Specifying Agent Strategy Spaces via LTL Synthesis

Benjamin Aminof[1] , **Giuseppe De Giacomo**[2] , **Aniello Murano**[3] , **Sasha Rubin**[4]

[1]Technical University of Vienna, Austria
[2]University of Oxford, United Kingdom
[3]University of Naples "Federico II', Italy
[4]University of Sydney, Australia

Abstract

We study a model of Agentic AI, building on LTL synthesis originally studied in formal methods, that consists of autonomous agents with independent sequential decision-making capabilities. Specifically, we associate with each agent a goal expressed in LTL, and assumptions on the strategies employed by its peers and that the agent can exploit while synthesizing a strategy to realize its goal. While we can solve the synthesis problem under assumptions for each such agent we are not only interested in (1) synthesizing strategies for individual agents. Indeed, assumptions in turn are recursively defined through these strategy spaces. Importantly, we do not assume the ability to access or analyze an agent's internal strategy, as we make no assumptions about the nature of the decision makers, which may be, for example, ML-based. Instead, we focus on (2) characterizing the set of traces that are generated by strategies that realize the specification assigned to each agent. Using this characterization, we are able to (3) verify that the whole system, when in execution, satisfies a global objective, regardless of the strategies chosen by the agents from their allowed spaces. Moreover, by observing the evolution of the execution trace, we can (4) identify whether an agent makes a move that violates its specification and assign precise responsibility for the violation. Technically, we present automata-theoretic techniques to solve these problems, and show that each of them is 2EXPTIME-complete, matching the complexity of classical LTL synthesis.

1 Introduction

Temporal Synthesis studies the automatic synthesis of interactive programs (strategies) from declarative specifications expressed in temporal logic (Church 1963; Pnueli and Rosner 1989; Finkbeiner 2016; Ehlers et al. 2017). In AI Temporal Synthesis has been used to provide a principled foundation for strategic reasoning in autonomous AI systems, see e.g., (De Giacomo and Vardi 2015; Cermák, Lomuscio, and Murano 2015; De Giacomo and Rubin 2018; Camacho, Bienvenu, and McIlraith 2019; Camacho, Bienvenu, and McIlraith 2018; Aminof et al. 2019; Geatti, Montali, and Rivkin 2024; Aminof et al. 2024). The key to this research path lies in the rich body of concepts developed in reasoning about actions (Reiter 2001) and planning (Ghallab, Nau, and Traverso 2016), combined with a deep treatment of nondeterministic environments and temporal objectives. In such settings, plans must be treated as strategies rather than being blurred with individual execution traces, and goal satisfaction evolves during execution rather than being reducible to reaching states with fixed properties. These features are naturally captured within the temporal synthesis framework.

In this paper, we exploit temporal synthesis to investigate a setting for Agentic AI where we have only partial control over the strategic actions of a set of autonomous agents with independent sequential decision-making capabilities. Crucially, our focus is not on synthesizing individual strategies, but on specifying and reasoning about the spaces of strategies that agents are allowed to choose from.

Specifically, we adopt a third-person view (cf. the first-person view of (Aminof et al. 2025a)). Imagine a "manager" of a set of independent autonomous agents with their own decision-making capabilities. The manager gives each agent a task and some information about the behavior/task of the other agents (that they can exploit to achieve their task). However, the manager knows that the agents are not fully under his control, i.e., each of the agents may not work to achieve exactly the given task, but some related goal due to the agent's bias and individual traits tainting what it was asked to do (and the same thing goes for the assumption that the agent makes about the other agents' behavior). Thus, in order to reason about the agents, the manager writes specifications (taking into account the information he provided to the agents, as well as his knowledge of their individual biases and traits) for each of the agents, that describe the set/space of possible strategies that each of the agent may choose.

These specifications are based on traces over a set of atomic propositions. Each agent (including the environment) controls exclusively only a subset. When all agents act together, they generate traces, also called "plays". Each instant in the trace is a "round" where each proposition is set to some boolean value by the agent who controls it. The designer has control over the turn-taking during a round: the agents are partitioned into groups such that the agents in each group move in parallel, and in each round, the groups move one after the other.

Goals for each agent are specified as a desired set of traces, using LTL (linear-time temporal logic). Assumptions instead are restrictions on the set of possible strategies adopted independently by each of the other agents. In fact,

we specify such sets of strategies *only indirectly* through enforcement of trace specifications or previously defined assumptions.

Notably, we do not assume that strategies are objects that one can observe and predicate about, as in Strategy Logic (Mogavero et al. 2014). Only executions compatible with a strategy are indeed observable. So, crucial to our setting is that given a space of strategies, we can build an automaton that recognizes the set of traces compatible with some strategy in the set. The possibility of moving from strategies to traces, and vice versa, is at the base of our approach.

In this framework, we study: 1. Solving synthesis under these kinds of nested assumptions for each of the individual agents. 2. Characterizing the set of traces that are generated by strategies that realize the specification assigned to each agent. 3. Verifying that the whole system, when in execution, satisfies a given temporal property for all choices of strategies from the agents' allowed spaces. 4. Identifying, by observing the execution trace, whether an agent makes a move that violates its specification and assigning precise responsibility for the violation. Technically, we present automata-theoretic techniques to solve each of these problems for LTL, and show that the corresponding decisions problems are 2EXPTIME-complete, matching the complexity of classical LTL realizability, i.e., for the problems in common (synthesis/realizability), adopting our rich framework comes at no extra computational cost.

2 Related Work

Agents can exploit assumptions they have about their environment, including other agents, in order to achieve their goal (Harel and Pnueli 1984). Assumptions have been modeled as finite-state systems, as is typical in planning, e.g., (Camacho, Bienvenu, and McIlraith 2019), and more generally by temporal-logic formulas, e.g., (Pnueli and Rosner 1989; Fisman, Kupferman, and Lustig 2010; Brenguier, Raskin, and Sankur 2017; Aminof et al. 2018; Fijalkow et al. 2020). Typical of these works is that assumptions are not nested (i.e., agent i can have an assumption about agent j's goal, but not an assumption about agent j's assumptions). In contrast, our work allows nested assumptions, e.g., agent i has an assumption about agent j's goal as well as agent j's assumptions about other agents' goals, including agent i. In fact, we introduce simple and direct expressions to capture these nested assumptions. Moreover, our expressions allow two agents to have different, even contradictory, assumptions about the same agent representing their particular points of view.

An alternative approach to our direct word-automata approach to solving the computational problems presented in this paper may be to use rich strategic logics such as Strategy Logic (SL) (Mogavero et al. 2014). For example, for the simple case where all agents move in parallel, one can naturally express our synthesis problem as an SL model-checking problem, and thereby obtain an algorithm that solves our synthesis problem. Unfortunately, such an algorithm is based on complex tree-automata, and moreover, will be exponentially more expensive than the optimal algorithm we present in this paper. Overall, solving all of the problems we present with a purely SL approach, would require overcoming three hurdles at the same time: (1) employing an extension of SL that can handle our mixture of turn-taking and parallel moves; (2) being able to reason about infinite-traces while talking about finite-traces (see the Guardrail Problem); all the while, (3) finding a way to cast these problems in a more effective fragment of Strategy Logic which can be solved in 2EXPTIME, e.g., (Gardy, Bouyer, and Markey 2020).

The work closest to ours is (Aminof et al. 2025a), which investigates LTL synthesis under structured assumptions about the environment. That model can be thought of as a special case of our framework with a very simple two-level assumption, that exclusively concerns the environment except for a single agent, called the protagonist, whose assumptions refer to the other agents. In particular, that paper takes a first person-view of the protagonist. In contrast, we take a third-person view, and allow arbitrarily complex, nested, and circular assumptions (circular in the sense that agent 1 can have an assumption about agent 2, while at the same time agent 2 can have an assumption about agent 1, and these can be nested in the assumptions of agent 3, etc.). In addition, whereas (Aminof et al. 2025a) had the environment move first and all the other agents move in parallel, we allow a general (fixed) division of the agents into parallel and sequential moves. Finally, (Aminof et al. 2025a) only studies the synthesis problem, whereas here, due to the third-person view, we also consider the model-checking and guardrail problems.

