Compositional Synthesis of Reactive Controllers for Multi-Agent Systems[⋆]

Rajeev Alur, Salar Moarref, and Ufuk Topcu

`alur@seas.upenn.edu, moarref@seas.upenn.edu, utopcu@utexas.edu`

Abstract. In this paper we consider the controller synthesis problem for multi-agent systems that consist of a set of controlled and uncontrolled agents. Controlled agents may need to cooperate with each other and react to the actions of uncontrolled agents in order to fulfill their objectives. Besides, the controlled agents may be imperfect, i.e., only partially observe their environment, for example due to the limitations in their sensors. We propose a framework for controller synthesis based on compositional reactive synthesis. We implement the algorithms symbolically and apply them to a robot motion planning case study where multiple robots are placed on a grid-world with static obstacles and other dynamic, uncontrolled and potentially adversarial robots. We consider different objectives such as collision avoidance, keeping a formation and bounded reachability. We show that by taking advantage of the structure of the system, compositional synthesis algorithm can significantly outperform centralized synthesis approach, both from time and memory perspective, and can solve problems where the centralized algorithm is infeasible. Our findings show the potential of symbolic and compositional reactive synthesis methods as planning algorithms in the presence of dynamically changing and possibly adversarial environment.

1 Introduction

Complex systems often consist of multiple agents (or components) interacting with each other and their environment to achieve certain objectives. For example, teams of robots are employed to perform tasks such as monitoring, surveillance, and disaster response in different domains including search and rescue [1], object transportation [2], and formation control [3]. With growing complexity of autonomous systems and their safety-critical nature, the need for *automated* and *reliable* design and analysis methods and tools is increasing. To this end, an ambitious goal in system design and control is to automatically synthesize controllers for controllable parts of the system that guarantee the satisfaction of the specified objectives. Given a model of the system describing the interaction of a controllable plant with its environment and an objective in a formal language such as linear temporal logic (LTL), controller synthesis problem seeks

⋆ This research was partially supported by awards NSF Expeditions in Computing CCF 1138996, AFRL FA8650-15-C-2546, ONR N000141310778, ARO W911NF-15-1-0592, NSF 1550212 and DARPA $W$911$NF$-16-1-0001.

to construct a finite-state controller that ensures that the system satisfies the objective, regardless of how its environment behaves. In this paper we consider the controller synthesis problem for multi-agent systems.

One of the main challenges in automated synthesis of systems is the scalability problem. This issue becomes more evident for multi-agent systems, as adding each agent can often increase the size of the state space exponentially. The pioneering work by Pnueli et. al [4] showed that reactive synthesis from LTL specifications is intractable which prohibited the practitioners from utilizing synthesis algorithms in practice. Distributed reactive synthesis [5] and multi-player games of incomplete information [6] are undecidable in general. Despite these discouraging results, recent advances in this growing research area have enabled automatic synthesis of interesting real-world systems [7], indicating the potential of the synthesis algorithms for solving realistic problems. The key insight is to consider more restricted yet practically useful subclasses of the general problem, and in this paper we take a step toward this direction.

The main motivation for our work is the growing interest in robotic motion planning from rich high-level specifications, e.g., LTL [8,9,10]. In most of these works, all agents are controlled and operate in static and fully-observable environments, and the applications of synthesis algorithms are restricted to very small examples due to the well-known state explosion problem. Since the reactive synthesis from LTL specifications is intractable, no algorithm will be efficient for all problems. Nevertheless, one can observe that in many application domains such as robot motion planning, the systems are structured, a fact that can be exploited to achieve better scalability.

In this paper, we consider a special class of multi-agent systems that are referred to as *decoupled* and are inspired by robot motion planning, decentralized control [11,12], and swarm robotics [13,14] literature. Intuitively, in a decoupled multi-agent system the transition relations (or dynamics) of the agents are decoupled, i.e., at any time-step, agents can make decisions on what action to take based on their own local state. For example, an autonomous vehicle can decide to slow down or speed up based on its position, velocity, etc. However, decoupled agents are coupled through objectives, i.e., an agent may need to cooperate with other agents or react to their actions to fulfill a given objective (e.g., it would not be a wise decision for an autonomous vehicle to speed up when the front vehicle pushes the break if collision avoidance is an objective.) In our framework, multi-agent systems consist of a set of controlled and uncontrolled agents. Controlled agents may need to cooperate with each other and react to the actions of uncontrolled agents in order to fulfill their objectives. Besides, controlled agents may be imperfect in the sense that they can only partially observe their environment, for example due to the limitations in their sensors. The goal is to synthesize controllers for each controlled agent such that the objectives are enforced in the resulting system.

