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

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) [24, 45] reason about the temporal interaction of such agents and allow for rigorous correctness guarantees using techniques such as model-checking. *Hyperproperties* [27] 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 [25].

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 [24, 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 $\{1, \ldots, n\}$ and LTL properties $\psi_1, \ldots, \psi_n$ 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:

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

where we abbreviate the strategy profiles $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{x}[i \mapsto y] = (x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$. In the variant SL we consider here (similar to the SL by Chatterjee et al. [24]), 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

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* [26] and HyperATL* [14, 17]). In HyperSL, atomic formulas have the form $\psi[\pi_1 : \vec{x}_1, \ldots, \pi_m : \vec{x}_m]$ where $\pi_1, \ldots, \pi_m$ are path variables, $\vec{x}_1, \ldots, \vec{x}_m$ are strategy profiles (assigning a strategy to each agent), and ψ is an LTL formula where atomic propositions are indexed by path variables from $\pi_1, \ldots, \pi_m$. The formula states that the plays resulting from strategy profiles $\vec{x}_1, \ldots, \vec{x}_m$, when bound to $\pi_1, \ldots, \pi_m$, (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:

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

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* [26], HyperATL* [14, 17], and HyperATL*$_S$ [18] (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 [26], HyperATL* [14, 17], and HyperATL*$_S$ [18], 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 [18] 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 [20]. 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[$\mathcal{F}$] [20] 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, 22, 29, 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, 26, 28, 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*$_S$ [18] 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, 18]. 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 $Agts = \{1, \ldots, n\}$. 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 $\psi_1 \, W \, \psi_2$ 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 $p(j + 1) = \kappa(p(j), \prod_{i \in Agts} f_i(p[0, j]))$. 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 $\delta(q, l) = q_1 \vee (q_2 \wedge q_3)$, 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 Appendix A. 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 [24, 45]) with the ability to express hyperproperties (as possible in logics such as HyperCTL* [26]). 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:

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

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 $\vee, \to, \leftrightarrow$, 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 $\exists x. (\exists y.(\mathsf{F}\,a)(x,y)) \wedge (\forall z. (\mathsf{G}\,b)(z,x))$, which can be expressed in HyperSL as follows:* $\exists x. \exists y. \forall y. (\mathsf{F}\,a_{\pi_1} \wedge \mathsf{G}\,b_{\pi_2})[\pi_1 : (x,y), \pi_2 : (z,x)]$. $\triangle$

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* $\Pi : \mathcal{V} \to S^{\omega}$ mapping path variables to paths (similar to the semantics of HyperCTL* [26]). 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	$\Pi(\pi)(0), \{\} \models_{\mathcal{G}} \varphi$
$\Pi \models_{\mathcal{G}} \psi_1 \wedge \psi_2$	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 $\psi[\pi_k : \vec{x}_k]_{k=1}^m$, 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 Appendix B.

5.1 SL and HyperSL

HyperSL naturally subsumes many (non-hyper) strategy logics [24, 45], which evaluate temporal properties on *individual* paths. We consider SL formulas defined by the following grammar:

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

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 Appendix B.1 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:

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

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 Appendix B.2 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 [24, 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 $\langle\!\langle \{1, 2\} \rangle\!\rangle \pi_1.\,\langle\!\langle \{3\} \rangle\!\rangle$ $\pi_2.\,(a_{\pi_1} \cup b_{\pi_2})$. We can express this in HyperSL as $\exists x_1, x_2.\,\forall x_3.\,\exists y_3.$ $\forall y_1, y_2.\,(a_{\pi_1} \cup b_{\pi_2})\big[\pi_1 : (x_1, x_2, x_3), \pi_2 : (y_1, y_2, y_3)\big]$.* △

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$_S^*$ [18].

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, 29, 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:

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

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 Appendix B.3 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 $((\mathcal{G}, \{\sim_o\}_{o \in Obs}), \varphi)$, we can effectively compute a HyperSL MC instance $(\mathcal{G}', \varphi')$, 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 [19, 21]: 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 [24, 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*
$$\varphi = \flat_1 \ldots \flat_m. \, \psi \big[\pi_k : \vec{x}_k \big]_{k=1}^m,$$
where $\flat_1, \ldots, \flat_m$ 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) [23, 45–47]. When translating SL[1G] into HyperSL, we obtain a formula of the form $\mathbb{Q}_1 x_1 \ldots \mathbb{Q}_m x_m. \psi[\pi : \vec{x}]$ (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*
$$\exists x. \forall y. \exists z. \big(F \, goal_\pi \big) \big[\pi : (x, y, z) \big],$$
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:
$$\exists x. \forall y. \exists z. \forall a. \forall b. \forall c. \, (\neg goal_{\pi'}) \cup goal_\pi \begin{bmatrix} \pi : (x, y, z) \\ \pi' : (a, b, c) \end{bmatrix}$$
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 $\varphi = \flat_1 \ldots \flat_m. \psi [\pi_k : \vec{x}_k]_{k=1}^m$ 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* [16–18, 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, b = 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 $b = \mathbb{Q}_1 x_1 \ldots \mathbb{Q}_n x_n$ over strategy variables $x_1, \ldots, x_n$, we define $\widetilde{b}$ as the analogous block of quantification of strategies $f_{x_1}, \ldots, f_{x_n}$, i.e., $\widetilde{b} := \mathbb{Q}_1 f_{x_1} \in Str(\mathcal{G}) \ldots \mathbb{Q}_n f_{x_n} \in Str(\mathcal{G})$. At the core of our model-checking algorithm, we construct automata that accept (zippings of) partial satisfying path assignments. Formally:

DEFINITION 2. *For $1 \le k \le m + 1$, 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 $zip(\Pi) \in \mathcal{L}(\mathcal{A})$ if and only if*

$$\widetilde{b}_k \cdots \widetilde{b}_m. \Pi\left[\pi_j \mapsto Play_{\mathcal{G}}\left(\dot{s}, \prod_{i\in Agts} f_{\vec{x}_j(i)}\right)\right]_{j=k}^m \models_{\mathcal{G}} \psi.$$

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 $zip(\Pi) \in \mathcal{L}(\mathcal{A}_3)$ 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 $zip(\Pi) \in \mathcal{L}(\mathcal{A}_2)$ iff*

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

i.e., we mimic the quantification of block b_2 to construct path π_2 (using the quantified strategies $f_z, f_w, f_v \in Str(\mathcal{G})$) 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 $1 \le k \le m + 1$. 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 $\vec{x} : Agts \to X$, a quantifier block b such that for every $i \in Agts$, $\vec{x}(i)$ is quantified in b, and an APA $\mathcal{A}$ over alphabet $(V \uplus \{\pi\} \to S)$. Let $\mathcal{B}$ be the results of $\mathtt{simulate}(\mathcal{G}, \dot{s}, \pi, \vec{x}, b, \mathcal{A})$. Then for any path assignment $\Pi : V \to S^\omega$, we have $zip(\Pi) \in \mathcal{L}(\mathcal{B})$ iff*

$$\widetilde{b}. zip\left(\Pi\left[\pi \mapsto Play_{\mathcal{G}}\left(\dot{s}, \prod_{i\in Agts} f_{\vec{x}(i)}\right)\right]\right) \in \mathcal{L}(\mathcal{A}). \quad (1)$$

That is, the automaton $\mathcal{B}$ accepts the zipping of an assignment $\Pi : V \to S^\omega$ iff by simulating the quantifier prefix in b, 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 $\mathtt{simulate}$ (Algorithm 1). In Eq. (1), $\widetilde{b}$ quantifies over strategies in $\mathcal{G}$, which are infinite objects (function $S^+ \to A$). 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, 18]. 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 b, 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 b, we can update the state of the π-simulation by constructing the action assignment $\prod_{i\in Agts} a_{\vec{x}(i)}$, 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 $zip(\Pi) \in \mathcal{L}(\mathcal{A}_3)$ iff $\Pi \models_{\mathcal{G}} \psi$ (cf. Example 6). We invoke $\mathtt{simulate}(\mathcal{G}, \dot{s}, \pi_2, \vec{x}, b_2, \mathcal{A}_3)$ where $\vec{x} = (w, z, v, v)$ and $b_2 = \exists z \forall w \exists v$, and let (Q, q_0, δ, c) be the DPA equivalent to $\mathcal{A}_3$ (computed in line 2). In this case, $\mathtt{simulate}$ computes the APA $\mathcal{B} = (Q \times S, (q_0, \dot{s}), \delta', c')$ over alphabet $\{\pi_1\} \to S$ where $\delta'((q, s), \vec{t})$ is defined as*

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

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 = ♭₁ ⋯ ♭ₘ. ψ[πₖ : x⃗ₖ]ₖ₌₁ᵐ):
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, ♭ₖ, 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

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

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 Appendix D 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., [18, 31] 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

Table 1: We compare HyMASMC **and** MCMAS-SL[1G] **on the scheduler problem from [23]. 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	$|S|$	$|S_{reach}|$	$t_{\text{MCMAS-SL[1G]}}$	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**

construction to translate the LTL formula ψ to an APA over alphabet $(\{\pi_1, \ldots, \pi_m\} \to S)$, 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:

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 $Qx.Qy.Qz.Qw. \psi[\pi_1 : (x, z), \pi_2 : (y, w)]$, leads to undecidable MC (see Appendix C.3). 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 restrictions

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

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

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 [18].

7.1 Model-Checking For Strategy Logic

We compare `HyMASMC` against `MCMAS-SL[1G]` [23] on (non-hyper) SL[1G] properties (cf. Section 6.2). In Table 1, we depict the verification times for the scheduling problem from [23] (which can be expressed in SL[1G] and ATL*). As in [18], we observe that `HyMASMC` performs much faster than `MCMAS-SL[1G]`, which we largely accredit to `HyMASMC`'s efficient automata backend using `spot` [30]. 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.

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.

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. On Alternating-Time Temporal Logic, Hyperproperties, and Strategy Sharing. In *Conference on Artificial Intelligence, AAAI 2024*.

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

[20] 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*.

[21] 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*.

[22] 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).

[23] 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*.

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

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

[26] 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*.

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

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

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

[30] 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*.

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

[32] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. 2018. Synthesizing Reactive Systems from Hyperproperties. In *International Conference on Computer Aided Verification, CAV 2018*.

[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).

A ADDITIONAL MATERIAL FOR SECTION 3

In this section we give details on the semantics of APAs. To make our later proofs (which use the APA semantics) easier, we use a DAG-based semantics (opposed to the more prominently used – but equivalent – tree-based semantics). We refer the reader to [53] for more details.