3 Preliminaries

The set of integers $\{1, \ldots, k\}$ is denoted $[k]$. For a set X, its powerset is denoted $\mathbb{P}(X)$, and the set of finite (resp. infinite) sequences of elements from X is denoted X^* (resp. X^ω). The empty sequence is written ϵ. For a sequence β, we write β_i for its ith element; the first element is β_0; the length of a finite sequence is $|\beta|$; the prefix of β of length i is denoted by $\beta_{<i}$ or $\beta_{\leq i-1}$; we write $\mathit{Inf}(\beta)$ for the set of elements appearing infinitely often in β, i.e., $x \in \mathit{Inf}(\beta)$ iff $x = \beta_i$ for infinitely many i. If β is a prefix of β' we also say that β' *extends* β. For a finite set AP of *atomic propositions* (aka *atoms*), sequences over $\mathbb{P}(AP)$ are called *traces* over AP. A *trace property* (aka *property*) is a set $\mathcal{P} \subseteq \mathbb{P}(AP)^\omega$ of infinite traces. If $\tau \in \mathcal{P}$ we say that τ *satisfies* $\mathcal{P}$. If $\mathcal{P}$ is non-empty we say that it is *satisfiable*. We will sometimes use logical operators for operations on trace properties. In particular, for $A, B \subseteq \mathbb{P}(AP)^\omega$, we may write $\neg A$ for A's complement ($\mathbb{P}(AP)^\omega \setminus A$), write $A \vee B$ for the union $A \cup B$, write $A \wedge B$ for the intersection $A \cap B$, and write $A \Rightarrow B$ for $\neg A \vee B$, i.e. for $(\mathbb{P}(AP)^\omega \setminus A) \cup B$.

3.1 Linear-time Temporal Logic (LTL)

Let AP be a set of atoms. The *formulas of LTL over AP* are defined by the following BNF (where $p \in AP$): $\varphi ::= p \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi$. We use the usual abbreviations, $\varphi \rightarrow \varphi' \doteq \neg\varphi \vee \varphi'$, $\mathsf{true} \doteq p \vee \neg p$, $\mathsf{F}\,\varphi \doteq \mathsf{true}\,\mathsf{U}\,\varphi$, $\mathsf{G}\,\varphi \doteq \neg\,\mathsf{F}\,\neg\varphi$, etc. The *size* $|\varphi|$ of a formula φ is the number of symbols in it. For $n \geq 0$, write τ_n for the valuation at

position n. For a trace τ, an integer n, and an LTL formula φ, the satisfaction relation $(\tau, n) \models \varphi$, stating that φ holds at step n of the sequence τ, is defined as follows: $(\tau, n) \models p$ iff $p \in \tau_n$; $(\tau, n) \models \varphi_1 \vee \varphi_2$ iff $(\tau, n) \models \varphi_1$ or $(\tau, n) \models \varphi_2$; $(\tau, n) \models \neg\varphi$ iff it is not the case that $(\tau, n) \models \varphi$; $(\tau, n) \models \mathsf{X}\,\varphi$ iff $(\tau, n+1) \models \varphi$; and $(\tau, n) \models \varphi_1 \,\mathsf{U}\, \varphi_2$ iff there exists $m \geq n$ such that: $(\tau, m) \models \varphi_2$ and $(\tau, j) \models \varphi_1$ for all $n \leq j < m$. We write $\tau \models \varphi$ if $(\tau, 0) \models \varphi$, read τ *satisfies* φ. The set of traces that satisfy φ is a trace property, and we will sometimes blur the distinction between the formula φ and this trace property.

3.2 Deterministic Automata

Our solution techniques compile formulas into automata (i.e., the automata-theoretic approach to logic). Since we will also be defining/solving games on automata, we use deterministic automata (both on finite- and infinite strings).

Transition systems. A *deterministic transition system* $T = (\Sigma, Q, \iota, \delta)$ consists of a finite *input alphabet* Σ (typically $\Sigma = \mathbb{P}(AP)$), a finite set Q of *states*, an *initial state* $\iota \in Q$, and a *transition function* $\delta : Q \times \Sigma \to Q$. The *size* of T is the number of its states. Let $\beta = \beta_0\beta_1 \cdots$ be a finite or infinite sequence of letters in Σ. The *run/path* induced by β (aka the *run of* T *on* β) is the sequence $q_0q_1 \cdots$ of states where $q_0 = \iota$ and $q_{i+1} = \delta(q_i, \beta_i)$ for every $i < |\beta|$. We extend δ to range over $Q \times \Sigma^*$ as follows: $\delta(q, \epsilon) = q$, and for $n > 0$, let $\delta(q, \beta_0\beta_1 \cdots \beta_n) = \delta(\delta(q, \beta_0 \cdots \beta_{n-1}), \beta_n)$. Given k deterministic transition systems $T_1, \ldots, T_k$ over the same alphabet, i.e., $T_i = (\Sigma, Q_i, \iota_i, \delta_i)$, for $i \in [k]$, their *product* $T_1 \times T_2 \ldots \times T_k$ is the deterministic transition system $(\Sigma, Q', \iota', \delta')$ where $Q' = Q_1 \times \ldots \times Q_k$, $\iota' = (\iota_1, \ldots, \iota_k)$, and $\delta((q_1, \ldots, q_k), a) = (\delta_1(q_1, a), \ldots, \delta_k(q_k, a))$.

Automata. A *deterministic automaton* (or simply *automaton*) $\mathcal{A} = (T, Acc)$ is a deterministic transition system $T = (\Sigma, Q, \iota, \delta)$ augmented with an *acceptance condition* Acc defined below. The acceptance condition determines which runs ρ are "successful", in which case we say that ρ *satisfies* the acceptance condition and that ρ is *accepting*. A string β over Σ is *accepted* by $\mathcal{A}$ if its run ρ satisfies the acceptance condition. The set of strings accepted by $\mathcal{A}$ is the *language* of $\mathcal{A}$, which we denote by $L(\mathcal{A})$. The *size* of an automaton is the number of its states. For automata on finite strings (*DFA*), $Acc \subseteq Q$ is the set of *final states*, and a finite run *satisfies* the acceptance condition if it ends in a final state. For automata on infinite strings, we work with Emerson-Lei acceptance conditions, which generalize all traditional acceptance conditions used for ω-regular languages (Emerson and Lei 1987; Hausmann, Lehaut, and Piterman 2024).

Emerson-Lei (EL) acceptance conditions are described by a triple (Γ, λ, B), where Γ is a finite set of *labels*, $\lambda\colon Q \to \mathbb{P}(\Gamma)$ is a *labeling function* that assigns to a state a (possibly empty) subset of labels, and $B\colon \mathbb{P}(\Gamma) \to \{\mathsf{true}, \mathsf{false}\}$ is a Boolean function over the set Γ (treated as variables). For a state $q \in Q$ and a label $l \in \lambda(q)$, we say that q *visits* l. We will sometimes (but not always) write B as a Boolean formula over the set Γ of variables, with the usual syntax and semantics; e.g., the formula $l_1 \wedge l_2 \wedge \neg l_3$ denotes the Boolean function that assigns true to a set $Z \subseteq \Gamma$ iff Z contains l_1 and l_2 but not l_3. Given a sequence $\rho \in Q^\omega$, the set of labels that are visited infinitely many times by states on ρ is $\mathit{Inf}_\lambda(\rho) = \bigcup\{\lambda(q) : q \in \mathit{Inf}(\rho)\}$. A sequence ρ satisfies the *EL acceptance condition* iff $B(\mathit{Inf}_\lambda(\rho)) = \mathsf{true}$, i.e., iff the set of labels that are visited infinitely often by states of ρ satisfies B. An automaton $\mathcal{A} = (D, (\Gamma, \lambda, B))$ with an EL acceptance condition, is called a *DELA (deterministic Emerson-Lei automaton)*.

A useful (and easily proved) property of DELAs is that they are effectively closed under Boolean operations:

Lemma 1. *1. Given a DELA $\mathcal{A} = (T, (\Gamma, \lambda, B))$, the DELA $(T, (\Gamma, \lambda, \neg B))$ accepts the complement of $L(\mathcal{A})$.*

2. Given DELA $\mathcal{A}_i = (T_i, (\Gamma_i, \lambda_i, B_i))$ for $i \in [k]$, suppose the Γ_i are pairwise disjoint, and the T_i have the same alphabet. Let $\mathcal{C}$ be the DELA $(T, (\Gamma, \lambda, B))$ where: $T = T_1 \times \cdots \times T_k$, $\Gamma = \cup_{i\in[k]}\Gamma_i$, the labeling function λ is defined by letting $\lambda(q_1, \cdots, q_k) = \bigcup_{i\in[k]} \lambda_i(q_i)$, and $B = \bigvee_{i\in[k]} B_i$ (resp. $\bigwedge_{i\in[k]} B_i$); then $\mathcal{C}$ accepts the language $\bigcup_{i\in[k]} L(\mathcal{A}_i)$ (resp. $\bigcap_{i\in[k]} L(\mathcal{A}_i)$).