To solve the controller synthesis problem for multi-agent systems one can directly construct the model of the system by composing those of the agents, and solve the problem centrally for the given objectives. However, the centralized method lack flexibility, since any change in one of the components requires the repetition of the synthesis process for the whole system. Besides the resulting system might be exponentially larger than the individual parts, making this approach infeasible in practice. Compositional reactive synthesis aims to exploit the structure of the system by breaking the problem into smaller and more manageable pieces and solving them separately. Then solutions to sub-problems are merged and analyzed to find a solution for the whole problem. The existing structure in multi-agent systems makes them a potential application area for compositional synthesis techniques.

To this end, we propose a compositional framework for decoupled multi-agent systems based on automatic decomposition of objectives and compositional reactive synthesis using maximally permissive strategies [15]. We assume that the objective of the system is given in conjunctive form. We make an observation that in many cases, each conjunct of the global objective only refers to a small subset of agents in the system. We take advantage of this structure to decompose the synthesis problem: for each conjunct of the global objective, we only consider the agents that are involved, and compute the maximally permissive strategies for those agents with respect to the considered conjunct. We then intersect the strategies to remove potential conflicts between them, and project back the constraints to subproblems, solving them again with updated constraints, and repeating this process until the strategies become fixed.

We implement the algorithms symbolically using binary decision diagrams (BDDs) and apply them to a robot motion planning case study where multiple robots are placed on a grid-world with static obstacles and other dynamic, uncontrolled and potentially adversarial robots. We consider different objectives such as collision avoidance, keeping a formation and bounded reachability. We show that by taking advantage of the structure of the system, the proposed compositional synthesis algorithm can significantly outperform the centralized synthesis approach, both from time and memory perspective, and can solve problems where the centralized algorithm is infeasible. Furthermore, using compositional algorithms we managed to solve synthesis problems for systems with multiple agents, more complex objectives and for grid-worlds of sizes that are much larger than the cases considered in similar works. Our findings show the potential of symbolic and compositional reactive synthesis methods as planning algorithms in presence of dynamically changing and possibly adversarial environment.

Contributions. The main contributions of the paper are as follow. We propose a framework for modular specification and compositional controller synthesis for multi-agent systems with imperfect controlled agents. We implement the methods symbolically using BDDs and apply them to a robot motion planning case study. We report on our experimental results and show that the compositional algorithm can significantly outperform the centralized approach.

Related Work. Compositional reactive synthesis has been considered in some recent works. Kupferman et al. [16] propose a compositional algorithm for LTL realizability and synthesis based on a Safraless approach that transforms the synthesis problem into a Büchi game. Baier et al. [17] give a compositional framework for treating multiple linear-time objectives inductively. Sohail et al. [18] propose an algorithm to compositionally construct a parity game from conjunctive LTL specifications. Alur et al. [19] show how local specifications of components can be refined compositionally to ensure satisfaction of a global specification. Lustig et al. [20] study the problem of LTL synthesis from libraries of reusable components. Alur et al. [21] propose a framework for compositional synthesis from a library of parametric and reactive controllers. Filiot et al. [15] reduce the LTL realizability problem to solving safety games. They show that, for LTL specifications written as conjunction of smaller LTL formulas, the problem can be solved compositionally by first computing winning strategies for each conjunct. Moreover, they show that compositional algorithms can handle fairly large LTL specifications. To the best of our knowledge, algorithms in [15] seems to be the most successful application of compositional synthesis in practice.

Two-player games of imperfect information are studied in [22,23,24,25], and it is shown that they are often more complicated than games of perfect information. The algorithmic difference is exponential, due to a subset construction that turns a game of imperfect information into an equivalent game of perfect information. In this paper, we build on the results of [15,25] and extend and adapt their methods to treat multi-agent systems with imperfect agents. To the best of our knowledge, compositional reactive synthesis is not studied in the context of multi-agent systems and robot motion planning.

The controller synthesis problem for systems with multiple controllable agents from a high-level temporal logic specification is also considered in many recent works (e.g., [8,26,27]). A common theme is based on first computing a discrete controller satisfying the LTL specification over a discrete abstraction of the system, which is then used to synthesize continues controllers guaranteed to fulfill the high-level specification. In many of these works (e.g., [28,29]) the agents' models are composed (either from the beginning or incrementally) to obtain a central model. The product of the central model with the specification automaton is then constructed and analyzed to compute a strategy. In [9], authors present a compositional motion planning framework for multi-robot systems based on a reduction to satisfiability modulo theories. However, their model cannot handle uncertain or dynamic environment. In [8,30] it is proposed that systems with multiple components can be treated in a decentralized manner by considering one component as a part of the environment of another component. However, these approaches cannot address the need for joint decision making and cooperative objectives. In this paper we consider compositional and symbolic algorithms for solving games in presence of a dynamic and possibly adversarial environment.