DEFINITION 3. *For a set Q, we write $\mathbb{B}^+(Q)$ for the set of all positive boolean formulas over Q, i.e., all formulas generated by the grammar*

$$\theta := q \mid \theta_1 \wedge \theta_2 \mid \theta_1 \vee \theta_2$$

where $q \in Q$. Given a subset $X \subseteq Q$ and $\theta \in \mathbb{B}^+(Q)$, we write $X \models \theta$ if the assignment that maps all states in X to $\top$ and those in $Q \setminus X$ to $\bot$ satisfies Ψ. For example $\{q_0, q_1\} \models q_0 \wedge (q_1 \vee q_2)$.

Let $\mathcal{A} = (Q, q_0, \delta, c)$ be an APA. A run DAG of $\mathcal{A}$ is a pair $\mathbb{D} = (V, E)$ of nodes and edges such that $V \subseteq Q \times \mathbb{N}$, $E \subseteq \bigcup_{i \in \mathbb{N}} (Q \times \{i\}) \times (Q \times \{i+1\})$. That is, each node in V consist of a state in Q and a depth and the edges in E only reach from depth i to depth $i+1$. For every $(q, i) \in V$, we define $sucs(q, i) := \{q' \mid ((q, i), (q', i+1)) \in E\}$ as the state component of (q, i)'s successor nodes.

A run of $\mathcal{A}$ on a word $u \in \Sigma^\omega$ is a run DAG $\mathbb{D} = (V, E)$ such that $(q_0, 0) \in V$ and for every $(q, i) \in V$, $sucs(q, i) \models \delta(q, u(i))$. That is, the run DAG starts in the initial state q_0 at depth 0, and for each node in the DAG the successors of each node satisfy the transition formula given by the transition function δ.

For example, consider a node (q, i) and assume that $\delta(q, u(i)) = q_1 \vee q_2$. Then the DAG must include either q_1 or q_2 (or both) as successors in the next level. In particular, if the transition function δ uses only disjunctions (no conjunctions) the automataon is non-deterministic. In this case, each node in the DAG can have a unique successor; the DAG is a line (a infinite sequence of states). Conversely, if $\delta(q, u(i)) = q_1 \wedge q_2$ then both q_1 and q_2 need to appear in the next level, i.e., we need to construct accepting runs from both of these states.

A run DAG $\mathbb{D}$ is accepting if for every infinite path in the DAG the minimal color that occurs infinitely many times (as given by c) is even. We define $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ as all infinite words on which $\mathcal{A}$ has an accepting run DAG.

B ADDITIONAL MATERIAL FOR SECTION 5

We provide some background on the temporal logics HyperLTL, SL, HyperATL*, and HyperATL$^*_S$ and give their full semantics.

B.1 SL and HyperSL

In this subsection, we provide addition details on the relation of SL and HyperSL (cf. Section 5.1). We consider a variant of SL that allows multiple agents to share the strategy by using explicit agent binding, but disallows agent bindings under temporal operators.

REMARK 1. *The strategy logic by Mogavero et al. [45] allows arbitrary nesting of agent binding within temporal operators. Our variant is equivalent to SL[BG] [45] – a fragment that follows a strict separation between state and path formulas and thereby forbids agent binding under temporal operators. Note that SL[BG] strictly subsumes the strategy logic by Chatterjee et al. [24]. We choose to restrict to the SL[BG] fragment as it is syntactically closer to temporal logics like CTL* and HyperCTL* and, thereby, also to HyperSL.*

State and path formulas in SL are defined as follows:

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

where $a \in AP$, $x \in \mathcal{X}$, and $i \in Agts$. Importantly, we assume that every state formula occurring in a path formula is closed.

The idea of SL is to separate the quantification of a strategy and the binding of a strategy to some agent. To accomplish the latter, it features an explicit agent-binding construct $(i, x)\varphi$ which evaluates φ after binding agent i to strategy x.

Semantics. Assume $\mathcal{G} = (S, s_0, \mathbb{A}, \kappa, L)$ is a fixed CGS. Given a path $p \in S^\omega$ we define the semantics of path formulas as expected:

$p \models_\mathcal{G} a$	iff	$a \in L(p(0))$
$p \models_\mathcal{G} \varphi$	iff	$p(0), \{\}, \{\} \models_\mathcal{G} \varphi$
$p \models_\mathcal{G} \psi_1 \wedge \psi_2$	iff	$p \models_\mathcal{G} \psi_1$ and $p \models_\mathcal{G} \psi_2$
$p \models_\mathcal{G} \neg\psi$	iff	$p \not\models_\mathcal{G} \psi$
$p \models_\mathcal{G} X\psi$	iff	$p[1, \infty] \models_\mathcal{G} \psi$
$p \models_\mathcal{G} \psi_1 \cup \psi_2$	iff	$\exists j \in \mathbb{N}. p[j, \infty] \models_\mathcal{G} \psi_2$ and
		$\forall 0 \le k < j. p[k, \infty] \models_\mathcal{G} \psi_1$

In the semantics of state formulas, we keep track of a strategy for each strategy variable via a (partial) strategy assignment $\Delta : \mathcal{X} \rightharpoonup Str(\mathcal{G})$, similar to the HyperSL semantics. As SL works with explicit agent bindings, we also keep track of a strategy for each agent using a (partial) function $\Theta : Agts \rightharpoonup Str(\mathcal{G})$. We can then define:

$s, \Delta, \Theta \models_\mathcal{G} \forall x. \varphi$	iff	$\forall f \in Str(\mathcal{G}). s, \Delta[x \mapsto f], \Theta \models_\mathcal{G} \varphi$
$s, \Delta, \Theta \models_\mathcal{G} \exists x. \varphi$	iff	$\exists f \in Str(\mathcal{G}). s, \Delta[x \mapsto f], \Theta \models_\mathcal{G} \varphi$
$s, \Delta, \Theta \models_\mathcal{G} (i, x)\varphi$	iff	$s, \Delta, \Theta[i \mapsto \Delta(x)] \models_\mathcal{G} \varphi$
$s, \Delta, \Theta \models_\mathcal{G} \varphi_1 \wedge \varphi_2$	iff	$s, \Delta, \Theta \models_\mathcal{G} \varphi_1$ and $s, \Delta, \Theta \models_\mathcal{G} \varphi_2$
$s, \Delta, \Theta \models_\mathcal{G} \varphi_1 \vee \varphi_2$	iff	$s, \Delta, \Theta \models_\mathcal{G} \varphi_1$ or $s, \Delta, \Theta \models_\mathcal{G} \varphi_2$
$s, \Delta, \Theta \models_\mathcal{G} \psi$	iff	$Play_\mathcal{G}\big(s, \prod_{i \in Agts} \Theta(i)\big) \models_\mathcal{G} \psi$

Strategy quantification updates the binding in Δ (as in HyperSL), whereas strategy binding updates the assignment of agents in Θ. For each path formula we use the strategy profile Θ (mapping each agent to a strategy) to construct the path on which we evaluate ψ. We write $\mathcal{G} \models_{SL} \varphi$ if $s_0, \{\}, \{\} \models_\mathcal{G} \varphi$ in the SL semantics.

Encoding. We can easily encode SL into HyperSL. Instead of using explicit agent bindings as in SL, in HyperSL, each path is annotated with an explicit strategy profile that assigns a strategy (variable) to each agent. We assume that in the SL formula no two quantifiers quantify the same strategy variable (which we can always ensure by α-renaming). Let $\dot\pi$ denote some *fixed* path variable. In our translation, we maintain an auxiliary mapping $\vec{x} : Agts \rightharpoonup \mathcal{X}$ mapping agents to strategy variables. We then translate path formulas as follows:

$(\!(a)\!) := a_{\dot\pi}$	$(\!(\neg\psi)\!) := \neg(\!(\psi)\!)$	$(\!(\psi_1 \wedge \psi_2)\!) := (\!(\psi_1)\!) \wedge (\!(\psi_2)\!)$
$(\!(\varphi)\!) := (\!(\varphi)\!)^{\{\}}$	$(\!(X\psi)\!) := X(\!(\psi)\!)$	$(\!(\psi_1 \cup \psi_2)\!) := (\!(\psi_1)\!) \cup (\!(\psi_2)\!)$

For state formulas we define:

$$(\!|\varphi_1 \wedge \varphi_2|\!)^{\vec{x}} := (\!|\varphi_1|\!)^{\vec{x}} \wedge (\!|\varphi_2|\!)^{\vec{x}} \qquad (\!|\varphi_1 \vee \varphi_2|\!)^{\vec{x}} := (\!|\varphi_1|\!)^{\vec{x}} \vee (\!|\varphi_2|\!)^{\vec{x}}$$

$$(\!|\forall x.\,\varphi|\!)^{\vec{x}} := \forall x.\, (\!|\varphi|\!)^{\vec{x}} \qquad\qquad (\!|\exists x.\,\varphi|\!)^{\vec{x}} := \exists x.\, (\!|\varphi|\!)^{\vec{x}}$$

$$(\!|\psi|\!)^{\vec{x}} := (\!|\psi|\!)[\dot{\pi} : \vec{x}] \qquad\qquad (\!|(i,x)\varphi|\!)^{\vec{x}} := (\!|\varphi|\!)^{\vec{x}[i \mapsto x]}$$

For path formulas, we resolve all atomic propositions on the fixed path variable $\dot{\pi}$. For state formulas, we record the strategy variable played by each agent in the function $\vec{x}$, and whenever we reach a path formula, use $\vec{x}$ to construct the fixed path $\dot{\pi}$. Note that the formula resulting from the translation uses boolean combination of state formulas. As already argued in Section 4, we can bring such formulas into the HyperSL syntax by introducing multiple paths (see Example 1). We can show that our translation is correct and thus prove Lemma 1:

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. We can easily show that for any game structure $\mathcal{G}$ and SL formula φ, we have that $\mathcal{G} \models_{SL} \varphi$ if an only if $\mathcal{G} \models (\!|\varphi|\!)^{\{\}}$. □

B.2 HyperATL* and HyperSL

In this subsection, we provide addition details on the relation of HyperATL* and HyperSL (cf. Section 5.2).

Our hyper variant of strategy logic considers strategies as first-order objects that can be compared in different strategy profiles. A weaker form of strategic reasoning is offered in ATL* [2] by only reasoning about the *outcome* of a strategic interaction. The ATL* formula $\langle\!\langle A \rangle\!\rangle \psi$ expresses that the agents in A have a joint strategy to enforce that the system follows some path that satisfies ψ. HyperATL* [14, 17] is a hyper-variant of ATL that combines strategic reasoning with the ability to express hyperproperties. Similar to ATL*, HyperATL* only considers the outcomes of a strategy but binds this outcome to a path variable which allows comparison w.r.t. a hyperproperty. Formulas in HyperATL* are generated by the following grammar:

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

where $a \in AP$, $\pi \in \mathcal{V}$, and $A \subseteq Agts$.