Moreover, emptiness of DELAs is effective:

Lemma 2. *(Emerson and Lei 1987) The non-emptiness problem for DELAs is NP-complete. Moreover, it can be solved in (deterministic) time that is polynomial in the size of the DELA, polynomial in the size of the Boolean formula, and exponential in the number of its labels.*

Finally, we can compile LTL formulas into DELAs:

Theorem 3. *(Vardi and Wolper 1994; Safra 1988) Fix AP. Given an LTL formula φ over AP, one can build a DELA $\mathcal{A}_\varphi = (T, (\Gamma, \lambda, B))$ that accepts exactly the traces over AP that satisfy φ, whose size is at most 2EXP in $|\varphi|$, whose number of labels $|\Gamma|$ is at most EXP in $|\varphi|$, and whose function B can be written as a Boolean formula of size linear in the number of labels.*

4 Framework

We consider a set $V = \{1, 2, \cdots, N\}$ of $N > 1$ agents (an environment, if it exists, is considered to be one of the agents). The interaction between the agents produce traces over a set $\mathbf{Z}$ of atoms. Each agent v has exclusive control of a subset $\mathbf{Z}^v \subseteq \mathbf{Z}$. That is, $\mathbf{Z}$ is partitioned into subsets controlled by each agent: $\mathbf{Z} = \mathbf{Z}^1 \cup \ldots \cup \mathbf{Z}^N$, with $\mathbf{Z}^v \cap \mathbf{Z}^w = \emptyset$ for $v \neq w$. Note that while we allow certain agents to be assigned no atoms, we do require that at least 2 agents have at least one atom each (otherwise we are effectively in a single-agent setting).

An evaluation of the atoms in $\mathbf{Z}$ produces a "round". In each round, each agent selects an evaluation of the atoms it controls. We consider a quite general way of turn-taking during a round: the agents are partitioned into groups $G_1, \cdots, G_k$, such that the agents in each group select their evaluations in parallel, and within each round the groups take turns, i.e., first G_1, then G_2, ..., until G_k.

For example, the case $k = 1$ where all the agents are in a single group corresponds to the often modeled case where all agents move in parallel. As another example, for two agents

($N = 2$), on top of the parallel option we can also have $k = 2$ and $G_1 = \{1\}, G_2 = \{2\}$ or $G_1 = \{2\}, G_2 = \{1\}$; in the first case, in each round agent 1 moves first and agent 2 second, and in the second case agent 2 moves first and agent 1 second.

Multi-agent setting A *multi-agent setting* refers to the set $V = \{1, 2, \cdots, N\}$ of agents, their groups $G_1, G_2, \cdots, G_k$, the set $\mathbf{Z}$ of atoms, and the partition of $\mathbf{Z}$ into $\mathbf{Z}^1 \cup \cdots \cup \mathbf{Z}^N$. We fix these for the rest of the paper. As mentioned above, we assume that there are at least two agents, each with at least one atom. We introduce some notation. For $i \in [k]$, define $\mathbf{Z}(i) = \bigcup_{v \in G_i} \mathbf{Z}^v$ to be the set of atoms controlled by the agents in group G_i. We denote by $\bar{Z} \doteq Z(1) \cdots Z(k)$ sequences where $Z(i) \subseteq \mathbf{Z}(i)$ for every $i \in [k]$. Intuitively, $\bar{Z}$ represents the moves of all the agents in a single round. Observe that (for a fixed partitioning of the agents into groups) $\bar{Z}$ can be easily reconstructed from the set $\cup_{i \in [k]} Z(i)$ of the atoms appearing along it.

Plays *Plays* are infinite sequences of the form $\pi \doteq \bar{Z}_0 \bar{Z}_1 \cdots$, i.e., they are elements in $(\mathbf{Z}(1) \cdots \mathbf{Z}(k))^\omega$. It is convenient to interpret LTL formulas not over the play π, but instead over an induced trace where the moves of each of the groups at each time step are combined into a single letter. Formally, we interpret LTL formulas over the *induced trace* over $\mathbf{Z}$ defined as: $\cup_{i \in [k]} Z_0(i) \cdot \cup_{i \in [k]} Z_1(i) \cdots$. So, for instance, the first element of the trace induced by π is $\cup_{i \in [k]} Z_0(i)$, not $\bar{Z}_0(1)$. Observe that every such trace is induced by a unique play π. It will be technically convenient to sometimes blur the distinction between the play π and its induced trace, e.g., we may say that a play satisfies an LTL formula in place of saying that its induced trace does. The play view will be used mainly when talking about strategies, and the induced trace view when considering such plays to be models of LTL formulas.

Histories A *history* is a finite prefix of a play. For $j \in [k]$, define a *j-history* to be a sequence in $(\mathbf{Z}(1) \cdots \mathbf{Z}(k))^* \cdot (\mathbf{Z}(1) \cdots \mathbf{Z}(j-1))$, i.e., a j-history is a prefix of a play containing zero or more full rounds followed by a partial round (if $j \neq 1$) that contains only the moves of the first $j-1$ groups of agents. Note that 1-histories are of the form $(\mathbf{Z}(1) \cdots \mathbf{Z}(k))^*$, and we call these *full histories* since they only contain full rounds. For an agent $v \in G_j$ we also use the term *v-history* for j-history. The set of all j-histories (resp. v-histories) is denoted H_j (resp. H_v).

Strategies A *v-strategy* is a function $\sigma_v : H_v \to \mathbb{P}(\mathbf{Z}^v)$. The set of all v-strategies is denoted Ξ^v. For a non-empty set $A \subseteq V$ of agents, a *strategy profile for A* is a function $\bar{\sigma}$ that assigns strategies to agents in A, i.e., its domain is A, and $\bar{\sigma}(v) \in \Xi^v$ for every $v \in A$. A strategy in the image of $\bar{\sigma}$ is said to be *in* $\bar{\sigma}$. If A is the set of all agents, then $\bar{\sigma}$ is a *full strategy profile*, otherwise it is a *partial strategy profile*.

Consistent plays/histories A play/history ρ is *consistent* with a v-strategy σ if for every proper prefix $\rho_{<i}$ in the domain of σ we have $\rho_i \cap \mathbf{Z}^v = \sigma(\rho_{<i})$. Further, ρ is *consistent* with a strategy profile $\bar{\sigma}$ iff it is consistent with every strategy in the image of $\bar{\sigma}$. Note that if $\bar{\sigma}$ is full then there is a unique play, denoted $\text{PLAY}(\bar{\sigma})$, that is consistent with $\bar{\sigma}$.

Assumptions An *assumption for agent v* (aka *v-assumption*) is a function Θ which assigns a set of strategies to each agent distinct from v, i.e., the domain of Θ is $V \setminus \{v\}$, and $\Theta(w) \subseteq \Xi_w$ for every w. A partial strategy profile $\bar{\sigma}$ is *in* Θ (denoted $\bar{\sigma} \in \Theta$) if it has the same domain as Θ, and $\bar{\sigma}(w) \in \Theta(w)$ for every w.

Enforcing We say that *σ_v enforces the trace property $\mathcal{P}$ (resp. under the v-assumption Θ)* if for every profile $\bar{\sigma}$ for $V \setminus \{v\}$ (resp. for every $\bar{\sigma} \in \Theta$), extending $\bar{\sigma}$ to a full strategy profile $\bar{\sigma}'$ by letting $\bar{\sigma}'(v) = \sigma_v$ (and $\bar{\sigma}'(w) = \bar{\sigma}(w)$ for $w \neq v$) we have that the trace induced by $\text{PLAY}(\bar{\sigma}')$ satisfies $\mathcal{P}$. Note that if $\Theta(w) = \emptyset$ for some $w \neq v$, then every v-strategy enforces every trace property $\mathcal{P}$ under Θ (vacuously).

Remark 1. *Let α be the set of v-strategies that enforce φ under Θ. Note that (depending on φ), the larger are the sets of strategies of the other agents as defined by Θ, the harder it may be for agent v to enforce φ, i.e., α becomes potentially smaller, and in the extreme case, it may become the empty set. On the other extreme, if $\Theta(w) = \emptyset$ for some $w \neq v$, then α is the set of all v-strategies (no matter what φ is).*

Operators Cns$(-)$ and Enf$(-)$ The set of plays consistent with a strategy σ (resp. strategy profile $\bar{\sigma}$) is denoted $\text{Cns}(\sigma)$ (resp. $\text{Cns}(\bar{\sigma})$). We extend this notation to sets of strategies, and sets of strategy profiles, in the natural way: e.g., for a set $\bar{\Xi}$ of strategy profiles we have $\text{Cns}(\bar{\Xi}) \doteq \bigcup_{\bar{\sigma} \in \bar{\Xi}} \text{Cns}(\bar{\sigma})$. The set of v-strategies that enforce a trace property $\mathcal{P}$ is denoted $\text{Enf}_v(\mathcal{P})$; and we say that *$\mathcal{P}$ is v-enforceable* if this set is not empty. Similarly, we write $\text{Enf}_v(\varphi)$ for the set of v-strategies that enforce the traces that satisfy an LTL formula φ, and say that *φ is v-enforceable* if this set is not empty.

The following Lemma shows how to turn assumptions into trace properties; the proof is straightforward from the definitions, see also a similar statement in (Aminof et al. 2025a).