2 Preliminaries

Linear temporal logic (LTL). We use LTL to specify system objectives. LTL is a formal specification language with two types of operators: logical connectives (e.g., $\neg$ (negation) and $\wedge$ (conjunction)) and temporal operators (e.g., $\bigcirc$ (next), $\mathcal{U}$ (until), $\Diamond$ (eventually), and $\Box$ (always)). Let $\mathcal{V}$ be a finite set of Boolean variables. A formula with no temporal operator is a Boolean formula or a *predicate*. Given a predicate ϕ over variables $\mathcal{V}$, we say $s \in 2^{\mathcal{V}}$ satisfies ϕ, denoted by $s \models \phi$, if the formula obtained from ϕ by replacing all variables in s by `true` and all other variables by `false` is valid. We call the set of all possible assignments to variables $\mathcal{V}$ *states* and denote them by $\Sigma_{\mathcal{V}}$, i.e., $\Sigma_{\mathcal{V}} = 2^{\mathcal{V}}$. An LTL formula over variables $\mathcal{V}$ is interpreted over infinite words $w \in (\Sigma_{\mathcal{V}})^{\omega}$. The language of an LTL formula Φ, denoted by $\mathcal{L}(\Phi)$, is the set of infinite words that satisfy Φ, i.e., $\mathcal{L}(\Phi) = \{w \in (\Sigma_{\mathcal{V}})^{\omega} \mid w \models \Phi\}$. We assume some familiarity of the reader with LTL. We often use predicates over $\mathcal{V} \cup \mathcal{V}'$ where $\mathcal{V}'$ is the set of primed versions of the variables in $\mathcal{V}$. Given a subset of variables $\mathcal{X} \subseteq \mathcal{V}$ and a state $s \in \Sigma_{\mathcal{V}}$, we denote by $s_{|\mathcal{X}}$ the projection of s to $\mathcal{X}$. For a set $\mathcal{Z} \subseteq \mathcal{V}$, let $Same(\mathcal{Z}, \mathcal{Z}')$ be a predicate specifying that the value of the variables in $\mathcal{Z}$ stay unchanged during a transition. Ordered binary decision diagrams (OBDDs) can be used for obtaining concise representations of sets and relations over finite domains [31]. If R is an n-ary relation over $\{0, 1\}$, then R can be represented by the BDD for its *characteristic function*: $f_R(x_1, \cdots, x_n) = 1$ if and only if $R(x_1, \cdots, x_n) = 1$. With a little bit abuse of notation and when it is clear from the context, we treat sets and functions as their corresponding predicates.

Game structures. A game structure $\mathcal{G}$ of imperfect information is a tuple $\mathcal{G} = (\mathcal{V}, \Lambda, \tau, \mathcal{OBS}, \gamma)$ where $\mathcal{V}$ is a finite set of variables, Λ is a finite set of actions, τ is a predicate over $\mathcal{V} \cup \Lambda \cup \mathcal{V}'$ defining $\mathcal{G}$'s transition relation, $\mathcal{OBS}$ is a finite set of observable variables, and $\gamma : \Sigma_{\mathcal{OBS}} \to 2^{\Sigma_{\mathcal{V}}} \setminus \emptyset$ maps each observation to its corresponding set of states. We assume that the set $\{\gamma(o) \mid o \in \Sigma_{\mathcal{OBS}}\}$ partitions the state space $\Sigma_{\mathcal{V}}$ (this assumption can be weakened to a covering of the state space where observations can overlap [25,24].) A game structure $\mathcal{G}$ is called *perfect information* if $\mathcal{OBS} = \mathcal{V}$ and $\gamma(s) = \{s\}$ for all $s \in \Sigma_{\mathcal{V}}$. We omit $(\mathcal{OBS}, \gamma)$ in the description of games of perfect information.

In this paper, we consider two-player turn-based game structures where player-1 and player-2 alternate in taking turns. Let $t \in \mathcal{V}$ be a special variable with domain $\{1, 2\}$ determining which player's turn it is during the game. Without loss of generality, we assume that player-1 always start the game. Let $\Sigma^i_{\mathcal{V}} = \left\{s \in \Sigma_{\mathcal{V}} \mid s_{|t} = i\right\}$ for $i = 1, 2$ denote player-i's states in the game structure. At any state $s \in \Sigma^i_{\mathcal{V}}$, the player-$i$ chooses an action $\ell \in \Lambda$ such that there exists a successor state $s' \in \Sigma_{\mathcal{V}'}$ where $(s, \ell, s') \models \tau$. Intuitively, at a player-i state, she chooses an available action according to the transition relation τ and the next state of the system is chosen from the possible successor states. For every state $s \in \Sigma_{\mathcal{V}}$, we define $\Gamma(s) = \{\ell \in \Lambda \mid \exists s' \in \Sigma_{\mathcal{V}'}.\ (s, \ell, s') \models \tau\}$ to be the set of available actions at that state. A *run* in $\mathcal{G}$ from an initial state $s_{init} \in \Sigma_{\mathcal{V}}$ is a sequence of states $\pi = s_0 s_1 s_2 \cdots$ such that $s_0 = s_{init}$ and for all $i > 0$, there is an action $\ell_i \in \Lambda$ with $(s_{i-1}, \ell_i, s'_i) \models \tau$, where s'_i is obtained by replacing the variables in s_i by their primed copies. A run π is maximal if either it is infinite or it ends in a state $s \in \Sigma_{\mathcal{V}}$ where $\Gamma(s) = \emptyset$. The *observation sequence* of π is the unique sequence $Obs(\pi) = o_0 o_1 o_2 \cdots$ such that for all $i \geq 0$, we have $s_i \in \gamma(o_i)$.