Semantics. The semantics of HyperATL* operates on a path assignment $\Pi : \mathcal{V} \rightharpoonup S^\omega$. For path formulas, we follow a similar semantics as used in HyperSL (cf. Section 4):

$$\begin{array}{lll} \Pi \models_{\mathcal{G}} a_\pi & \text{iff} & a \in L(\Pi(\pi)(0)) \\ \Pi \models_{\mathcal{G}} \psi_1 \wedge \psi_2 & \text{iff} & \Pi \models_{\mathcal{G}} \psi_1 \text{ and } \Pi \models_{\mathcal{G}} \psi_2 \\ \Pi \models_{\mathcal{G}} \neg\psi & \text{iff} & \Pi \not\models_{\mathcal{G}} \psi \\ \Pi \models_{\mathcal{G}} \mathsf{X}\psi & \text{iff} & \Pi[1,\infty] \models_{\mathcal{G}} \psi \\ \Pi \models_{\mathcal{G}} \psi_1 \cup \psi_2 & \text{iff} & \exists j \in \mathbb{N}.\,\Pi[j,\infty] \models_{\mathcal{G}} \psi_2 \text{ and} \\ & & \forall 0 \le k < j.\,\Pi[k,\infty] \models_{\mathcal{G}} \psi_1 \end{array}$$

For state formulas, we need to consider all possible outcomes under strategies for a subset of the agents. Given $A \subseteq Agts$ and strategies

$\{f_i \mid i \in A\}$ we define $out_{\mathcal{G}}\big(s, \{f_i \mid i \in A\}\big) \subseteq S^\omega$ as

$$out_{\mathcal{G}}\big(s, \{f_i \mid i \in A\}\big) :=$$
$$\Big\{ Play_{\mathcal{G}}\big(s, \prod_{i \in Agts} f_i\big) \mid \forall i \in Agts \setminus A.\, f_i \in Str(\mathcal{G}) \Big\}.$$

That is, $out_{\mathcal{G}}\big(s, \{f_i \mid i \in A\}\big)$ contains all possible paths compatible with the strategies in $\{f_i \mid i \in A\}$. We can then evaluate a state formula in the context of a state s and path assignment Π.

$$\begin{array}{lll} s, \Pi \models_{\mathcal{G}} \psi & \text{iff} & \Pi \models_{\mathcal{G}} \psi \\ s, \Pi \models_{\mathcal{G}} \langle\!\langle A \rangle\!\rangle \pi.\,\varphi & \text{iff} & \exists\{f_i \mid i \in A\}. \\ & & \forall p \in out_{\mathcal{G}}\big(s, \{f_i \mid i \in A\}\big).\, s, \Pi[\pi \mapsto p] \models_{\mathcal{G}} \varphi \\ s, \Pi \models_{\mathcal{G}} [\![A]\!]\pi.\,\varphi & \text{iff} & \forall\{f_i \mid i \in A\}. \\ & & \exists p \in out_{\mathcal{G}}\big(s, \{f_i \mid i \in A\}\big).\, s, \Pi[\pi \mapsto p] \models_{\mathcal{G}} \varphi \end{array}$$

That is, a formula $\langle\!\langle A \rangle\!\rangle \pi.\,\varphi$ holds when there exist strategies for all agents in A such that all possible outcomes under those fixed strategies, when bound to π, satisfy φ. Likewise, $[\![A]\!]\pi.\,\varphi$ holds when all possible strategies admits some path that, when bound to π, satisfies φ. We write $\mathcal{G} \models_{\text{HyperATL}^*} \varphi$ if $s_0, \{\} \models_{\mathcal{G}} \varphi$ in the HyperATL* semantics.

Encoding. Similar to the fact that ATL* can be encoded in SL [24], we can encode HyperATL* in HyperSL. The idea is to treat the ATL-quantifier $\langle\!\langle A \rangle\!\rangle$ as a strategy quantifier that existentially quantifies over strategies for agents in A and then universally over strategies for agents outside of A. HyperATL* path formulas can be translated verbatim into HyperSL path formulas. To translate state formulas we maintain a auxiliary mapping $l : \mathcal{V} \rightharpoonup \mathcal{X}^{Agts}$ from path variables to strategy profiles:

$$(\!|\psi|\!)^l := \psi[\pi_1 : l(\pi_1), \ldots, \pi_m : l(\pi_m)]$$

$$(\!|\langle\!\langle A \rangle\!\rangle \pi.\varphi|\!)^l := \exists_{i \in A} x_i.\, \bigvee_{i \in Agts \setminus A} x_i.\, (\!|\varphi|\!)^{l[\pi \mapsto \prod_{i \in Agts} x_i]}$$

$$(\!|[\![A]\!]\pi.\varphi|\!)^l := \bigvee_{i \in A} x_i.\, \exists_{i \in Agts \setminus A} x_i.\, (\!|\varphi|\!)^{l[\pi \mapsto \prod_{i \in Agts} x_i]}$$

In the first case, we assume that $\pi_1, \ldots, \pi_m$ are the path variables used in ψ. In the second and third case, we assume that $x_1, \ldots, x_n$ are fresh strategy variables for each agent. Intuitively, we replace each $\langle\!\langle A \rangle\!\rangle \pi$ quantifier with existential quantification over fresh strategies for A, followed by universal quantification over strategies for $Agts \setminus A$. In the auxiliary mapping l, we record which strategy profile we later want to use to construct π. For the path formula ψ we then reconstruct all paths used in the formula using the strategy profiles recorded in l. We can use our translation to prove Lemma 2:

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. An easy induction shows that for any game structure $\mathcal{G}$ and HyperATL* formula φ it holds that $\mathcal{G} \models_{\text{HyperATL}^*} \varphi$ if and only if $\mathcal{G} \models (\!|\varphi|\!)^{\{\}}$. □

HyperATL$_S^*$. As our translation $(\!|\varphi|\!)^{\{\}}$ explicitly quantifies over strategies for all agents, we can easily enforce that two agents *share* a strategy by simply using the same strategy (variable) for both.

Using a slight modification of the previous translation, we can thus handle the sharing constraints from HyperATL$^*_S$ [18]:

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 $\mathcal{G} \models \varphi'$.*

B.3 SL$_{ii}$ and HyperSL

In this subsection, we provide addition details on the relation of SL$_{ii}$ and HyperSL (cf. Section 5.3).

SL$_{ii}$ [12] extends SL (cf. Section 5.1) by allowing strategies that only observe parts of the system. The formal model of SL$_{ii}$ are game structures that are endowed with an observations. Let $\mathcal{G} = (S, s_0, \mathbb{A}, \kappa, L)$ be a fixed game structure and let Obs be a fixed finite set of so-called *observations*. An *observation family* $\{\sim_o\}_{o \in Obs}$ associates an equivalence relation $\sim_o \subseteq S \times S$ with each $o \in Obs$.

For a strategy with observation o, two states $s \sim_o s'$ appear identical. This naturally extends to finite plays: Two finite plays $p, p' \in S^+$ are o-indistinguishable, written $p \sim_o p'$, if $|p| = |p'|$ and for each $0 \leq i < |p|$, $p(i) \sim_o p'(i)$. An o-*strategy* is a function $f : S^+ \to \mathbb{A}$ that cannot distinguish between o-indistinguishable plays, i.e., for all $p, p' \in S^+$ with $p \sim_o p'$ we have $f(p) = f(p')$. We denote with $Str(\mathcal{G}, o)$ the set of all o-strategies in $\mathcal{G}$. We consider SL$_{ii}$ formulas that are generated by the following grammar:

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

where $a \in AP$, $x \in \mathcal{X}$, $i \in Agts$, and $o \in Obs$ is an observation. Compared to SL, strategy quantification in SL$_{ii}$ does not range over arbitrary strategies but over strategies that respect a given observation.

Semantics. The semantics of SL (cf. Appendix B.1) only needs to be changed in the case of strategy quantification:

$$s, \Delta, \Theta \models_{\mathcal{G}} \forall x^o. \varphi \quad \text{iff} \quad \forall f \in Str(\mathcal{G}, o). s, \Delta[x \mapsto f], \Theta \models_{\mathcal{G}} \varphi$$
$$s, \Delta, \Theta \models_{\mathcal{G}} \exists x^o. \varphi \quad \text{iff} \quad \exists f \in Str(\mathcal{G}, o). s, \Delta[x \mapsto f], \Theta \models_{\mathcal{G}} \varphi$$

where we restrict quantification to o-strategies. Given a game structure $\mathcal{G}$, a family $\{\sim_o\}_{o \in Obs}$, and an SL$_{ii}$ formula φ we write $(\mathcal{G}, \{\sim_o\}_{o \in Obs}) \models_{SL_{ii}} \varphi$ if $s_0, \{\}, \{\} \models_{\mathcal{G}} \varphi$ in the SL$_{ii}$ semantics. See [12, 13] for concrete examples of SL$_{ii}$.