Lemma 4. *Fix an agent v, a trace-property $\mathcal{G}$, and an assumption Θ for v. A v-strategy enforces $\mathcal{G}$ assuming Θ iff it enforces Cns$(\Theta) \Rightarrow \mathcal{G}$.*

The importance of this Lemma is that it allows us to reason about assumptions using traces instead of strategies, and thus employ word automata instead of the more complex tree automata and logics over trees, as is common in formalisms for expressing complex strategic properties.

4.1 Specifications

As discussed in the introduction, each agent v has ascribed to it a goal φ_v and an assumption Θ_v about the behavior of the other agents. To ease the exposition, we will simply refer to these as the goal and assumption of agent v (without mentioning that these are ascribed to it by a third party). The goals φ_v are specified as a desired set of traces, e.g., using LTL, and the assumptions Θ_v are restrictions on the sets of possible strategies for each of the agents other than v. We now describe a novel formalism for specifying the goals and assumptions for agents.

Syntax. Consider the following grammar (in BNF):

$$\alpha_v ::= \text{Enforce}_v(\varphi) \mid \text{Enforce}_v(\varphi, \Theta_v)$$
$$\Theta_v ::= (\alpha_1, \cdots \alpha_{v-1}, \alpha_{v+1}, \cdots, \alpha_N)$$

where $v \in V$ varies over the agents, and φ varies over the set of all LTL formulas (over the atomic propositions $\mathbf{Z}$). The expressions α_v are called *strategies expressions*, and they denote sets of agent v's strategies. The expressions Θ_v are called *assumption expressions*, and specify the sets of strategies that each of the agents other than v are assumed to pick their strategies from. Intuitively, $\text{Enforce}_v(\varphi)$ is the set of agent-v strategies that enforce φ no matter what other agents do, whereas $\text{Enforce}_v(\varphi, \Theta_v)$ is the set of agent-v strategies that enforce φ provided that each agent $w \neq v$ uses a strategy from the set $\Theta_v(w)$ (note that if agent w uses a strategy not in $\Theta_v(w)$ then φ may or may not hold).

Remark 2. *Formally speaking, α_v and Θ_v (for $v \in V$) are non-terminals in the grammar, and a string generated by the grammar cannot contain non-terminals. Thus, e.g., (for $N = 2$) $\mathit{Enforce}_1(p \,\mathsf{U}\, q, \Theta_1)$ is not a string generated by the grammar since it contains an underived non-terminal Θ_1. On the other hand, $\mathit{Enforce}_1(p \,\mathsf{U}\, q, \mathit{Enforce}_2(\mathsf{F}\, p))$ is a string generated by the grammar. For the latter, we may also write $\alpha_1 = \mathit{Enforce}_1(p \,\mathsf{U}\, q, \Theta_1)$, $\Theta_1 = (\alpha_2)$, $\alpha_2 = \mathit{Enforce}_2(\mathsf{F}\, p)$.*

Semantics. We define, by mutual recursion, what each expression in the Syntax denotes:

1. $\text{Enforce}_v(\varphi)$ (resp. $\text{Enforce}_v(\varphi, \Theta_v)$) denotes the set of v-strategies enforcing φ (resp. under the assumption Θ_v).
2. $\Theta_v = (\alpha_1, \cdots, \alpha_{v-1}, \alpha_{v+1}, \cdots, \alpha_N)$ denotes the v-assumption defined by letting $\Theta_v(w) = \alpha_w$ for all $w \neq v$.

In general, depending on the context we will refer to α_v as either an expression (i.e., syntactically) or as a set of v-strategies (i.e., semantically). In particular, we will say that a strategy σ *satisfies* the expression α_v to mean that it is in the set α_v denotes, and that a trace is *consistent* with the expression α_v to mean that it is consistent with some strategy that satisfies α_v.

Example 1. *Consider $N = 2$, and the expressions: $\alpha_1 = \mathit{Enforce}_1(\varphi_1, \Theta_1), \Theta_1 = (\alpha_2), \alpha_2 = \mathit{Enforce}_2(\varphi_2)$.*

The expression α_2 says that agent 2 is trying to enforce φ_2 no matter which strategy agent 1 uses. The expression α_1 says that agent 1 is trying to enforce φ_1 provided that agent 2 uses a strategy from α_2, i.e., provided that agent 2 uses a strategy that (unconditionally) enforces φ_2.

In the setting of Example 1, the computational problem that asks to find an agent 1 strategy in α_1 is known as the "LTL synthesis under assumptions problem" (Aminof et al. 2018). As noted in that paper, this already captures many settings, including nondeterministic planning with temporally extended goals (for this, we treat agent 2 as the "environment", φ_2 captures the planning domain, and we treat agent 1 as the planning agent, and φ_1 as its goal).

Example 2. *In the previous example, agent 1's assumption about agent 2 happens to be equal to agent 2's specification. However, our formalism allows for these to be different, e.g.:*

$$\alpha_1 = \mathit{Enforce}_1(\varphi_1, \Theta_1) \quad \Theta_1 = (\mathit{Enforce}_2(\psi_2))$$
$$\alpha_2 = \mathit{Enforce}_2(\varphi_2, \Theta_2) \quad \Theta_2 = (\mathit{Enforce}_1(\psi_2))$$

Here agent 1 is trying to enforce φ_1, assuming that agent 2 is trying to enforce ψ_2. However, agent 2 is actually trying to enforce φ_2 (not ψ_2, as agent 1 assumes), assuming that agent 1 is trying to enforce ψ_2 (not φ_1, as he actually is).

Example 3. *A particular multi-agent generalization of synthesis under assumptions was given in (Aminof et al. 2025a). The setup can be expressed in our formalism as follows. For $N \geq 2$, partition the agents into two groups: $G_2 = \{N\}$ and $G_1 = \{1, 2, \cdots, N-1\}$. Thus, agent N (considered to be the environment) reacts to the parallel actions of the other agents. Now, define the following expressions:*

$$\alpha_N = \mathit{Enforce}_N(\varphi_N)$$
$$\alpha_i = \mathit{Enforce}_i(\varphi_i, \Theta_i) \text{ for } i \neq 1, N$$
$$\Theta_i = (S_1, \cdots, S_{i-1}, S_{i+1}, \cdots, S_{N-1}, \alpha_N) \text{ for } i \neq 1, N$$
$$\alpha_1 = \mathit{Enforce}_1(\varphi_1, (\alpha_2, \alpha_3, \cdots, \alpha_N))$$

where S_i is shorthand for $\mathit{Enforce}_i(\mathsf{true})$. α_N describes environment strategies that enforces φ_N no matter what the other agents do. α_i (for $i \neq 1, N$) describe the agent-i strategies that enforces φ_i provided that the environment uses a strategy from α_N (assuming no limitations on the strategies of the other agents). Finally, α_1 describes agent-1 strategies that enforces φ_1 provided that every other agent $i \neq 1$ uses a strategy from α_i.

This example (when $N > 3$) also shows that our formalism allows two agents (e.g., agents $1, 2$) to have different assumptions about a third agent (e.g., agent 3).

Note that the grammar does not allow one to write recursive expressions, i.e., where an expression is used inside its own assumption. While this recursive case may seem interesting at first, after further scrutiny it turns out to be less appealing, see Section 7.

4.2 Fundamental Problems

We consider four problems for a given multi-agent setting.

Realizability problem. Given an expression α_v, decide if there exists a v-strategy that satisfies α_v (or say there is none).

Synthesis problem. Given an expression α_v, return a finite-state v-strategy that satisfies α_v (or say there is none).

Model-checking problem. Given expressions α_v, one for each $v \in V$, and an LTL formula φ, decide if every trace in $\cap_v \text{Cns}(\alpha_v)$ satisfies φ, i.e., decide if every trace that is consistent with some strategy that satisfies α_v, for every $v \in V$, satisfies φ.

Guardrail problem. Given an expression α_v, build a deterministic automaton over finite strings that accepts those full histories h for which there is no v-strategy in α_v with which h is consistent. Such an automaton is called a *guardrail automaton for* α_v. The decision version of this

problem is to decide, given α_v and a full history h, if there is no v-strategy in α_v with which h is consistent.

We now discuss the intuitive meaning of having such a guardrail automaton $\mathcal{A}$. Consider a full strategy profile $\bar{\sigma}$. After $n+1$ rounds according to $\bar{\sigma}$, the history produced is $h = \bar{Z}_0\bar{Z}_1 \cdots \bar{Z}_n$. If h is accepted by $\mathcal{A}$, then this means that $\bar{\sigma}(v)$ is not in α_v, i.e., agent v violated its specification. Moreover, if h is the shortest full history consistent with $\bar{\sigma}$ that $\mathcal{A}$ accepts, then it was the last move of agent v, i.e., $Z_n \cap \mathbf{Z}^v$ that, intuitively, caused the violation. Indeed, the prefix $h' = \bar{Z}_0\bar{Z}_1 \cdots \bar{Z}_{n-1}$ of h, being shorter than h, is not accepted by $\mathcal{A}$. Hence, it is consistent with some non-empty subset $S \subseteq \alpha_v$ of v-strategies; and if agent v had continued to use any strategy $\sigma_v \in S$ (and the other agents continued using their strategies in $\bar{\sigma}$, as they did for h) also for the $n+1$ round, then unlike h, the resulting history would have remained consistent with σ_v and rejected by $\mathcal{A}$.