Strategies. A *strategy* $\mathtt{S}$ in $\mathcal{G}$ for player-i, $i \in \{1, 2\}$, is a function $\mathtt{S} : (\Sigma_{\mathcal{V}})^* . \Sigma^i_{\mathcal{V}} \to \Lambda$. A strategy $\mathtt{S}$ in $\mathcal{G}$ for player-2 is *observation-based* if for all prefixes $\rho_1, \rho_2 \in (\Sigma_{\mathcal{V}})^*.\Sigma_{\mathcal{V}}^2$, if $Obs(\rho_1) = Obs(\rho_2)$, then $\mathtt{S}(\rho_1) = \mathtt{S}(\rho_2)$. In this paper, we are interested in the existence of observation-based strategies for player-2. Given two strategies $\mathtt{S}_1$ and $\mathtt{S}_2$ for player-1 and player-2, the *possible outcomes* $\Omega_{\mathtt{S}_1,\mathtt{S}_2}(s)$ from a state $s \in \Sigma_{\mathcal{V}}$ are runs: a run $s_0 s_1 s_2 \cdots$ belongs to $\Omega_{\mathtt{S}_1,\mathtt{S}_2}(s)$ if and only if $s_0 = s$ and for all $j \geq 0$ either s_j has no successor, or $s_j \in \Sigma_{\mathcal{V}}^i$ and $(s_j, \mathtt{S}_i(s_0 \cdots s_j), s'_{j+1}) \models \tau$ where $s_j \in \Sigma_{\mathcal{V}}^i$.

Winning condition. A game $(\mathcal{G}, \phi_{init}, \Phi)$ consists of a game structure $\mathcal{G}$, a predicate ϕ_{init} specifying an initial state $s_{init} \in \Sigma_{\mathcal{V}}$, and an LTL objective Φ for player-2. A run $\pi = s_0 s_1 \cdots$ is winning for player-2 if it is infinite and $\pi \in \mathcal{L}(\Phi)$. Let Π be the set of runs that are winning for player-2. A strategy $\mathtt{S}_2$ is winning for player-2 if for all strategies $\mathtt{S}_1$ of player-1, we have $\Omega_{\mathtt{S}_1,\mathtt{S}_2}(s_{init}) \subseteq \Pi$, that is, all possible outcomes are winning for player-2. Note that We assume the nondeterminism is always on player-1's side. We say the game $(\mathcal{G}, \phi_{init}, \Phi)$ is realizable if and only if the system has a winning strategy in the game $(\mathcal{G}, \phi_{init}, \Phi)$.

Constructing the knowledge game structure. For a game structure $\mathcal{G} = (\mathcal{V}, \Lambda, \tau, \mathcal{OBS}, \gamma)$ of imperfect information, a game structure $\mathcal{G}^K$ of perfect information can be obtained using a subset construction procedure such that for any objective Φ, there exists a deterministic observation-based strategy for player-2 in $\mathcal{G}$ with respect to Φ if and only if there exists a deterministic winning strategy for player-2 in $\mathcal{G}^K$ for Φ [22,25]. Intuitively, each state in $\mathcal{G}^K$ is a set of states of $\mathcal{G}$ that represents player-2's knowledge about the possible states in which the game can be after a sequence of observations. In the worst case, the size of $\mathcal{G}^K$ is exponentially larger than the size of $\mathcal{G}$. We refer to $\mathcal{G}^K$ as the *knowledge* game structure corresponding to $\mathcal{G}$. In the rest of this section, we only consider game structures of perfect information.