Injective Labeling and Action Recording. To prove Theorem 1, we need to translate SL$_{ii}$ MC instances into equisatisfiable HyperSL instances. In order to translate the model-checking instances, we first modify the underlying game structure. The reason for this is simple: Strategies are defined as functions $S^+ \to \mathbb{A}$ and $\sim_o$ is defined as a direct relation on states, i.e., both are defined directly on *components of the game structure*. In contrast, *within* our logic, we only observe the evaluation of the atomic propositions. In a first step, we thus modify the game structure provided us with sufficient information within its atomic propositions:

Definition 4. *A game structure $\mathcal{G} = (S, s_0, \mathbb{A}, \kappa, L)$ is injectively labeled (IL), if $L : S \to 2^{AP}$ is injective, i.e., two labels are equal iff the state is equal. A game structure is action recording (AR) if for each agent $i \in Agts$ and every action $a \in \mathbb{A}$, there exists an atomic proposition $\langle i, a \rangle \in AP$ that holds in a state exactly when i played*

action a in the last step. That is, for all $s \in S$ and all action profile $\prod_{i \in Agts} a_i$, we have

$$L\big(\kappa(s, \prod_{i \in Agts} a_i)\big) \cap \{\langle i, a \rangle \mid i \in Agts, a \in \mathbb{A}\} = \bigcup_{i \in Agts} \{\langle i, a_i \rangle\}.$$

We can always ensure that $\mathcal{G}$ is injectively labeled by adding at most $\lceil \log |S| \rceil$ fresh atomic propositions. The evaluation of all formulas (which do not refer to these fresh propositions) remains unchanged when adding such fresh propositions. Similarly, we can always ensure that a structure records actions. This increases the number of states by a factor of $|\mathbb{A}|^{|Agts|}$ but does not increase the branching in the system, nor does it change the semantics of SL$_{ii}$ if we extend $\sim_o$ to the new states in the obvious way:

Lemma 7. *Given an SL$_{ii}$ MC instance $(\mathcal{G}, \{\sim_o\}_{o \in Obs}, \varphi)$ there exists an effectively computable SL$_{ii}$ instance $(\mathcal{G}', \{\sim'_o\}_{o \in Obs}, \varphi')$ where (1) $\mathcal{G}'$ is IL and AR, and (2) $(\mathcal{G}, \{\sim_o\}_{o \in Obs}) \models_{SL_{ii}} \varphi$ iff $(\mathcal{G}', \{\sim'_o\}_{o \in Obs}) \models_{SL_{ii}} \varphi'$.*

PROOF. Assume $\mathcal{G} = (S, s_0, \mathbb{A}, \kappa, L)$. Define $AP' := AP \uplus \{\langle i, a \rangle \mid i \in Agts, a \in \mathbb{A}\}$. We then define

$$\mathcal{G}' = (S \times (Agts \to \mathbb{A}), (s_0, \prod_{i \in Agts} a), \mathbb{A}, \kappa', L')$$

where $a \in \mathbb{A}$ is some arbitrary action (in the initial state, we do not need to track the last played action). For an action profile $\prod_{i \in Agts} a_i$, we define κ' and l' by

$$\kappa'\big((s, \_), \prod_{i \in Agts} a_i\big) := (\kappa(s, \prod_{i \in Agts} a_i), \prod_{i \in Agts} a_i)$$
$$L'(s, \prod_{i \in Agts} a_i) := L(s) \uplus \{\langle i, a_i \rangle \mid i \in Agts\}.$$

The idea behind $\mathcal{G}'$ is that we record the action profile that was last used in the second component of each state. In each transition, we ignore the action profile in the current step, and record the new action profile n the second component. In the labeling function, we can then use the action profile $\prod_{i \in Agts} a_i$ in the second component to set the APs $\langle i, a_i \rangle$ for all $i \in Agts$. We define $\{\sim'_o\}_{o \in Obs}$ by

$$\sim'_o := \Big\{ \big((s, \_), (s', \_)\big) \mid s \sim_o s' \Big\}.$$

That is, for any observation cannot distinguish states based on the second position. In particular note that $(s, \_)$ and $(s, \_)$ are always indistinguishable, i.e., all states we expanded that we added are indistinguishable under the new observation.

It is easy to see that $\mathcal{G}'$ is AR and that $(\mathcal{G}, \{\sim_o\}_{o \in Obs}) \models_{SL_{ii}} \varphi$ iff $(\mathcal{G}', \{\sim'_o\}_{o \in Obs}) \models_{SL_{ii}} \varphi$. Note that we did not change the formula. In a second step, we can ensure that $\mathcal{G}'$ is also IL by simply adding sufficiently many new propositions. As those new propositions are never used in φ the semantics is unchanged. □

Identifying Indistinguishable States. In the next step, we construct a formula that identifies pairs of states that are indistinguishable according to $\sim_o$. For $o \in Obs$, we define formula $ind_o(\pi_1, \pi_2)$ as follows:

$$\bigvee_{(s,s') \in \sim_o} \Big(\bigwedge_{a \in L(s)} a_\pi \wedge \bigwedge_{a \in AP \backslash L(s)} \neg a_\pi \wedge \bigwedge_{a \in L(s')} a_{\pi'} \wedge \bigwedge_{a \in AP \backslash L(s')} \neg a_{\pi'} \Big)$$

It is easy to see that on any injectively labeled game structure $ind_o(\pi_1, \pi_2)$ holds if the two paths bound to π_1, π_2 are $\sim_o$-related in their first state.

Enforcing Partial Information. Given an observation $o \in Obs$, and strategy variable x, we define a formula $indStrat_o(x)$ that holds on a strategy iff this strategy is an o-strategy (in all reachable situations and for all agents) as follows:

$$indStrat_o(x) := \forall y_1, \ldots, y_n, y_1', \ldots, y_n'.$$

$$\bigwedge_{i=1}^{n} \psi_o^i \begin{bmatrix} \pi_1 : (y_1, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_n) \\ \pi_2 : (y_1', \ldots, y_{i-1}', x, y_{i+1}', \ldots, y_n') \end{bmatrix}$$

where

$$\psi_o^i := \Big(\mathsf{X} \Big(\bigwedge_{a \in \mathbb{A}} \langle i, a \rangle_{\pi_1} \leftrightarrow \langle i, a \rangle_{\pi_2} \Big) \Big) \mathsf{W} \Big(\neg ind_o(\pi_1, \pi_2) \Big).$$

The path formula ψ_o^i compares two paths π_1, π_2 and states that as long as a prefix on those to paths is o-indistinguishable (i.e., $ind_o(\pi_1, \pi_2)$ holds in each step), the action selected by agent i is the same on both paths (using the fact that the structure records actions). As we do not know which agents might end up playing strategy x we assert that x behaves as a o-strategy for all agents. For each $i \in Agts$ we thus compare two paths where i plays x, but all other agents play some arbitrary strategy, and assert that ψ_o^i holds for those two paths. Strategy x must thus respond to two o-indistinguishable prefixes with the same action in all reachable situations for all agents.

The Translation. Using $indStrat_o(x)$ as a building block, we can modify the translation of SL into HyperSL from Appendix B.1, and instead translate the much stronger SL_{ii}.

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

PROOF. In a first step, we use Lemma 7 to ensure that $\mathcal{G}$ is IL and AR. We can then translate φ using a similar translation to the one used in Appendix B.1. The only cases that require changening are the translation of quantification:

$$(\!(\forall x^o.\varphi)\!)^{\vec{x}} := \forall x.\, indStrat_o(x) \rightarrow (\!(\varphi)\!)^{\vec{x}}$$

$$(\!(\exists x^o.\varphi)\!)^{\vec{x}} := \exists x.\, indStrat_o(x) \wedge (\!(\varphi)\!)^{\vec{x}}$$

Note that the resulting formula uses implications between state formulas which is not supported by the HyperSL syntax. It is, however, easy to see that we can push the boolean operations into the path formula as observed in Example 1.

We claim that for any SL_{ii} MC instance $(\mathcal{G}, \{\sim_o\}_{o \in Obs}, \varphi)$ where $\mathcal{G}$ is IL and AR, we have that $(\mathcal{G}, \{\sim_o\}_{o \in Obs}) \models_{SL_{ii}} \varphi$ iff $\mathcal{G} \models (\!(\varphi)\!)^{\{\}}$.

To prove the above we need to argue that $indStrat_o(x)$ really expresses that x is a o-strategy. It is easy to see that for any strategy $f \in Str(\mathcal{G})$ we have $s, [x \mapsto f] \models indStrat_o$ if and only if f is a o-strategy in all reachable situations from s. That is, f does not necessarily behave as an o-strategy in all situations, but at least in those situations that are actually reachable under f. As any strategy will only ever be queried on plays that are compatible with the strategy itself, this suffices to encode the SL_{ii} semantics. □

C ADDITIONAL MATERIAL FOR SECTION 6

In this section we prove the correctness of our HyperSL[SPE] model-checking algorithm (Algorithm 2). Our algorithm hinges on `simulate` procedure (Algorithm 1) and the resulting properties (Proposition 2). We dedicate the entire Appendix D to a proof of Proposition 2, and here focus on the correctness (and complexity) of Algorithm 2.

C.1 Correctness Proof of Algorithm 2

As already argued in the main part of the paper, our correctness proof relies on an inductive argument that establishes that we compute $(\mathcal{G}, \dot{s}, k)$-summaries for each $1 \le k \le m + 1$.

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

PROOF. We show the statement by induction on $1 \le k \le m + 1$ (from $k = m + 1$ to $k = 1$). For the base case ($k = m + 1$), we observe that the APA $\mathcal{A}_{m+1}$ (computed in line 4) is a $(\mathcal{G}, \dot{s}, m + 1)$-summary.

For the induction step we can assume – by induction hypothesis – that prior to line 6, $\mathcal{A}_{k+1}$ is a $(\mathcal{G}, \dot{s}, k+1)$-summary. Recall that $\mathcal{A}_{k+1}$ is an APA over $(\{\pi_1, \ldots \pi_k\} \rightarrow S)$ and $\mathcal{A}_k$ over $(\{\pi_1, \ldots \pi_{k-1}\} \rightarrow S)$. We claim that $\mathcal{A}_k$ is a $(\mathcal{G}, \dot{s}, k)$-summary. To show this, take any $\Pi : \{\pi_1, \ldots \pi_{k-1}\} \rightarrow S^\omega$ and we need to show that (cf. Definition 2 of $(\mathcal{G}, \dot{s}, k)$-summary) $zip(\Pi) \in \mathcal{L}(\mathcal{A}_k)$ if and only if

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

By adding parenthesis, the latter holds if and only if

$$\widetilde{\flat_k}.\Big(\widetilde{\flat_{k+1}} \cdots \widetilde{\flat_m}. \Pi \Big[\pi_j \mapsto Play_{\mathcal{G}} \big(\dot{s}, \prod_{i \in Agts} f_{\vec{x}_j(i)} \big) \Big]_{j=k}^{m} \models_{\mathcal{G}} \psi \Big)$$

which holds if and only if

$$\widetilde{\flat_k}.\Big(\widetilde{\flat_{k+1}} \cdots \widetilde{\flat_m}. \Pi' \Big[\pi_j \mapsto Play_{\mathcal{G}} \big(\dot{s}, \prod_{i \in Agts} f_{\vec{x}_j(i)} \big) \Big]_{j=k+1}^{m} \models_{\mathcal{G}} \psi \Big)$$

where $\Pi' = \Pi[\pi_k \mapsto Play_{\mathcal{G}}(\dot{s}, \prod_{i \in Agts} f_{\vec{x}_k(i)})]$. By the assumption that $\mathcal{A}_{k+1}$ is a $(\mathcal{G}, \dot{s}, k + 1)$-summary we can replace the inner part and get that the above is equivalent to

$$\widetilde{\flat_k}.zip(\Pi') \in \mathcal{L}(\mathcal{A}_{k+1}).$$

After unfolding the definition of Π', this becomes

$$\widetilde{\flat_k}.\, zip\Big(\Pi \big[\pi_k \mapsto Play_{\mathcal{G}} \big(\dot{s}, \prod_{i \in Agts} f_{\vec{x}_k(i)} \big) \big] \Big) \in \mathcal{L}(\mathcal{A}_{k+1}).$$