Note that the guardrail automaton $\mathcal{A}$ is meant to be used to monitor plays as they progress round-by-round. Thus, it is only designed to detect violations that occur within a finite number of rounds. To illustrate, consider the case where agent 1 has an atom p, and $\alpha_1 = \mathrm{Enf}_1(\mathsf{F}\,p)$. Since every history is consistent with α_1 then $\mathcal{A}$ accepts no history. Nonetheless, the strategy "never set p to be true" is not in α_1, but one cannot detect within a finite number of rounds if this strategy is used and not some strategy that is in α_1.[1]

5 Solving the Fundamental Problems

In this section we present an automata-theoretic approach to solving the fundamental problems stated in Section 4.2. We first state two theorems (whose proofs we defer to later sections) that will serve as the basis for our solutions.

The first theorem says, intuitively, that we can translate a strategy expression into a DELA:

Theorem 5. *Given a multi-agent setting, and a strategies expression α_v, one can construct a DELA $\mathcal{A}_{\alpha_v}$ whose language is Cns(α_v). Moreover, measured in the size of α_v, we have that $\mathcal{A}_{\alpha_v}$ is computable in 2EXPTIME, its size is at most 2EXP, and its Boolean formula has size at most 1EXP.*

This gives a useful corollary:

Corollary 6. *Given a multi-agent setting, and a strategies expression α_v, one can construct a DELA $\mathcal{A}$ such that for every v-strategy σ_v, we have that σ_v is in α_v iff σ_v enforces $L(\mathcal{A})$. Moreover, measured in the size of α_v, we have that $\mathcal{A}$ is computable in 2EXPTIME, its size is at most 2EXP, and its Boolean formula has size at most 1EXP.*

Proof. The case $\alpha_v = \mathrm{Enforce}_v(\varphi)$ follows immediately from Theorem 3. For the case $\alpha_v = \mathrm{Enforce}_v(\varphi, \Theta_v)$, proceed as follows. For every $w \neq v$, apply Theorem 5 to $\Theta(w)$ to get a DELA $\mathcal{A}_w$ whose language is $\mathrm{Cns}(\Theta(w))$. Apply Theorem 3 to get a DELA $\mathcal{A}_\varphi$ whose language is the set of traces satisfying φ. Apply Lemma 1 to get a DELA $\mathcal{A}$ for the language $(\cap_{w\neq v} L(\mathcal{A}_w)) \Rightarrow A_\varphi$. Correctness follows from Lemma 4. □

[1] Our automata-based methods can easily handle such violations that only occur on infinite traces.

The second theorem says that, given a multi-agent setting, and a DELA $\mathcal{A}$ over the alphabet $\mathbb{P}(\mathbf{Z})$, we can reduce the problem of whether agent v can enforce $L(\mathcal{A})$ to solving a certain two-player game played on a DELA built from $\mathcal{A}$. To state this theorem, we introduce the following notions.

Two-player games on automata. A *game on a DELA* is a tuple $(\mathcal{A}, \mathbf{A}, \mathbf{B})$ where $\mathbf{A}, \mathbf{B}$ are disjoint sets of atoms and $\mathcal{A}$ is a DELA with alphabet $\Sigma = \mathbb{P}(\mathbf{A} \cup \mathbf{B})$. We first informally describe the two-player game played on $\mathcal{A}$. The *players* (aka *opponents*) are called Adam and Eve. Adam controls the atoms from $\mathbf{A}$ and Eve from $\mathbf{B}$. The game proceeds by the players pushing a pebble along the transition system of $\mathcal{A}$ as follows: if the pebble is currently in q, first Adam moves by selecting $A' \subseteq \mathbf{A}$, then Eve moves by selecting $B' \subseteq \mathbf{B}$, and the pebble's position in the game is updated to the state $\delta(q, (A' \cup B'))$. Starting at the initial state, this interaction generates an infinite run of $\mathcal{A}$, and Adam is declared the winner if the run is accepting (and otherwise Eve is declared the winner).

We now describe this game formally. A *play* $\pi = A_0 \cdot B_0 \cdot A_1 \cdot B_1 \cdots$ is an element of $(\mathbb{P}(\mathbf{A}) \cdot \mathbb{P}(\mathbf{B}))^\omega$; this play is an alternative representation of the trace $(A_0 \cup B_0)(A_1 \cup B_1) \cdots$, *induced* by π. A *history* h is a finite prefix of a play. A *strategy for Adam* is a function $\sigma : (\mathbb{P}(\mathbf{A}) \cdot \mathbb{P}(\mathbf{B}))^* \to \mathbb{P}(\mathbf{A})$ that maps histories ending in Eve's moves (including the empty history since Adam moves first) to an Adam move. A *strategy for Eve* is a function $\sigma : (\mathbb{P}(\mathbf{A}) \cdot \mathbb{P}(\mathbf{B}))^* \cdot \mathbb{P}(\mathbf{A}) \to \mathbb{P}(\mathbf{B})$ that maps histories ending in Adam's moves to an Eve move. A play π is *consistent* with a strategy if for every history $\pi_{<i}$ in the domain of σ we have that $\pi_i = \sigma(\pi_{<i})$. If σ is a strategy for Adam and δ is a strategy for Eve then we write $\textsc{play}(\sigma, \delta)$ for the unique play consistent with both of these strategies. It will be technically convenient to sometimes blur the distinction between a play and the trace it induces. E.g., we say that a play π *satisfies* a trace property $\mathcal{P}$ if the trace induced by π is in $\mathcal{P}$; or say that a trace τ is consistent with a strategy σ if the play inducing τ is consistent with σ. We say that a strategy σ for Adam (resp. Eve) is *winning* if for all strategies δ for Eve (resp. Adam) we have that $\textsc{play}(\sigma, \delta)$ is accepted (resp. not accepted) by $\mathcal{A}$. Deciding if a given player has a winning strategy, and returning such a (finite-state) strategy, is called *solving* the game for that player. If $\mathbf{A}, \mathbf{B}$ are understood, we may write "solving the game on $\mathcal{A}$", instead of "solving the game $(\mathcal{A}, \mathbf{A}, \mathbf{B})$".

Proposition 7. *(Hausmann, Lehaut, and Piterman 2024; Aminof et al. 2025b) Games on DELAs can be solved in time polynomial in the size of the DELA, exponential in the number of labels, and polynomial in the size of the formula.*

We now state the second Theorem.

Theorem 8. *Given a multi-agent setting, an agent v, and a DELA $\mathcal{A}$ over the alphabet $\mathbb{P}(\mathbf{Z})$, one can build, in* PTIME*, a DELA game over an automaton $\mathcal{A}_v$ such that there is a v-strategy enforcing $L(\mathcal{A})$ iff Eve has a winning strategy in the game. Moreover, one can transform in* PTIME *an Eve winning strategy to a v-strategy enforcing $L(\mathcal{A})$.*

We now show how to leverage these results to solve our fundamental problems from Section 4.2. In addition, we also

establish tight computational complexities for these problems, and note that the lower-bounds already hold for the case that $N = 2$ (which can be considered the case of a single agent operating in an environment). Moreover, the complexity of our Realizability Problem is exactly the same as classical LTL Realizability which might be surprising since our formalism allows for arbitrarily nested and circular assumptions.

5.1 Solving the Realizability Problem

Given a strategies expression α_v, we must decide if there is a strategy in α_v. To do this, apply Theorem 5 to obtain the DELA $\mathcal{A}_{\alpha_v}$ whose language is $\text{Cns}(\alpha_v)$. Note that $L(\mathcal{A})$ is non-empty iff α_v is non-empty, which reduces the realizability problem to the non-emptiness problem of $\mathcal{A}_{\alpha_v}$. Combining the complexities given in Theorem 5, and the cost of solving the DELA non-emptiness problem (Lemma 2), we get that this procedure is in 2EXPTIME.

For the lower bound, note that the LTL synthesis problem, which is known to be 2EXPTIME-hard (Pnueli and Rosner 1990), can be easily encoded by using two of the agents in the given multi-agent setting. Thus, we have proven that:

Theorem 9. *The Realizability Problem is 2EXPTIME-complete.*

5.2 Solving the Synthesis Problem

Given a non-empty strategies-expression α_v, we must return a strategy in α_v. Apply Corollary 6 to get a DELA $\mathcal{A}$ such that a v-strategy is in α_v iff it enforces $L(\mathcal{A})$. Apply Theorem 8 to get a game on the DELA $\mathcal{A}_v$ such that there is a v-strategy enforcing $L(\mathcal{A})$ iff Eve has a winning strategy in the game on $\mathcal{A}_v$. Apply Proposition 7 to solve the game on $\mathcal{A}_v$, to get a winning strategy for Eve, and return the induced v-strategy. The complexity of this algorithm is 2EXPTIME.