Solving games. In this paper, we use the bounded synthesis approach [32,15] to solve the synthesis problems from LTL specifications. In [15], it is shown how LTL formulas can be reduced to *safety games*. Formally, a safety game is a game $(\mathcal{G}, \phi_{init}, \Phi)$ with a special safety objective $\Phi = \Box(\mathtt{True})$. That is, any infinite run in the game structure $\mathcal{G}$ starting from an initial state $s \models \phi_{init}$ is winning for player-2. We drop Φ from description of safety games as it is implicitly defined. Intuitively, in a safety game, the goal of player-2 is to avoid the dead-end states, i.e., states that there is no available action. We refer the readers to [15,33] for details of reducing LTL formulas to safety games and solving them. Composition of two game structures $\mathcal{G}_1 = (\mathcal{V}^1, \Lambda^1, \tau^1), \mathcal{G}_2 = (\mathcal{V}^2, \Lambda^2, \tau^2)$ of perfect information, denoted by $\mathcal{G}^\otimes = \mathcal{G}_1 \otimes \mathcal{G}_2$, is a game structure $\mathcal{G}^\otimes = (\mathcal{V}^\otimes, \Lambda^\otimes, \tau^\otimes)$ of perfect information where $\mathcal{V}^\otimes = \mathcal{V}^1 \cup \mathcal{V}^2$, $\Lambda^\otimes = \Lambda^1 \cup \Lambda^2$, and $\tau^\otimes = \tau^1 \wedge \tau^2$. To solve a game $(\mathcal{G}, \phi_{init}, \Phi)$, we first obtain the game structure $\mathcal{G}^\Phi$ corresponding to Φ using the methods proposed in [15], and then solve the safety game $(\mathcal{G} \otimes \mathcal{G}^\Phi, \phi_{init})$ to determine the winner of the game and compute a winning strategy for player-2, if one exists.

Maximally permissive strategies. Safety games are *memory-less determined*, i.e., player-2 wins the game if and only if there exists a strategy $\mathtt{S} : \Sigma_{\mathcal{V}}^2 \to \Lambda$. Intuitively, a memory-less strategy only depends on the current state and is independent from the history of the game. Let $(\mathcal{G}, \phi_{init})$ be a safety game, where

$\mathcal{G} = (\mathcal{V}, \Lambda, \tau)$ is a perfect information game. Assume $W \subseteq \Sigma_{\mathcal{V}}$ be the set of winning states for player-2, i.e., from any state $s \in W$ there exists a strategy $\mathtt{S}_2$ such that for any strategy $\mathtt{S}_1$ chosen by player-1, all possible outcomes $\pi \in \Omega_{\mathtt{S}_1,\mathtt{S}_2}(s)$ are winning. The *maximally permissive strategy* $\mathcal{S} : \Sigma^2_{\mathcal{V}} \to 2^{\Lambda}$ for player-2 is defined as follows: for all $s \in \Sigma^2_{\mathcal{V}}$, $\mathcal{S}(s) = \{\ell \in \Lambda \mid \forall r \in \Sigma_{\mathcal{V}'}.\ (s, \ell, r) \models \tau_s \to r \in W\}$, i.e., the set of actions ℓ where all ℓ-successors belong to the set of winning states. It is well-known that $\mathcal{S}$ subsumes all winning strategies of player-2 in the safety game $(\mathcal{G}, \Phi_{init})$. Composition of two maximally permissive strategies $\mathcal{S}_1, \mathcal{S}_2 : \Sigma^2_{\mathcal{V}} \to 2^{\Lambda}$, denoted by $\mathcal{S} = \mathcal{S}_1 \otimes \mathcal{S}_2$, is defined as $\mathcal{S}(s) = \mathcal{S}_1(s) \cap \mathcal{S}_2(s)$ for any $s \in \Sigma_{\mathcal{V}}$, i.e., the set of allowed actions by $\mathcal{S}$ at any state $s \in \Sigma_{\mathcal{V}}$ is the intersection of the allowed actions by $\mathcal{S}_1$ and $\mathcal{S}_2$. The restriction of the game structure $\mathcal{G}$ with respect to its maximally permissive strategy $\mathcal{S}$ is the game structure $\mathcal{G}[\mathcal{S}] = (\mathcal{V}, \Lambda, \tau \wedge \phi_{\mathcal{S}})$ where $\phi_{\mathcal{S}}$ is the predicate encoding $\mathcal{S}$, i.e., for all $(s, \ell) \in \Sigma^2_{\mathcal{V}} \times \Lambda$, $(s, \ell) \models \phi_{\mathcal{S}}$ if and only if $\ell \in \mathcal{S}(s)$. Intuitively, $\mathcal{G}[\mathcal{S}]$ is the same as $\mathcal{G}$ but player-2's actions are restricted according to $\mathcal{S}$.