Now recall that we defined $\mathcal{A}_k = \mathtt{simulate}(\mathcal{G}, \dot{s}, \pi_k, \vec{x}_k, \flat_k, \mathcal{A}_{k+1})$. By Proposition 2 we now have that the above holds iff

$$zip(\Pi) \in \mathcal{L}(\mathcal{A}_k).$$

as required. □

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

PROOF. Note that the alphabet of $\mathcal{A}$ is the singleton set $(\emptyset \rightarrow S)$. We thus get that $\mathcal{L}(\mathcal{A}) \ne \emptyset$ iff $zip(\{\}) \in \mathcal{L}(\mathcal{A})$ where $\{\}$ is the unique path assignment $\emptyset \rightarrow S^\omega$ so $zip(\{\})$ is the unique word over $(\emptyset \rightarrow S)$. Now by Definition 2 we have that $zip(\{\}) \in \mathcal{L}(\mathcal{A})$ iff

$$\widetilde{\flat_1} \cdots \widetilde{\flat_m}. \{\} \Big[\pi_j \mapsto Play_{\mathcal{G}} \big(\dot{s}, \prod_{i \in Agts} f_{\vec{x}_j(i)} \big) \Big]_{j=1}^{m} \models_{\mathcal{G}} \psi$$

where we add all paths to the empty path assignment $\{\}$. As we add to the empty path assignment, the above is thus equivalent to

$$\widetilde{b_1} \cdots \widetilde{b_m}. \left[\pi_j \mapsto Play_{\mathcal{G}}\left(\dot{s}, \prod_{i \in Agts} f_{\vec{x}_j(i)}\right)\right]_{j=1}^m \models_{\mathcal{G}} \psi$$

which exactly expresses $\dot{s}, \{\} \models_{\mathcal{G}} \varphi$ in the HyperSL semantics. □

C.2 Model-Checking Complexity

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

PROOF. Each time we invoke simulate (Algorithm 1) the automaton size increases by two exponents. A formula with block-rank m requires m applications of simulate (cf. Algorithm 2), so the final automaton $\mathcal{A}_1$ has size that is $2m$-times exponential in the size of ψ and $\mathcal{G}$. Automaton $\mathcal{A}_1$ operates on a singleton alphabet ($\emptyset \to S$), so we can decide its emptiness in polynomial time. Model checking is thus in $2m$-EXPTIME in the size of ψ and $\mathcal{G}$. □

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

PROOF. For HyperATL* it is known that checking a formula with m quantifiers is $(2m-1)$-EXPSPACE-hard (in the size of the formula) [17, Thm. 7.1 and 7.2]. As the translation of a HyperATL* formula with m qunatifiers into a HyperSL[SPE] formula is linear (cf. Lemma 2 and Appendix B.2) and yields a formula with block-rank m, the lower bound follows. □

C.3 Beyond HyperSL[SPE]

As we argued in Section 6.7, any HyperSL formula where the prefix cannot be grouped into blocks as in Definition 1, MC becomes in general undecidable.

LEMMA 8. *Model checking for a HyperSL formula of the form*

$$\exists x. \exists y. \forall z. \forall w. \psi \begin{bmatrix} \pi_1 : (x, z) \\ \pi_2 : (y, w) \end{bmatrix}$$

is, in general, undecidable.

PROOF. We encode the HyperLTL realizability problem of a $\forall^2$ formula, which is known to be undecidable [32]. Given a HyperLTL formula $\varphi = \forall \pi_1. \forall \pi_2. \psi$ over $I \uplus O$ we define

$$\psi' := \psi \wedge \left(\left(X\left(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2}\right)\right) W \left(\bigvee_{a \in I} a_{\pi_1} \leftrightarrow a_{\pi_2}\right)\right)$$

That is, the two paths π_1, π_2 should satisfy ψ and, in addition, when given the same sequence of inputs, the output should be the same. We claim that $\varphi = \forall \pi_1. \forall \pi_2. \psi$ is realizable if and only if

$$\mathcal{G}_{(I,O)} \models \exists x. \exists y. \forall z. \forall w. \psi' \begin{bmatrix} \pi_1 : (x, z) \\ \pi_2 : (y, w) \end{bmatrix}$$

where $\mathcal{G}_{(I,O)}$ is the CGS in which agent 1 can set the inputs I and agent 2 can set the outputs O (see, e.g., [2]). The intuition is that the additional conjunct requires that the two strategies bound to x and y denote the *same* strategy. The above formula thus states that there exists some strategy that controls the outputs such that all pairs of traces under that strategy satisfy ψ. Deciding the existence of such a strategy is undecidable [32], so we get the desired result. □

D PROOF OF PROPOSITION 2

In this section, we prove Proposition 2:

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

$$\widetilde{b}. zip\left(\Pi\left[\pi \mapsto Play_{\mathcal{G}}\left(\dot{s}, \prod_{i \in Agts} f_{\vec{x}(i)}\right)\right]\right) \in \mathcal{L}(\mathcal{A}). \quad (1)$$

For any $m \in \mathbb{N}$, we define $[m] := \{1, \ldots, m\}$. To make the proof of its correctness easier, we assume that

$$b = \forall 1. \exists 2. \forall 3 \ldots \exists 2m. \quad (2)$$

That is, we assume that the strategy variables are number in $[2m]$. And, moreover, we assume the quantifier block alternates strictly in every step: odd strategy variables (numbers) are universally quantified and even variables are existentially quantified. Note that this assumption is w.l.o.g., we can always add quantification over additional strategy variables that wil never be used for the construction of π. Having a fixed alternation makes formal reasoning and notation easier.

The Candidate. Let $\mathcal{A}_{det} = (Q, q_0, \delta, c)$ be the DPA constructed from $\mathcal{A}$ in line 2 of Algorithm 1 (using Proposition 1). Following the construction in Algorithm 1, we get that $\mathcal{B}$ (the result of $\mathrm{simulate}(\mathcal{G}, \dot{s}, \pi, \vec{x}, b, \mathcal{A}))$ satisfies

$$\mathcal{B} = \left(Q \times S, (q_0, \dot{s}), \delta', c'\right),$$

where $c'(q, s) := c(q)$, and for $\vec{t} \in S^V$, $\delta'((q, s), \vec{t})$ is defined as as

$$\bigwedge_{a_1 \in \mathbb{A}} \bigvee_{a_2 \in \mathbb{A}} \cdots \bigvee_{a_{2m} \in \mathbb{A}} \left(\delta(q, \vec{t}[\pi \mapsto s]), \kappa\left(s, \prod_{i \in Agts} a_{\vec{x}(i)}\right)\right),$$

With this construction fixed, it remains to argue that it accepts the desired language. Expressed as a lemma:

LEMMA 9. *For any path assignment $\Pi : V \to S^\omega$ we have $zip(\Pi) \in \mathcal{L}(\mathcal{B})$ if and only if*

$$\widetilde{b}. zip\left(\Pi\left[\pi \mapsto Play_{\mathcal{G}}\left(\dot{s}, \prod_{i \in Agts} f_{\vec{x}(i)}\right)\right]\right) \in \mathcal{L}(\mathcal{A}). \quad (3)$$

D.1 Concurrent Parity Games

The main prove idea in showing that $\mathcal{B}$ accepts the language desired by Proposition 2, goes via the determinacy of concurrent parity games.

DEFINITION 5 (CONCURRENT PARITY GAME). *A concurrent parity game (CPG) is a a tuple $\mathcal{P} = (V, v_0, \mathbb{A}, m, \mu, c)$ where V is a (possibly infinite) set of vertices, $v_0 \in V$ is an initial vertex, $\mathbb{A}$ is a finite set of actions, $m \in \mathbb{N}$ gives the number of alternations (so $2m$ is the number of player), $\mu : V \times ([2m] \to \mathbb{A}) \to V$ is a transition function, and $c : V \to C$ is a coloring for some finite $C \subseteq \mathbb{N}$.*

We refer to the protagonist in CPGs as players to distinguish them from the agents in a game structure. Similar to the simplyfying assumption we made of b, we will later quantify universally over strategies for odd players and existentially for even players. A strategy in $\mathcal{P}$ is a function $f : V^+ \to \mathbb{A}$ and we write $Str(\mathcal{P})$

for the set of all strategies in $\mathcal{P}$. When given a strategy profile $\vec{f} : [2m] \rightarrow Str(\mathcal{P})$ assigning a strategy for each player, and a vertex $v \in V$, we define $Play_{\mathcal{P}}(v, \vec{f}) \in V^{\omega}$ as the unique play $p \in V^{\omega}$ such that $p(0) = v$, and for every $j \in \mathbb{N}$, $p(j + 1) = \mu(p(j), \prod_{l \in [2m]} \vec{f}(l)(p[0, j]))$ (similar to the definition in CGSs, cf. Section 3). We say an infinite play $p \in V^{\omega}$ is *even* if the minimal color that occurs infinity many times (as given by c) is even. In this case we write $even(p)$.

DEFINITION 6. *The CPG $\mathcal{P}$ is* won by the existential team *if*

$$\forall f_1 \in Str(\mathcal{P}).\exists f_2 \in Str(\mathcal{P}).\forall f_3 \in Str(\mathcal{P}) \dots \exists f_{2m} \in Str(\mathcal{P}).$$

$$even\Big(Play_{\mathcal{P}}(v_0, \prod_{l \in [2m]} f_l)\Big).$$

That is we quantify over strategies in $\mathcal{P}$ for all player $l \in [2m]$; universally for odd player and existentially for even player in alternating fashion. The game is won (by the existential team) if the existential quantifier can ensure that the resulting play is even.

D.2 Construction of $\mathcal{P}_{\Pi}$

Assume we are given a fixed strategy assignment $\Pi : V \rightarrow S^{\omega}$. We design an infinite-state CPG $\mathcal{P}_{\Pi}$ as

$$\mathcal{P}_{\Pi} = (Q \times S \times \mathbb{N}, (q_0, \acute{s}, 0), \mathbb{A}, m, \mu, c')$$

where Q is the set of states in $\mathcal{A}_{det}$, $c'(q, s, N) := c(q)$ and for each action profile $\vec{a} : [2m] \rightarrow \mathbb{A}$ we define $\mu((q, s, N), \vec{a})$ as

$$\Big(\delta(q, zip(\Pi)(N)[\pi \mapsto s]), \kappa(s, \prod_{i \in Agts} \vec{a}(\vec{x}(i))), N + 1\Big)$$

In the following we abbreviate $V := Q \times S \times \mathbb{N}$ for the vertices in $\mathcal{P}_{\Pi}$, and define $v_0 := (q_0, \acute{s}, 0)$ as the initial vertex of $\mathcal{P}_{\Pi}$.