5.3 Solving the Model-checking Problem

We reduce the model-checking problem to the realizability problem as follows. Given $\alpha_1, \cdots, \alpha_N, \varphi$, build an instance of the realizability problem by adding a new agent $(N + 1)$, with no atoms (i.e., $\mathbf{Z}^{N+1} = \emptyset$), placed in G_1 (any group will work because agent N has no atoms), and build the expression $\alpha_{N+1} = \text{Enforce}_{N+1}(\varphi, \Theta_{N+1})$ where Θ_{N+1} is defined by letting $\Theta_{N+1}(v) = \alpha_v$ for every $v \in [N]$. It is easy to see that agent $N + 1$ has a single strategy (which outputs the empty set at every point in time), and that every trace is consistent with it. Thus, this strategy is in α_{N+1} iff all traces consistent with Θ_{N+1} satisfy φ.

Alternatively, we can reason more directly as follows. For each $v \in [N]$, apply Theorem 5 to get a DELA $\mathcal{A}_v$ whose language is $\text{Cns}(\alpha_v)$; apply Theorem 3 to φ to get a DELA $\mathcal{A}_\varphi$ that accepts the traces that satisfy φ; apply Lemma 1 to build a DELA $\mathcal{A}'$ for the Boolean combination $\left(\bigcap_{v \in V} L(\mathcal{A}_v)\right) \cap \neg L(\mathcal{A}_\varphi)$, and check that $L(\mathcal{A}')$ is empty using Lemma 2.

Conclude that the Model-checking Problem is in 2EXPTIME. For the lower bound, we reduce from the Non-Realizability problem which is 2EXPTIME-hard since the Realizability problem is (Theorem 9), and since deterministic complexity classes are closed under complement. Given an instance α_v of the Non-realizability problem, build an instance of the model-checking problem by taking α_v, together with $\alpha_w = \text{Enforce}_w(\mathsf{true})$ for $w \neq v$, and $\varphi = \mathsf{false}$. It is easy to see that the formula false holds on all traces consistent with $\alpha_1, \cdots, \alpha_N$ iff there are no such traces, i.e., iff α_v is not realizable. Thus, we have proven that:

Theorem 10. *The Model-checking Problem is 2EXPTIME-complete.*

5.4 Solving the Guardrail Problem

Apply Theorem 5 to the given α_v to get a DELA $\mathcal{A}$ whose language is $\text{Cns}(\alpha_v)$. Let D be its deterministic transition system, and let Q be its set of states. Compute (using Lemma 2) the set $F \subseteq Q$ of states q for which $\mathcal{A}$, with initial state q, has an empty language. The guardrail automaton is the DFA (D, F). It is easy to see that a full history h is consistent with a strategy in α_v iff h is a prefix of a trace accepted by $\mathcal{A}$ iff h is rejected by (D, F).

Note that deciding if a full history h is consistent with some strategy in α_v is thus in 2EXPTIME. It is also 2EXPTIME-hard since if h is the empty string then this is just the Realizability Problem (see Theorem 9).

Thus, we have proven that:

Theorem 11. *Given α_v, we can compute a guardrail automaton in 2EXPTIME, and of size 2EXP. Moreover, the decision version of this Problem is 2EXPTIME-complete.*

6 Proving Theorems 5 and 8

We start with the proof of Theorem 8.

6.1 Proof of Theorem 8

Given a multi-agent setting, an agent v, and a DELA $\mathcal{A}$ over the alphabet $\mathbb{P}(\mathbf{Z})$. We will construct a DELA $\mathcal{A}_v$, and a game $(\mathcal{A}_v, \mathbf{A}, \mathbf{B})$ on this DELA, such that there is a v-strategy enforcing $L(\mathcal{A})$ iff Eve has a winning strategy in the game on $\mathcal{A}_v$.

In the special case where the grouping of agents is such that either all (resp. none) of the other agents move before v on each round, things relatively easily transfer to the 2-player setting by identifying v with Eve (resp. Adam) and directly simulating all the other agents by Adam (resp. Eve). Things get more complicated when in each round some of the moves of the other agents are visible to v while others are invisible, since then, neither Eve (resp. Adam) who sees (resp. does not see) her opponent's move in each round are suitable to represent v without further work. Our ability to always represent v by Eve, as we claim in Theorem 8 — transferring enforcing v-strategies to winning Eve strategies (and vice-versa) — relies solely on maintaining at each point in time the same information visible to Eve's strategy as to the v-strategy. This is achieved by time-shifting a portion of Adam's move from one round to the next: namely, the portion corresponding to the agents that do not move before v in the multi-agent setting. It is important to note that the co-operation inherent in lumping all the other agents into Adam, as well as the choice of what moves of Eve are visible

to Adam at each point in time, has no effect on the correctness of the reduction. Indeed, it is clear from the definition of enforcing strategies that no amount of cooperation among the other agents, or advance knowledge of the moves of v, can compromise in any way the fact that playing against a v-strategy enforcing $L(\mathcal{A})$ results in a play in $L(\mathcal{A})$.

We now formally describe the game $(\mathcal{A}_v, \mathbf{A}, \mathbf{B})$. As is clear from the discussion above, we set $\mathbf{A} = \cup_{w \neq v} \mathbf{Z}^w$ and $\mathbf{B} = \mathbf{Z}^v$. Before we describe the automaton $\mathcal{A}_v$, we need to first define how we time-shift.

Let m be such that v is in group G_m, and recall that the domain of v-strategies are v-histories. Such histories contain the moves of all the agents in previous rounds, and only the moves of the agents in groups $< m$ in the last round. Contrast this with strategies of Eve, whose domain (by our choice of $\mathbf{A}, \mathbf{B}$) are histories which contain the moves of all the agents *also in the last round* (except for v). Let $Z_t(l) \subseteq \mathbf{Z}(l)$ be the moves of the agents in group G_l at round t. To realign the strategies of v and Eve to have the same information available to them at each time step, we will construct $\mathcal{A}_v$ to accept not $L(\mathcal{A})$, but a slightly modified trace property which ignores $Z_0(m), \cdots, Z_0(k)$ (at time step 0), and associates the atoms in $Z_t(m) \cdots Z_t(k)$ from time step t with time-step $t-1$, before feeding them to $\mathcal{A}$. Thus, effectively, when Adam makes a move at round t (all of which is visible to Eve at round t), as far as $\mathcal{A}_v$ is concerned, the part of that move that would have been visible to v at round t remains associated with this round, whereas the other part is associated with the previous round (hence it is also visible to v at round t).

Definition 1 (The DELA $\mathcal{A}_v$)**.** *Given an agent $v \in G_m$, and a DELA $\mathcal{A} = ((\Sigma, Q, \iota, \delta), (\Gamma, \lambda, B))$ over the alphabet $\mathbb{P}(\mathbf{Z})$. Let $\mathbf{Y}^v = \cup_{i \in [m-1]} \mathbf{Z}(i) \cup \mathbf{Z}^v$, and define the DELA $\mathcal{A}_v = ((\Sigma, Q', \iota', \delta'), (\Gamma, \lambda', \neg B))$ as follows:*

- $Q' \doteq \{\iota'\} \cup (Q \times \mathbb{P}(\mathbf{Y}^v))$ *where ι' is a fresh state.*
- $\delta'(\iota', Z) \doteq (\iota, Z \cap \mathbf{Y}^v)$*, and* $\delta'((q, Y), Z) \doteq (q', Z \cap \mathbf{Y}^v)$ *where* $q' = \delta(q, Y \cup (Z \setminus \mathbf{Y}^v))$.
- $\lambda(\iota') = \emptyset$ *and* $\lambda(q, Z) = \lambda(q)$.

Define *shift*$(L(\mathcal{A}))$ to be the following trace property: a trace $Z_0 Z_1 Z_2 \cdots$ is in *shift*$(L(\mathcal{A}))$ iff the trace $\rho_0 \rho_1 \rho_2 \cdots$ is in $L(\mathcal{A})$, where $\rho_i = ((Z_i \cap \mathbf{Y}^v) \cup (Z_{i+1} \setminus \mathbf{Y}^v))$. It is not hard to see that $L(\mathcal{A}_v) = \Sigma^\omega \setminus$ *shift*$(L(\mathcal{A}))$ (observe that the acceptance condition of $\mathcal{A}_v$ is the negation of that of $\mathcal{A}$, since we are looking for winning strategies of Eve, i.e., strategies that enforce the non-acceptance of the DELA $\mathcal{A}_v$). Intuitively, for every input letter (except the first one), the automaton $\mathcal{A}'$ takes the moves of v and the agents in groups G_1 to G_{m-1} (that were stored in the second co-ordinate of the state of the automaton) from the previous letter, together with the moves of all the other agents from the current letter, and uses these to simulate a transition of $\mathcal{A}$. Notice that the moves of the agents in groups G_m to G_k (with the exception of agent v's moves) in the first input letter are discarded.