3 Multi-Agent Systems

In this section we describe how we model multi-agent systems and formally state the problem that is considered in the rest of the paper. Typically game structures arise from description of open systems in a particular language [34]. In our framework, we use *agents* to specify a system in a modular manner. An agent $\mathtt{a} = (\mathtt{type}, \mathcal{I}, \mathcal{O}, \Lambda, \tau, \mathcal{OBS}, \gamma)$ is a tuple where $\mathtt{type} \in \{\text{controlled}, \text{uncontrolled}\}$ indicates whether the agent can be controlled or not, $\mathcal{O}$ ($\mathcal{I}$) is a set of output (input) variables that the agent can (cannot, respectively) control by assigning values to them, Λ is a set of actions for the agent, and τ is a predicate over $\mathcal{I} \cup \mathcal{O} \cup \Lambda \cup \mathcal{O}'$ that specifies the possible transitions of the agent where $\mathcal{O}'$ is the primed copies of the variables $\mathcal{O}$, $\mathcal{OBS}$ is a set of observable variables, and $\gamma : \Sigma_{\mathcal{OBS}} \to 2^{\Sigma_{\mathcal{I} \cup \mathcal{O}}}$ is the observation function that maps agent's observations to its corresponding set of states. Intuitively, τ defines what actions an agent can choose at any state $s \in \Sigma_{\mathcal{I}} \times \Sigma_{\mathcal{O}}$ and what are the possible next valuations over agent's output variables for the chosen action. That is, $(i, o, \ell, o') \models \tau$ for $i \in \Sigma_{\mathcal{I}}$, $o \in \Sigma_{\mathcal{O}}$, $\ell \in \Lambda$, and $o' \in \Sigma_{\mathcal{O}'}$ means that at any state s of the system with $s_{|\mathcal{I}} = i$ and $s_{|\mathcal{O}} = o$, the agent can take action ℓ, and a state with component o' is a possible successor. A *perfect agent* is an agent with $\mathcal{OBS} = \mathcal{I} \cup \mathcal{O}$ and $\gamma(s) = \{s\}$ for all $s \in \Sigma_{\mathcal{I}} \times \Sigma_{\mathcal{O}}$. We omit $(\mathcal{OBS}, \gamma)$ in the description of perfect agents. An agent $\mathtt{a}$ is called *local* if and only if its transition relation τ is a predicate over $\mathcal{O} \cup \Lambda \cup \mathcal{O}'$, i.e., it does not depend on any uncontrolled variable $v \in \mathcal{I}$.

A multi-agent system $\mathcal{M} = \{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_n\}$ is defined as a set of agents $\mathtt{a}_i = (\mathtt{type}_i, \mathcal{I}_i, \mathcal{O}_i, \Lambda_i, \tau_i, \mathcal{OBS}_i, \gamma_i)$ for $1 \leq i \leq n$. Let $\mathcal{V} = \bigcup_{i=1}^{n} \mathcal{O}_i$ be the set of agents' output variables. We assume that the set of output variables of agents are pairwise disjoint, i.e., $\forall 1 \leq i \leq n.\ \mathcal{O}_i \cap \mathcal{O}_j = \emptyset$, and the set of input variables $\mathcal{I}_i$ for each agent $\mathtt{a}_i \in \mathcal{M}$ is a subset of variables controlled by other agents, i.e., $\mathcal{I}_i \subseteq \mathcal{V} \backslash \mathcal{O}_i$. We further make some simplifying assumptions. First, we assume that all uncontrolled agents are perfect, i.e., uncontrolled agent has perfect information about the state of the system at any time-step. Second, we assume that all controlled agents are *cooperative* while uncontrolled ones can play adversarially, i.e., the controlled agents cooperate with each other and make joint decisions to enforce the global objective. Finally, we assume that the observation variables for controlled agents are pairwise disjoint, i.e., $\forall 1 \leq i \leq n.\ \mathcal{OBS}_i \cap \mathcal{OBS}_j = \emptyset$, and that each controlled agent has perfect knowledge about other controlled agents' observations. That is, controlled agents share their observations with each other. Intuitively, it is as if the communication between controlled agents is instantaneous and error-free, i.e., they have perfect communication and tell each other what they observe. This assumption helps us preserve the two-player game setting and to stay in a decidable subclass of the more general problem of multi-player games with partial information. Note that multiplayer games of incomplete information are undecidable in general [6].

In this paper we focus on a special setting where all agents are local. A multi-agent system $\mathcal{M} = \{\mathsf{a}_1, \mathsf{a}_2, \cdots, \mathsf{a}_n\}$ is *dynamically decoupled* (or decoupled in short) iff all agents $\mathsf{a} \in \mathcal{M}$ are local. Intuitively, agents in a decoupled multi-agent system can choose their action based on their own local state and regardless of the local states of other agents in the system. That is, the availability of actions for each agent in any state of the system is only a function of the agent's local state. Such setting arises in many applications, e.g., robot motion planning, where possible transitions of agents are independent from each other. For example, how a robot moves around a room is usually based on its own characteristics and motion primitives [9]. Note that this does not mean that the controlled agents are completely decoupled, as the objectives might concern different agents in the system, e.g., collision avoidance objective for a system consisting of multiple controlled robots, which requires cooperation between agents.