Let us discuss the construction of $\mathcal{P}_{\Pi}$. Actions in $\mathcal{P}_{\Pi}$ are the same as in $\mathcal{G}$ (i.e., $\mathbb{A}$). For each quantified strategy variable $1, \dots, 2m$ (cf. Equation (2)), we have a corresponding player in $\mathcal{P}_{\Pi}$ (i.e., there are m alternations, so the set of players is $[2m]$). We operate on vertices (q, s, N), where $q \in Q$ and $s \in S$ are similar to the construction of $\mathcal{B}$. In addition we track the current step N. To update q we invoke the transition function δ of $\mathcal{A}_{det}$ on the current automaton state and letter $zip(\Pi)(N)[\pi \mapsto s]$. Note that this corresponds to the input of $\mathcal{B}$: $\mathcal{B}$ reads the zipping of a strategy assignment Π as an input and thus the Nth letter of the input is $zip(\Pi)(N)$. That is, we "hardcode" Π into the game. For this, we maintain the current step via counter N and "pretend" the input in the Nth step was $zip(\Pi)(N)$. Similar to $\mathcal{B}$, we update the automaton state by passing $zip(\Pi)(N)[\pi \mapsto s]$ to $\mathcal{A}_{det}$'s transition function. To update the state of the simulation of $\mathcal{G}$ we proceed as in $\mathcal{B}$, i.e., given a function $\vec{a} : [2m] \rightarrow \mathbb{A}$ that fixes actions for all strategy variables (or, equivalently, players in $\mathcal{P}_{\Pi}$), each agent $i \in Agts$ will simply play the action assigned to strategy variable $\vec{x}(i)$, i.e., $\vec{a}(\vec{x}(i))$.

The idea of $\mathcal{P}_{\Pi}$ is to serve as intermediate representation between the target language (Eq. (3)) and the definition of $\mathcal{B}$. In the semantics of CPGs the quantification over strategies for the players occurs "outside", i.e., strategies are fixed globally (cf. Definition 6). As players in $\mathcal{P}_{\Pi}$ correspond exactly to the strategy variables used in $\flat$ ($[2m]$), this "outer" quantification mimics the quantification found in Eq. (3). On the other hand, the updates of automaton and system state in $\mathcal{P}_{\Pi}$ are similar to the updates performed in $\mathcal{B}$.

D.3 The First Implication

As a first step, we will show that $\mathcal{P}_{\Pi}$ is won by the existential player if and only Π satisfies Eq. (3). This step is easy: The quantification in $\mathcal{P}_{\Pi}$ and Eq. (3) is very similar (i.e., occurs "outside" the game). We can, therefore, transfer strategies between $\mathcal{G}$ and $\mathcal{P}_{\Pi}$ as follows:

LEMMA 10. *The following holds:*

- *For any $\Delta : [2m] \rightarrow Str(\mathcal{P}_{\Pi})$ there exists a $\widetilde{\Delta} : [2m] \rightarrow Str(\mathcal{G})$ such that*

$$zip\Big(\Pi[\pi \mapsto Play_{\mathcal{G}}(\acute{s}, \prod_{i \in Agts} \widetilde{\Delta}(\vec{x}(i)))]\Big) \in \mathcal{L}(\mathcal{A}).$$

if and only if $even(Play_{\mathcal{P}_{\Pi}}(v_0, \Delta))$.

- *For any $\Delta : [2m] \rightarrow Str(\mathcal{G})$ there exists a $\widetilde{\Delta} : [2m] \rightarrow Str(\mathcal{P}_{\Pi})$ such that*

$$zip\Big(\Pi[\pi \mapsto Play_{\mathcal{G}}(\acute{s}, \prod_{i \in Agts} \Delta(\vec{x}(i)))]\Big) \in \mathcal{L}(\mathcal{A}).$$

if and only if $even(Play_{\mathcal{P}_{\Pi}}(v_0, \widetilde{\Delta}))$.

PROOF. We show both claims separately.

- We translate strategies in $\mathcal{P}_{\Pi}$ to strategies in $\mathcal{G}$. Let $f : V^+ \rightarrow \mathbb{A}$ be some strategy in $\mathcal{P}_{\Pi}$. We define $\widetilde{f} : S^+ \rightarrow \mathbb{A}$ as follows: Let $u \in S^+$ be given. Define $q \in Q^+$ as follows: We define $q(0) = q_0$ (the initial state of $\mathcal{A}_{det}$). For $0 \leq i < |u|$ we then define $q(i + 1) := \delta(q(i), zip(\Pi)(i)[\pi \mapsto u(i)])$. The sequence q thus gives the unique state sequence in $\mathcal{A}_{det}$ when reading the first $|u|$ states of $zip(\Pi)$ extended with u. Now consider the finite play in $\mathcal{P}_{\Pi}$

$$\tau = \big(q(0), u(0), 0\big) \cdots \big(q(|u| - 1), u(|u| - 1), |u| - 1\big).$$

The intuition is that τ corresponds to the unique path in $\mathcal{P}_{\Pi}$ that, when projected onto the system state, gives u. We define $\widetilde{f}(u) := f(\tau)$.
For any strategy assignment $\Delta : [2m] \rightarrow Str(\mathcal{P}_{\Pi})$ in $\mathcal{P}_{\Pi}$ we define $\widetilde{\Delta} : [2m] \rightarrow Str(\mathcal{G})$ as the assignment obtained by applying $\widetilde{\cdot}$ point wise.
It is now easy to see that $Play_{\mathcal{G}}(\acute{s}, \prod_{i \in Agts} \widetilde{\Delta}(\vec{x}(i))) \in S^{\omega}$ equals $Play_{\mathcal{P}_{\Pi}}(v_0, \Delta)$ (when projecting vertices in $Q \times S \times \mathbb{N}$ on S). By construction, the automaton component in each vertex of $\mathcal{P}_{\Pi}$ simply simulates $\mathcal{A}_{det}$ on the generated sequence of system state and thus accepts play iff the corresponding automaton sequence is accepting. We thus get that the

$$zip\Big(\Pi[\pi \mapsto Play_{\mathcal{G}}(\acute{s}, \prod_{i \in Agts} \widetilde{\Delta}(\vec{x}(i)))]\Big) \in \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_{det})$$

if and only if $even(Play_{\mathcal{P}_{\Pi}}(v_0, \Delta))$, as required.

- Let $f : S^+ \rightarrow \mathbb{A}$ be a strategy in $\mathcal{G}$. We define $\widetilde{f} : V^+ \rightarrow \mathbb{A}$ in $\mathcal{P}_{\Pi}$ as follows: For any $u \in V^+$ let $s \in S^+$ be the projection on the system state in u. We define $\widetilde{f}(u) := f(s)$, i.e., only query f on the sequences of system states. For any strategy assignment $\Delta : [2m] \rightarrow Str(\mathcal{G})$ in $\mathcal{P}_{\Pi}$ we define $\widetilde{\Delta} : [2m] \rightarrow Str(\mathcal{P}_{\Pi})$ as the assignment obtained by applying $\widetilde{\cdot}$ point wise. As in the first case, it is easy to see that $Play_{\mathcal{G}}(\acute{s}, \prod_{i \in Agts} \Delta(\vec{x}(i))) \in S^{\omega}$ is equal to

$Play_{\mathcal{P}_\Pi}(v_0, \widetilde{\Delta})$ (when projecting vertices in $Q \times S \times \mathbb{N}$ on S). As $\mathcal{P}_\Pi$ simulates $\mathcal{A}_{det}$ on the sequence of system states we thus get

$$zip\Big(\Pi\big[\pi \mapsto Play_{\mathcal{G}}\big(\dot{s}, \prod_{i \in Agts} \Delta(\vec{x}(i))\big)\big]\Big) \in \mathcal{L}(\mathcal{A}).$$

if and only if $even(Play_{\mathcal{P}_\Pi}(v_0, \widetilde{\Delta}))$, as required. □

PROPOSITION 3. $\mathcal{P}_\Pi$ is won by the existential team if and only if

$$\widetilde{b}. \, zip\Big(\Pi\big[\pi \mapsto Play_{\mathcal{G}}\big(\dot{s}, \prod_{i \in Agts} f_{\vec{x}(i)}\big)\big]\Big) \in \mathcal{L}(\mathcal{A}). \quad (4)$$

PROOF. In $\mathcal{P}_\Pi$ we quantify over strategies in $\mathcal{P}_\Pi$ for each player in $[2m]$ (cf. Definition 6). Conversely, Eq. (4) uses the same quantitation order (in the prefix $\widetilde{b}$) but quantifies over strategies in $\mathcal{G}$ via strategy variables $1, \ldots, 2m$ (Eq. 2). Using the translation in Lemma 10 we can translate any strategy assignment in $\mathcal{P}_\Pi$ to an equivalent one in $\mathcal{G}$, and vice versa. As the type (i.e., universal or existential) and order of quantification is $\mathcal{P}_\Pi$ and Eq. (4) is the same (cf. Eq. 2) and we can translate strategies between $\mathcal{G}$ and $\mathcal{P}_\Pi$, the result follows. □

D.4 Positional Determinacy

The more challenging direction is to show that $\mathcal{P}_\Pi$ is won by the existential team iff $\mathcal{B}$ accepts $zip(\Pi)$. The key challenge is that the way strategies are quantified is fundamentally different: In $\mathcal{P}_\Pi$ we quantify over full strategies in advance (i.e., "outside") and in $\mathcal{B}$ we (conjunctively or disjunctively) pick actions in each step of the automaton (i.e., "inside"). The key ingredient we use is the *positional determinacy* of CPGs. Intuitively, a positional strategy is one that decides on an action based solely on the current vertex of the game.

DEFINITION 7. A positional strategy in $\mathcal{P} = (V, v_0, \mathbb{A}, m, \mu, c)$ is a function $f : V \to \mathbb{A}$. We write $PosStr(\mathcal{P})$ for the set of all positional strategies in $\mathcal{P}$.

