of discrete event systems. *Inf. Sci.* (2013).

[53] Moshe Y. Vardi. 1995. Alternating Automata and Program Verification. In *Computer Science Today: Recent Trends and Developments*.

[54] J. Todd Wittbold and Dale M. Johnson. 1990. Information Flow in Nondeterministic Systems. In *Symposium on Security and Privacy, SP 1990*.

[55] Kuize Zhang, Xiang Yin, and Majid Zamani. 2019. Opacity of Nondeterministic Transition Systems: A (Bi)Simulation Relation Approach. *IEEE Trans. Autom. Control.* (2019).