In our framework, the user describes the agents and specifies the objective as a conjunctive LTL formula. From description of the agents, a game structure is obtained that encodes how the state of the system evolves. Formally, given a decoupled multi-agent system $\mathcal{M} = \mathcal{M}^{\mathsf{u}} \biguplus \mathcal{M}^{\mathsf{c}}$ partitioned into a set $\mathcal{M}^{\mathsf{u}} = \{\mathsf{u}_1, \cdots, \mathsf{u}_m\}$ of uncontrolled agents and a set $\mathcal{M}^{\mathsf{c}} = \{\mathsf{c}_1, \cdots, \mathsf{c}_n\}$ of controlled agents, the turn-based game structure $\mathcal{G}^{\mathcal{M}}$ induced by $\mathcal{M}$ is defined as $\mathcal{G}^{\mathcal{M}} = (\mathcal{V}, \mathit{\Lambda}, \tau, \mathcal{OBS}, \gamma)$ where $\mathcal{V} = \{t\} \cup \bigcup_{\mathsf{a} \in \mathcal{M}} \mathcal{O}_{\mathsf{a}}$ is the set of all variables in $\mathcal{M}$ with t as a turn variable, $\mathit{\Lambda} = \bigcup_{\mathsf{a} \in \mathcal{M}} \mathit{\Lambda}_{\mathsf{a}}$ is the set of actions, $\mathcal{OBS} = \bigcup_{\mathsf{c} \in \mathcal{M}^{\mathsf{c}}} \mathcal{OBS}_{\mathsf{c}}$ is the set of all observation variables of controlled agents (note that we assume all uncontrolled agents are perfect,) and τ and γ are defined as follows:

$$\begin{aligned}
\tau &= \tau_e \vee \tau_s \\
\tau_e &= t = 1 \wedge t' = 2 \wedge \bigwedge_{\mathsf{u} \in \mathcal{M}^{\mathsf{u}}} \tau_{\mathsf{u}} \wedge \bigwedge_{\mathsf{c} \in \mathcal{M}^{\mathsf{c}}} \mathit{Same}(\mathcal{O}_{\mathsf{c}}, \mathcal{O}'_{\mathsf{c}}) \\
\tau_s &= t = 2 \wedge t' = 1 \wedge \bigwedge_{\mathsf{c} \in \mathcal{M}^{\mathsf{c}}} \tau_{\mathsf{c}} \wedge \bigwedge_{\mathsf{u} \in \mathcal{M}^{\mathsf{u}}} \mathit{Same}(\mathcal{O}_{\mathsf{u}}, \mathcal{O}'_{\mathsf{u}}) \\
\gamma &= \bigwedge_{\mathsf{c} \in \mathcal{M}^{\mathsf{c}}} \gamma_{\mathsf{c}}
\end{aligned}$$

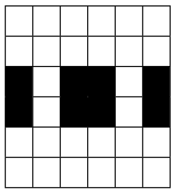

Fig. 1: Grid-world with static obstacles

Intuitively, at each step, uncontrolled agents take actions consistent with their transition relations, and their variables get updated while the controlled agents' variables stay unchanged. Then the controlled agents react concurrently and simultaneously by taking actions according to their transition relations, and their corresponding variables get updated while the uncontrolled agents' variables stay unchanged.

Example 1. Let R_1 and R_2 be two robots in an $n \times n$ grid-world similar to the one shown in Figure 1. Assume R_1 is an uncontrolled robot, whereas R_2 can be controlled. In the sequel, let i range over $\{1, 2\}$. At each time any robot R_i can move to one of its neighboring cells by taking an action from the set $\Lambda_i = \{up_i, down_i, right_i, left_i\}$. Furthermore, assume that R_2 has imperfect sensors and can only observe R_1 when R_1 is in one of its adjacent cells. Let (x_i, y_i) represent the position of robot R_i in the grid-world at any time[1]. We define $\mathcal{O}_i = \{x_i, y_i\}$ and $\mathcal{I}_i = \mathcal{O}_{3-i}$ as the output and input variables, respectively. Note that the controlled variables by one agent are the input variables of the other agent. Transition relation $\tau_i = \bigwedge_{\ell \in \Lambda_i} \tau_\ell$ is defined as conjunction of four parts corresponding to robot's action where

$$\begin{aligned}
\tau_{up_i} &= (y_i > 1) \wedge up_i \wedge (y'_i \leftrightarrow y_i - 1) \wedge Same(x_i, x'_i)\\
\tau_{down_i} &= (y_i < n) \wedge down_i \wedge (y'_i \leftrightarrow y_i + 1) \wedge Same(x_i, x'_i)\\
\tau_{left_i} &= (x_i > 1) \wedge left_i \wedge (x'_i \leftrightarrow x_i - 1) \wedge Same(y_i, y'_i)\\
\tau_{right_i} &= (x_i < n) \wedge right_i \wedge (x'_i \leftrightarrow x_i - 1) \wedge Same(y_i, y'_i)
\end{aligned}$$