Positional determinacy in the context of (classical) turn-based 2-player parity games means that players can pick an action based solely on the current vertex of the game. In contrast, the quantification over strategies in CPGs can have multiple alternations, the strategy for each player thus depends on the current vertex of the game and the action selected by all strategies quantified before it. We will represent the existentially quantified strategies using *Skolem functions* (known, e.g., from first-order logic and sometimes called *dependence map* [45]) that get the actions selected by strategies quantified earlier as an explicit input.

DEFINITION 8. A positional k-Skolem strategy CPG $\mathcal{P} = (V, v_0, \mathbb{A}, m, \mu, c)$ is a function $\zeta : V \times \mathbb{A}^k \to \mathbb{A}$. We write $SkoStr(\mathcal{P}, k)$ for the set of positional k-Skolem strategies in $\mathcal{P}$.

A positional k-Skolem strategy ζ can pick an action based on the current vertex of the game and k actions that have been selected previously. The intuition is that the strategy for player (or strategy variable) $2k$ (which is existentially quantified) can observe the actions selected by the k-universally quantified strategies before it.

DEFINITION 9. Assume e is a function that maps each $k \in [m]$ to a positional k-Skolem strategy in $\mathcal{P}$ and let $o : [m] \to PosStr(\mathcal{P})$ map each each $k \in [m]$ to a positional strategy in $\mathcal{P}$. We combine e

and o into a mapping $com(e, o) : [2m] \to Str(\mathcal{P})$ as follows. For a path $u \in V^+$, we define $last(u) \in V$ as the last vertex in v. For an odd index $2k - 1$ we then define $com(e, o)(2k - 1) : V^+ \to \mathbb{A}$ as

$$com(e, o)(2k - 1)(u) := o(k)\big(last(u)\big)$$

For an even index $2k$ we define $com(e, o)(2k) : V^+ \to \mathbb{A}$ as

$$com(e, o)(2k)(u) := e(k)\big(last(u), (o(1)(last(u)),$$
$$o(3)(last(u)),$$
$$\ldots,$$
$$o(2k - 1)(last(u)))\big).$$

The idea of $com(e, o)$ is to combine the Skolem strategies in e and the positional strategies in o into a strategy for each player. For odd player we simply take the strategy given by o (applying it to the last vertex in the given sequence). For even players, we query the Skolem strategy given by e and provide it with the actions that all universally quantified (odd) strategies before it have selected. We can now state that CPG are positionally determined by making use of Skolem functions:

PROPOSITION 4 ([40]). Let $\mathcal{P} = (V, v_0, \mathbb{A}, m, \mu, c)$ be a CPG. We have that $\mathcal{P}$ is won by the existential players if and only if

$$\exists_{k \in [m]} \zeta_k \in SkoStr(\mathcal{P}, k). \bigvee_{k \in [m]} f_k \in PosStr(\mathcal{P}).$$
$$even\Big(Play_{\mathcal{P}}(v_0, \prod_{k \in [2m]} com(e, o)(k))\Big)$$

where $e(k) := \zeta_k$ and $o(k) := f_k$ for $k \in [m]$.

Proposition 4 states that instead of following the quantifier prefix as in Definition 6 we can instead quantify over Skolem functions $\zeta_1, \ldots, \zeta_m$ for all existentially quantified variables. Put informally, the proposition thus states that existentially quantified strategies only need to know the current vertex and all actions selected by universally quantified strategies quantified before it. A proof Proposition 4 follows directly from the the construction in [40, Thm. 4.1].

D.5 The Second Implication

Using Proposition 4 we can now prove:

PROPOSITION 5. $\mathcal{P}_\Pi$ is won by the existential team if and only if $zip(\Pi) \in \mathcal{L}(\mathcal{B})$.

We prove both directions of Proposition 5 separately (in Lemmas 11 and 12).

LEMMA 11. If $\mathcal{P}_\Pi$ is won by the existential team then $zip(\Pi) \in \mathcal{L}(\mathcal{B})$.

PROOF. Assume that $\mathcal{P}_\Pi$ is won by the existential player. Using Proposition 4 we thus get positional sklolem functions $\zeta_1, \ldots, \zeta_m$ such that for

$$\bigvee_{k \in [m]} f_k \in PosStr(\mathcal{P}). \, even\Big(Play_{\mathcal{P}_\Pi}(v_0, \prod_{k \in [2m]} com(e, o)(k))\Big) \quad (5)$$

where $e(k) := \zeta_k$ and $o(k) := f_k$ for $k \in [m]$.

We use the Skolem functions to construct an accepting run DAG $\mathbb{D}$ of $\mathcal{B}$ on $zip(\Pi)$. We construct $\mathbb{D}$ iteratively. We begin with a DAG that consist of the single node $((q_0, \dot{s}), 0)$, i.e., we start with the

unique initial state of $\mathcal{B}$ (note that by definition of run DAGs, nodes are indexed by their current depth, cf. Appendix A). Now assume that there exists some node that we have not visited. Let this note be $((q, s), N)$. We observe that (q, s, N) is a vertex in $\mathcal{P}_\Pi$.

The Nth input read by $\mathcal{B}$ on $zip(\Pi)$ is $zip(\Pi)(N)$. By definition of $\mathcal{B}$ the transition function from (q, s) on the Nth input is

$$\delta'\big((q, s), zip(\Pi)(N)\big) = \bigwedge_{a_1 \in \mathbb{A}} \bigvee_{a_2 \in \mathbb{A}} \cdots \bigvee_{a_{2m} \in \mathbb{A}} (q', s') \qquad (6)$$

where $q' = \delta(q, zip(\Pi)(N)[\pi \mapsto s])$ and $s' = \kappa\big(s, \prod_{i \in Agts} a_{\vec{x}(i)}\big)$.

We need to add children of the node $((q, s), N)$ that full the above formula. We will construct a set $Y \subseteq Q \times S$ that is a model for the above, i.e., $Y \models \delta'\big((q, s), zip(\Pi)(N)\big)$. We will construct Y by following all the action selection in the prefix of $\delta'\big((q, s), zip(\Pi)(N)\big)$ and construct an intermediate set of functions $Z \subseteq ([2m] \to \mathbb{A})$. We can think of Z as combinations of actions $a_1, \ldots, a_{2m}$ that we select in the prefix of Eq. (6). This set Z is constructed in accordance with the Skolem functions $\zeta_1, \ldots, \zeta_m$, i.e., for every disjunctive choice we will selected the action that is picked by the respective Skolem function. Initially, we set $Z = \{\{\}\}$ as the singleton set containing only the empty function $\{\} : [2m] \to \mathbb{A}$. For every k from 1 to $2m$ we do the following: If k is odd, so action a_k is chosen conjunctively we add all possible actions. That is, we update $Z := \{\vec{a}[k \mapsto a] \mid \vec{a} \in Z, a \in \mathbb{A}\}$. If k is even, so $k = 2k'$ for some k', we can pick one action for k for each element in Z. Given $\vec{a} \in Z$ (so $\vec{a}$ is a function $\{1, \ldots, k-1\} \to \mathbb{A}$) we define the action $a_{\vec{a}}$ as

$$a_{\vec{a}} := \zeta_{k'}\big((q, s, N), (\vec{a}(1), \vec{a}(3), \ldots, \vec{a}(2k'-1))\big).$$

That is, we use the k'th Skolem functions and query it with the current vertex (q, s, N) and the action selected by all previously chosen conjunctive action choices (at odd positions) in $\vec{a}$. We then update $Z := \{\vec{a}[k \mapsto a_{\vec{a}}] \mid \vec{a} \in Z\}$.

After repeating this procedure, we have constructed a set $Z \subseteq ([2m] \to \mathbb{A})$ that maps all players to actions. Informally, this corresponds to a possible assignment of the actions selected in the prefix of $\delta'\big((q, s), zip(\Pi)(N)\big)$. For each $\vec{a} \in Z$ we can define a unique $s_{\vec{a}}$ by following the construction in the definition of $\mathcal{B}$. That is, we define $s_{\vec{a}} = \kappa\big(s, \prod_{i \in Agts} \vec{a}(\vec{x}(i))\big)$. We then define

$$Y := \Big\{\big(\delta(q, zip(\Pi)(N)[\pi \mapsto s]), s_{\vec{a}}\big) \mid \vec{a} \in Z\Big\}.$$

It is easy to see that $Y \models \delta'\big((q, s), zip(\Pi)(N)\big)$: In the selection of actions (when constructing Z) we have consider all possible actions for each conjunctive choice and picked a particular action for each disjunctive choice.

In our run DAG we now add a node $((q', s'), N+1)$ for each $(q', s') \in Y$, and add an edge from $((q, s), N)$ to $((q', s'), N+1)$.

Let $\mathbb{D}$ be the infinite DAG obtained by following this construction. It is easy to see that $\mathbb{D}$ is a valid run of $\mathcal{B}$ on $zip(\Pi)$ (as we have argued above all children that we add to any given node satisfy the transition formula).

It remains to argue that $\mathbb{D}$ is accepting. For this we consider an arbitrarily infinite path τ in $\mathbb{D}$, s.t.,

$$\tau = ((q_0, s_0), 0)((q_1, s_1), 1)((q_2, s_2), 2) \cdots$$

We define an analogous path in $\mathcal{P}_\Pi$ as follows (by simply regrouping parenthesis):

$$\tau' = (q_0, s_0, 0)(q_1, s_1, 1)(q_2, s_2, 2) \cdots$$

In construction $\mathbb{D}$, we always mimicked the choice for each distinctively chosen action by using the Skolem functions $\zeta_1, \ldots, \zeta_m$. We thus get that τ' is a play in $\mathcal{P}_\Pi$ under these Skolem functions: That is, there exist $f_1, f_2, \ldots, f_m \in PosStr(\mathcal{P}_\Pi)$ such that

$$Play_{\mathcal{P}_\Pi}\big(v_0, \prod_{k \in [2m]} com(\boldsymbol{e}, \boldsymbol{o})(k)\big) = \tau'$$

where $\boldsymbol{e}(k) := \zeta_k$ and $\boldsymbol{o}(k) := f_k$ for $k \in [m]$. By the choice of $\zeta_1, \ldots, \zeta_m$ (cf. Eq. (5)) we thus get that $even(\tau')$.

Now the sequence of colors traversed in τ' is the same as the sequence of colors in the path τ in the run DAG. The minimal color that appears infinitely often in τ is thus also even and so the path is accepting. As this holds for all paths, $\mathbb{D}$ is accepting.