We now show how to transform a v-strategy enforcing $L(\mathcal{A})$ to an Eve winning strategy, and vice versa. This is quite simple (though a bit technically tedious), and amounts to (i) transforming v-histories which are composed of concatenations of the moves from k different groups, to histories in the game which are concatenations of moves of only 2 players; (ii) performing the relevant time shifts. For the first direction, given a v-strategy σ, define an Eve strategy σ' by letting $\sigma'(h') \doteq \sigma(h)$ where h is formed from h' by shifting each block of moves of the agents in groups $m+1$ to k one round to the left (and discarding them from the first round). Formally, let $h' = A_0 \cdot B_0 \cdots A_n \cdot B_n \cdot A_{n+1}$, and let $\bar{Z}_i$ be the concatenation of $(A_i \cap \mathbf{Z}(1)) \cdots (A_i \cap \mathbf{Z}(m-1))$ and $(B_{i+1} \cup (A_{i+1} \cap \mathbf{Z}(m)))(A_{i+1} \cap \mathbf{Z}(m+1)) \cdots (A_{i+1} \cap \mathbf{Z}(k))$. Finally, let h be $\bar{Z}_0 \bar{Z}_1 \cdots \bar{Z}_n \bar{Z}_{n+1}$ (note that $\bar{Z}_{n+1}$ is $(A_{n+1} \cap \mathbf{Z}(1)) \cdots (A_{n+1} \cap \mathbf{Z}(m-1))$). It is not too hard to see that if σ enforces $L(\mathcal{A})$ then σ' enforces *shift*$(L(\mathcal{A}))$.

For the other direction, we simply perform the reverse transformation than the one given above. Given an Eve strategy σ', define a v-strategy σ by letting $\sigma(h) \doteq \sigma'(h')$ where h' is formed from h as follows. Let $h = \bar{Z}_0 \bar{Z}_1 \cdots \bar{Z}_n$ where $\bar{Z}_i = Z_i(1) \cdots Z_i(k)$ for $i < n$, and $\bar{Z}_n = Z_n(1) \cdots Z_n(m-1)$; then $h' = A_0 B_0 \cdots A_{n-1} B_{n-1} A_n$ where: $A_0 = (\cup_{j<m} Z_0(j))$, and for $i > 1$, $A_i = (\cup_{j \leq m-1} Z_i(j)) \cup (\cup_{j \geq m} Z_{i-1}(j) \setminus \mathbf{Z}^v)$, and for $i < n$, $B_i = Z_i(m) \cap \mathbf{Z}^v$. Note that $A_0 = (\cup_{j \leq m-1} Z_0(j))$, and this amounts to making the (arbitrary) choice that in the two player game the first move A_0 of all the agents except v that are in groups $\geq m$, is the empty set. This arbitrary choice is justified since $\mathcal{A}_v$ ignores these moves. Similar to the first direction, it is not too hard to see that if σ' enforces *shift*$(L(\mathcal{A}))$ then σ enforces $L(\mathcal{A})$.

The proof of Theorem 8 is complete by observing that the construction of $\mathcal{A}_v$, as well as the transformations of strategies given above, are clearly performed in PTIME (recall that the multi-agent setting — and thus the set $\mathbf{Z}$ of atoms — is considered to be constant, and not part of the input).

6.2 Proof of Theorem 5

The next theorem shows how to convert a DELA $\mathcal{A}$ into one for the traces consistent with strategies enforcing $L(\mathcal{A})$; its proof follows the same line of reasoning as the one stated in (Aminof et al. 2025a, Theorem 8).

Theorem 12. *Given a DELA $\mathcal{A} = (D, (\Gamma, \lambda, B))$ and an agent v, a DELA $\mathcal{A}'$ for Cns(Enf$_v$($L(\mathcal{A})$)) can be built in time polynomial in the size of the transition system D, exponential in the number $|\Gamma|$ of labels, and polynomial in the size of the Boolean formula B. Moreover, the size of $\mathcal{A}'$ is linear in the size of $\mathcal{A}$.*

We can now give the construction of $\mathcal{A}_{\alpha_v}$ stated in Theorem 5. Given a strategies expression α_v, the construction proceeds in three steps: in Step (A) we construct an expression that describes a trace property $\Psi(\alpha_v)$ such that a v-strategy is in α_v iff it enforces $\Psi(\alpha_v)$; in Step (B), by traversing the parse-tree for the expression $\Psi(\alpha_v)$, we build a DELA $\mathcal{A}_{\Psi(\alpha_v)}$ whose language is $\Psi(\alpha_v)$; and in Step (C) we obtain from $\mathcal{A}_{\Psi(\alpha_v)}$ the desired DELA $\mathcal{A}_{\alpha_v}$. We call the expressions constructed in Step (A) *trace-properties expressions*. They define trace properties using LTL formulas (where a formula represents the set of traces that satisfy it), standard Boolean set operations like $\wedge$ (intersection), $\neg$ (complement), etc., and the operator Cns(Enf$_v(-)$) which

takes a trace property $\mathcal{P}$ and returns the set of traces consistent with v-strategies that enforce $\mathcal{P}$.

Step (A) Let α_w be a strategies expression for agent w. Build a trace-properties expression $\Psi(\alpha_w)$ by induction on the structure of α_w, as follows:

- Suppose $\alpha_w = \text{Enforce}_w(\varphi)$ where φ is an LTL formula. Define $\Psi(\alpha_w)$ to be φ.
- Suppose $\alpha_w = \text{Enforce}_w(\varphi, \Theta_w)$ where φ is an LTL formula and $\Theta_w = (\alpha_1, \cdots \alpha_{w-1}, \alpha_{w+1}, \cdots, \alpha_N)$. Define $\Psi(\alpha_w)$ to be $(\wedge_{x \neq w} \text{Cns}(\text{Enf}_x(\Psi(\alpha_x)))) \Rightarrow \varphi$.

The fact that a w-strategy is in α_w iff it enforces $\Psi(\alpha_w)$ follows by induction on the structure of α_w: it is trivially true for $\alpha_w = \text{Enforce}_w(\varphi)$, and follows from Lemma 4 for $\alpha_w = \text{Enforce}_w(\varphi, \Theta_w)$, and the fact that $\text{Enf}_x(\Psi(\alpha_x)) = \alpha_x$ (for all $x \neq w$), and that $\wedge_{x \neq w} \text{Cns}(\alpha_x) = \text{Cns}(\Theta_w)$. The output of this step is the trace-properties expression $\Psi(\alpha_v)$.

Step (B) Given a trace-properties expression Ψ we build (by induction on the structure of the expression Ψ) a DELA $\mathcal{A}_\Psi$ whose language is the traces defined by Ψ: if Ψ is an LTL formula φ then apply Theorem 3; if Ψ is a Boolean combination, apply the fact that DELAs are closed under Boolean operations (Lemma 1); and if Ψ is of the form $\text{Cns}(\text{Enf}_x(\Psi'))$, apply Theorem 12 to the DELA $\mathcal{A}_{\Psi'}$. The output of this step is the DELA $\mathcal{A}_{\Psi(\alpha_v)}$.

Step (C) Finally, obtain the DELA $\mathcal{A}_{\alpha_v}$ by applying Theorem 12 to the DELA $\mathcal{A}_{\Psi(\alpha_v)}$.

Proof of Theorem 5. By Step (C) the language of $\mathcal{A}_{\alpha_v}$ is the set of traces consistent with strategies that enforce $L(\mathcal{A}_{\Psi(\alpha_v)})$; by Step (B), language $L(\mathcal{A}_{\Psi(\alpha_v)})$ equals the trace property $\Psi(\alpha_v)$; and by Step (A) the strategies enforcing this property are exactly the strategies in α_v. Hence, overall $L(\mathcal{A}_{\alpha_v})$ is the set of all traces consistent with strategies in α_v. This proves the first part of Theorem 5. For the size and complexity analysis, proceed as follows.