We have constructed an accepting run DAG of $\mathcal{B}$ on $zip(\Pi)$, so $zip(\Pi) \in \mathcal{L}(\mathcal{B})$ as required. □

LEMMA 12. *If $zip(\Pi) \in \mathcal{L}(\mathcal{B})$ then $\mathcal{P}_\Pi$ is won by the existential team.*

PROOF. Assume that $zip(\Pi) \in \mathcal{L}(\mathcal{B})$, and let $\mathbb{D}$ be an accepting run DAG of $\mathcal{B}$. We will use this DAG to construct Skolem functions $\zeta_1, \ldots, \zeta_m$ such that

$$\bigvee_{k \in [m]} f_k \in PosStr(\mathcal{P}). \, even\Big(Play_{\mathcal{P}_\Pi}\big(v_0, \prod_{k \in [2m]} com(\boldsymbol{e}, \boldsymbol{o})(k)\big)\Big) \, (7)$$

where $\boldsymbol{e}(k) := \zeta_k$ and $\boldsymbol{o}(k) := f_k$ for $k \in [m]$. By Proposition 4 this would imply that $\mathcal{P}_\Pi$ is won by the existential team.

For any node $x = ((q, s), N)$ in the run DAG $\mathbb{D}$, the children of $((q, s), N)$ satisfy

$$\delta'\big((q, s), zip(\Pi)(N)\big) = \bigwedge_{a_1 \in \mathbb{A}} \bigvee_{a_2 \in \mathbb{A}} \cdots \bigvee_{a_{2m} \in \mathbb{A}} (q', s') \qquad (8)$$

where $q' = \delta(q, zip(\Pi)(N)[\pi \mapsto s])$ and $s' = \kappa\big(s, \prod_{i \in Agts} a_{\vec{x}(i)}\big)$.

We will construct functions $\alpha_1^x, \ldots, \alpha_m^x$ where $\alpha_k^x : \mathbb{A}^k \to \mathbb{A}$ that will pick a choice for each disjunction based on the previous choices for each conjunction. We first define $\alpha_1^x : \mathbb{A} \to \mathbb{A}$: For any $a^1 \in \mathbb{A}$ we define $\alpha_1^x(a^1)$ by *fixing* the first conjunctively chosen action a_1 in Eq. (8) to be a^1. As a_1 was chosen conjunctivley, the children of $x = ((q, s), N)$ still satisfy Eq. (8) with a_1 fixed to a^1. Now define $a^2 \in \mathbb{A}$ as some valid choice for the (disjunctively chosen) a_2. That is, we define a^2 such that the children of $x = ((q, s), N)$ in $\mathbb{D}$ still form a model of Eq. (8) with actions $a_1 := a^1, a_2 := a^2$ fixed. We set $\alpha_1^x(a^1) := a^2$. After having defined α_1^x we can define $\alpha_2^x(a^1, a^3)$: We fix the first conjunctively chosen action be a^1, the second disjunctive action be $\alpha_1^x(a^1)$ and the second conjunctive action be a^3, and the define $\alpha_2^x(a^1, a^3)$ as some action that we can select for the second disjunctive choice such that the children of $x = ((q, s), N)$ in $\mathbb{D}$ satisfy the subformula with those actions fixed. The construction of the remaining $\alpha_1^x, \ldots, \alpha_m^x$ is analogous. Intuitively, each α_k^x serves as a Skolem function for the kth disjunctive action by fixing an action based solely on the earlier conjunctive actions. Together they guarantee that by following the selected action by each Skolem function for each disjunctive choice, we always reach some child of $x = ((q, s), N)$ in $\mathbb{D}$.

We can now define the desired positional Skolem functions $\zeta_1, \ldots, \zeta_m$ for $\mathcal{P}_\Pi$. We define ζ_k as follows: Given any vertex $v = (q, s, N)$ in $\mathcal{P}_\Pi$ and actions $a_1, \ldots, a_k \in \mathbb{A}$ (corresponding to the actions selected by all universally quantified strategies before $2k$) we check if $x := ((q, s), N)$ is a node in $\mathbb{D}$. If there does not exists such a node we can return an arbitrary action (we will later argue that this situation will never be reached in any play $\mathcal{P}_\Pi$). Otherwise x is a node in $\mathbb{D}$ and we get the Skolem functions $\alpha_1^x, \ldots, \alpha_m^x$ we have constructed earlier. We define $\zeta_k(v, (a^1, \ldots, a^k)) := \alpha_k^x(a^1, \ldots, a^k)$, i.e., we select the action that α_k^x picks for the kth disjunction when using the actions $a^1, \ldots, a^k$ for the previous k conjunctions.

It remains to argue that the Skolem functions $\zeta_1, \ldots, \zeta_m$ fulfill Eq. (7). Let $f_1, \ldots, f_m \in PosStr(\mathcal{P}_\Pi)$ and consider the resulting play

$$\tau = Play_{\mathcal{P}_\Pi}\left(v_0, \prod_{k \in [2m]} com(\boldsymbol{e}, \boldsymbol{o})(k)\right)$$
$$= (q_0, s_0, 0)(q_1, s_1, 1)(q_2, s_2, 2) \cdots$$

where $\boldsymbol{e}(k) := \zeta_k$ and $\boldsymbol{o}(k) := f_k$ for $k \in [m]$. We consider the equivalent path in the run DAG (by regrouping parenthesis):

$$\tau' = ((q_0, s_0), 0)((q_1, s_1), 1)((q_2, s_2), 2) \cdots .$$

By construction of $\zeta_1, \ldots, \zeta_m$ we always pick the actions that are disjunctively chosen in accordance with $\mathbb{D}$. It is therfore easy to see that τ' is an infinite path in $\mathbb{D}$. By the assumption that $\mathbb{D}$ is accepting the minimal color that appears infinitely often is even. As the automaton state sequence agrees with that of τ, we thus get that $even(\tau)$ holds. So $\zeta_1, \ldots, \zeta_m$ satisfy Eq. (7). By Proposition 4 this implies that $\mathcal{P}_\Pi$ is won by the existential team, as required. □

By Proposition 3, we get that Π satisfies Eq. (3) iff $\mathcal{P}_\Pi$ is won by the existential team (Proposition 3). By Proposition 5, $\mathcal{P}_\Pi$ is won by the existential team iff $zip(\Pi) \in \mathcal{L}(\mathcal{B})$. Consequently, Π satisfies Eq. (3) iff $zip(\Pi) \in \mathcal{L}(\mathcal{B})$, proving Lemma 9 and thus of Proposition 2.

E DETAILS ON THE SECTION 7

In this section, we provide additional details on the formulas checked in Section 7.

E.1 Details on Section 7.1

We check the following HyperSL[SPE] formula:

$$\exists x. \forall x_1, \ldots, x_n. \left(\bigwedge_{i=1}^n \mathsf{G}\left(\langle wt, i\rangle_\pi \to \mathsf{F}\neg\langle wt, i\rangle_\pi\right)\right)$$
$$[\pi : (sched \mapsto x, y_1 \mapsto x_1, \ldots, y_n \mapsto x_n)]$$

This formula states that the scheduling agent $sched$ has a strategy such that none of the working agents $y_1, \ldots, y_n$ starves, i.e., whenever agent y_i waits for a grant (modeled by proposition $\langle wt, i\rangle$), it will eventually not wait any more. Note that this formula is equivalent to the SL[1G] specification used by Cermák et al. [23]. We check it for various values of $n \in \mathbb{N}$ and give the results in Table 1.

E.2 Details on Section 7.2

In Section 7.2, we check random formulas from a range of different families. We assume we are given a CGS $\mathcal{G}$ over atomic proposition

AP and agents $Agts$ (These CGS are obtained automatically from the ISPL models from [39]).

Security (Sec). We select some agent $i \in Agts$ and some AP $g \in AP$ modeling a goal, an AP $h \in AP$ modelling a high-security input, and an AP $o \in AP$ modeling an observable output. We then construct the following HyperSL[SPE] formula:

$$\exists x. \forall y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_n. \exists z_1, \ldots, z_n.$$
$$(\mathsf{F}\, g_\pi \wedge \mathsf{G}(o_\pi \leftrightarrow o_{\pi'}) \wedge \mathsf{F}(h_\pi \leftrightarrow h_{\pi'}))$$
$$\begin{bmatrix} \pi : (y_1, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_n) \\ \pi' : (z_1, \ldots, z_n) \end{bmatrix}$$

This formula states that i has a strategy to eventually reach g. Moreover, it may not leak all information about h via o. We model this using the idea of non-inference [43]. The idea is that the behavior in o should not leak h, so there must be "plausible deniability". That is, the same observation via o is also possible for some different input sequence via h.

Concretely, i should be able to reach g on path π no matter what the other agents play. In addition, there must exists some path π' (which we state by quantifying the strategies of all agents existentially), that has the same observations $\mathsf{G}(o_\pi \leftrightarrow o_{\pi'})$ but a different high-security input ($\mathsf{F}(h_\pi \leftrightarrow h_{\pi'})$).

Good-Enough Synthesis (GE). In many situations, asking for a strategy that wins in all situations is too restrictive. Instead, it often suffices to look for strategies that are *good-enough* (GE), i.e., strategies that win on every possible input sequence for which there exists a winning output sequence [1, 3]. We can express this formally using HyperSL[SPE]. Concretely, assume that $\varphi = \mathbb{Q}_1 x_1, \ldots, \mathbb{Q}_m x_m. \psi[\pi : \vec{x}]$ is any HyperSL formula over a single path variable (or, equivalently, a SL[1G] formula). We want to express that that φ only needs to holds on traces with input $i \in AP$, on which some path with the same input actually satisfies ψ.

We can express that φ is a GE-strategy as follows:

$$\mathbb{Q}_1 x_1, \ldots, \mathbb{Q}_m x_m. \forall y_1, \ldots, y_m.$$
$$(\mathsf{G}(i_\pi \leftrightarrow i_{\pi'}) \wedge \psi[\pi'/\pi]) \to \psi$$

where we write $\psi[\pi'/\pi]$ for the formula with all occurrences of π replaced with π'.

In the formula, we quantify over strategies $x_1, \ldots, x_m$ as in φ and use these strategies to construct path π. Afterwards, we universally quantify over any path π' in the system by picking strategies $y_1, \ldots, y_n$ for all agents. We then state that ψ only needs to hold on π provided π' has the same input and satisfies ψ. Phrased differently, π only needs to win, provided some path with the same inputs can ensure ψ. Note that, depending on the prefix in φ, this is not expressible in weaker hyperlogics such as HyperATL* and HyperATL$_S^*$.

Random (Rnd). For the random category, we use a random LTL formula (sampled using spot [30]) and add a prefix of quantifiers to yield a HyperSL[SPE] formula.