For step (A), the size of $\Psi = \Psi(\alpha_v)$ is linear in the size of α_v, and can be computed from α_v in linear time. To calculate the cost of step (B), we first calculate the size of the DELA constructed for each node in the parse-tree of Ψ. Let z be the size of the largest input LTL formula appearing in Ψ. By Theorem 3, for each leaf, the associated DELA has at most 2EXP in z many states, and an acceptance condition whose size (both in terms of the number of labels and in terms of the size of the Boolean formula) is 1EXP in z. An internal node constructed using Theorem 12 has a DELA whose size is essentially the same as that of its child node. By Lemma 1, an internal node representing a Boolean operation has a DELA whose number of states is a product of the number of states of its $N-1$ child DELAs, and whose number of labels (resp. size of Boolean formula) is linear in the sum of the number of labels (resp. in the sizes of the Boolean formulas) of its child DELAs. Overall, we get that the number of states of any constructed DELA in Step (B) is at most the number of states of the biggest leaf DELA raised to the power $2^{O(H)}$, where H is the depth of the parse tree (observe that the multi-agent setting is considered fixed, and thus the number of agents N is a constant). Since H is bounded by the size of α_v, each constructed DELA (and in particular $\mathcal{A}_{\Psi(\alpha_v)}$) has at most 2EXP many states, 1EXP many labels, and a Boolean formula of size at most 1EXP in the size of the input.

As for the cost of Step (C), observe that the DELA $\mathcal{A}_{\alpha_v}$ was constructed using Theorem 12, similar to DELAs for sub-expressions of the form $\text{Cns}(\text{Enf}_x(\Psi'))$ constructed in Step (B), and thus the cost calculation of this step is essentially covered by the calculation there.

It is easy to see that for leaf nodes, as well as for Boolean operations nodes, the time spent in step (B) constructing the corresponding DELA is linearly related to the size of this DELA. By the upper-bound for the sizes of the DELAs given above, we can deduce that the time needed, in Steps (B) or (C), to construct a DELA using Theorem 12 is at most 2EXP in the size of the input. Hence, since the size of the parse tree of Ψ is polynomial in the input size, the total time spent in step (B) is 2EXP in the input size. Hence, by combining the costs of steps (A), (B), and (C), and using the stated complexity for solving EL-games (Proposition 7), one gets the required 2EXPTIME upper-bound. □

7 Discussion

Our strategy expressions (Section 4) are non-recursive. We now discuss recursive relationships between strategy expressions, as well as the semantic problems that arise in this case.

We can assign variables to represent strategy expressions, and use these variables inside the assumptions. For instance: $x_1 = \text{Enforce}_1(\varphi_1, (x_2))$ and $x_2 = \text{Enforce}_2(\varphi_2, (x_1))$. Here, x_i is a variable that varies over sets of strategies of agent i. This system of equations naturally captures the scenario that two agents know each others' goals.

What does it mean to solve these equations? The obvious semantics is that a solution is a pair (S_1, S_2) where S_i is a set of i-strategies, such that plugging in S_i into the RHSs yields the LHSs. In other words, (S_1, S_2) is a fixpoint of the induced operator $F(X_1, X_2) := (\text{Enf}_1(\varphi_1, X_2), \text{Enf}_2(\varphi_2, X_1))$. Unfortunately, such operators do not (in general) satisfy the common monotonocity requirement for fixpoints to actually exist. Indeed, if one *increases* X_1, X_2, and then re-evaluates the equations, one obtains *smaller* X_1, X_2, because increasing an assumption potentially makes it harder to enforce the goal under that assumption see (Remark 1). Notably, if one tries to compute a greatest (or least) fixpoint, the sequence of steps $(F^n(\Xi^1, \Xi^2))_{n \geq 0}$ may oscillate and never fix.[2]

Finally, ignoring the non-trivial problem of computing a fixpoint (X_1, X_2) if one happens to exist, what is the appeal of such a fixpoint? In our example, if an agent can enforce its goal unilaterally, then such a fixpoint is not very appealing, it may actually be dangerous. Indeed, such a fixpoint may contain strategies that do not enforce its goal unilater-

[2] One way to avoid the oscillation problem is to intersect, i.e., consider $G(X_1, X_2) := (\text{Enf}_1(\varphi_1, X_2) \cap X_1, \text{Enf}_2(\varphi_2, X_1) \cap X_2)$. However: if agent i cannot enforce φ_i unilaterally, then the ith component of $G^n(\Xi^1, \Xi^2)$ is $\emptyset$ for $n \geq 1$. On the other hand, if both can enforce unilaterally, then $(\text{Enf}_1(\varphi_1), \text{Enf}_2(\varphi_2))$ is a fixpoint, and the agents gain nothing by having an assumption!

ally, and employing such a strategy is risky if the other agent deviates from the fix point. Otherwise, in the special case that neither agent can enforce its goal unilaterally, both are incentivized to stick to that fixpoint if the other does, since in this case they each achieve their goal. This is reminiscent of Nash Equilibria in game-theory, and it suffers from the same issues. There may be multiple fixpoints, so how do the agents agree on one? And, if they do have a way to agree, why don't they agree on a single co-operative strategy profile, i.e., on a Nash Equilibrium?

In summary, while the recursive case is not as appealing as it seems at first sight, it may deserve further investigation to explore scenarios where it may make sense.

AI Declaration

Generative AI was not used in any part of the research or writing of this paper.

Acknowledgements

This research was funded in part by the Austrian Science Fund (FWF) PAT2028925, by the EPSRC grant EP/Y028872/1, Mathematical Foundations of Intelligence: An "Erlangen Programme" for AI, and by PRIN 2020 RIPER - CUP E63C22000400001 and MASE GameL - CUP F53C25001420001.

References

Aminof, B.; De Giacomo, G.; Murano, A.; Rubin, S.; et al. 2018. Synthesis under assumptions. In *KR*, 615–616.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019. Planning under LTL Environment Specifications. In *ICAPS*, 31–39.

Aminof, B.; De Giacomo, G.; Rubin, S.; and Zuleger, F. 2024. Proper Linear-time Specifications of Environment Behaviors in Nondeterministic Planning and Reactive Synthesis. In *KR*.

Aminof, B.; De Giacomo, G.; Perelli, G.; and Rubin, S. 2025a. LTL synthesis under multi-agent environment assumptions. In *KR*.

Aminof, B.; De Giacomo, G.; Rubin, S.; and Vardi, M. 2025b. LTLf+ and PPLTL+: Extending LTLf and PPLTL to Infinite Traces. In *IJCAI*.

Brenguier, R.; Raskin, J.; and Sankur, O. 2017. Assume-Admissible Synthesis. *Acta Informatica* 54(1):41–83.

Camacho, A.; Bienvenu, M.; and McIlraith, S. 2018. Finite LTL Synthesis with Environment Assumptions and Quality Measures. In *KR*.

Camacho, A.; Bienvenu, M.; and McIlraith, S. 2019. Towards a Unified View of AI Planning and Reactive Synthesis. In *ICAPS*.

Cermák, P.; Lomuscio, A.; and Murano, A. 2015. Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. In *AAAI*, 2038–2044.

Church, A. 1963. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, 23–35.

De Giacomo, G., and Rubin, S. 2018. Automata-Theoretic Foundations of FOND Planning for LTLf and LDLf Goals. In *IJCAI*.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. In *IJCAI*, 1558–1564.

Ehlers, R.; Lafortune, S.; Tripakis, S.; and Vardi, M. Y. 2017. Supervisory control and reactive synthesis: a comparative introduction. *Discret. Event Dyn. Syst.* 27(2):209–260.

Emerson, E. A., and Lei, C. 1987. Modalities for Model Checking: Branching Time Logic Strikes Back. *Sci. Comput. Program.* 8(3):275–306.

Fijalkow, N.; Maubert, B.; Murano, A.; and Vardi, M. Y. 2020. Assume-guarantee synthesis for prompt linear temporal logic. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 117–123. ijcai.org.

Finkbeiner, B. 2016. Synthesis of reactive systems. *Dependable Softw. Syst. Eng.* 45:72–98.

Fisman, D.; Kupferman, O.; and Lustig, Y. 2010. Rational Synthesis. In *TACAS*.

Gardy, P.; Bouyer, P.; and Markey, N. 2020. Dependences in strategy logic. *Theory Comput. Syst.* 64(3):467–507.

Geatti, L.; Montali, M.; and Rivkin, A. 2024. Foundations of reactive synthesis for declarative process specifications. In *AAAI*.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Harel, D., and Pnueli, A. 1984. On the development of reactive systems. In *Logics and models of concurrent systems*. Springer. 477–498.

Hausmann, D.; Lehaut, M.; and Piterman, N. 2024. Symbolic Solution of Emerson-Lei Games for Reactive Synthesis. In *FoSSaCS*.

Mogavero, F.; Murano, A.; Perelli, G.; and Vardi, M. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM TOCL* 15(4):34:1–34:47.

Pnueli, A., and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *POPL*, 179–190. ACM.

Pnueli, A., and Rosner, R. 1990. Distributed Reactive Systems Are Hard to Synthesize. In *FOCS*.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT.

Safra, S. 1988. On the Complexity of omega-Automata. In *FOCS*.

Vardi, M. Y., and Wolper, P. 1994. Reasoning About Infinite Computations. *Inf. Comput.* 115(1):1–37.