Intuitively, each τ_ℓ for $\ell \in \Lambda_i$ specifies whether the action is available in the current state and what is its possible successors. For example, τ_{up_i} indicates that if R_i is not at the top row $(y_i > 1)$, then the action up_i is available and if applied, in the next state the value of y_i is decremented by one and the value of x_i does not change. Next we define the observation function γ_2 for R_2. It is easier and more intuitive to define γ_2^{-1}, and since observations partition the state space $\gamma_2 = (\gamma_2^{-1})^{-1}$ is defined. Formally,

$$\gamma_2^{-1}(a, b, c, d) = \begin{cases} (a, b, c, d) & \text{if } a - 1 \leq c \leq a + 1 \wedge b - 1 \leq d \leq b + 1 \\ (\bot, \bot, c, d) & \text{otherwise} \end{cases}$$

[1] Note that variables x_i and y_i are defined over a bounded domain and can be encoded by a set of Boolean variables. To keep the example simple, we use their bounded integer representation here.

Let $\mathcal{OBS}_2 = \{x_1^o, y_1^o, x_2^o, y_2^o\}$ where $x_1^o, y_1^o \in \{\bot, 1, 2, \cdots, n\}$ and $x_2^o, y_2^o \in \{1, \cdots, n\}$. Intuitively, R_2 observes its own local state perfectly. Furthermore, if R_1 is in one of its adjacent cells, its position is observed perfectly, otherwise, R_1 is away and its location cannot be observed. γ_2 can be symbolically encoded as $\bigvee_{o \in \Sigma_{\mathcal{OBS}}}(o \wedge \phi_{\gamma(o)})$ where $\phi_{\gamma(o)}$ is the predicate specifying the set $\gamma(o)$. Finally, we let $R_1 = (\text{uncontrolled}, \mathcal{I}_1, \mathcal{O}_1, \Lambda_1, \tau_1)$ and $R_2 = (\text{controlled}, \mathcal{I}_2, \mathcal{O}_2, \Lambda_2, \mathcal{OBS}_2, \gamma_2)$. Note that R_1 (R_2) is modeled as a perfect (imperfect, respectively) local agent.

The game structure $\mathcal{G}^{\mathcal{M}}$ of imperfect information corresponding to multi-agent system $\mathcal{M} = \{R_1, R_2\}$ is a tuple $\mathcal{G}^{\mathcal{M}} = (\mathcal{V}, \Lambda, \tau, \mathcal{OBS}, \gamma)$ where $\mathcal{V} = \{t\} \cup \mathcal{O}_1 \cup \mathcal{O}_2$, $\Lambda = \Lambda_1 \cup \Lambda_2$, $\tau = \tau_e \vee \tau_s$, $\tau_e = t = 1 \wedge t' = 2 \wedge \tau_1 \wedge Same(\mathcal{O}_2, \mathcal{O}_2')$, $\tau_s = t = 2 \wedge t' = 1 \wedge \tau_2 \wedge Same(\mathcal{O}_1, \mathcal{O}_1')$, $\mathcal{OBS} = \mathcal{OBS}_2$, and $\gamma = \gamma_2$. □

We now formally define the problem we consider in this paper.

Problem 1. Given a decoupled multi-agent system $\mathcal{M} = \mathcal{M}^{\mathtt{u}} \biguplus \mathcal{M}^{\mathtt{c}}$ partitioned into uncontrolled $\mathcal{M}^{\mathtt{u}} = \{\mathtt{u}_1, \cdots, \mathtt{u}_m\}$ and controlled agents $\mathcal{M}^{\mathtt{c}} = \{\mathtt{c}_1, \cdots, \mathtt{c}_n\}$, a predicate ϕ_{init} specifying an initial state, and an objective $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_k$ as conjunction of $k \geq 1$ LTL formulas Φ_i, compute strategies $\mathtt{S}_1, \cdots, \mathtt{S}_n$ for controlled agents such that the strategy $\mathtt{S} = \mathtt{S}_1 \otimes \cdots \otimes \mathtt{S}_n$ defined as composition of the strategies is winning for the game $(\mathcal{G}^{\mathcal{M}}, \phi_{init}, \Phi)$, where $\mathcal{G}^{\mathcal{M}}$ is the game structure induced by $\mathcal{M}$.

4 Compositional Controller Synthesis

We now explain our solution approach for Problem 1 stated in Section 3. Algorithm 1 summarizes the steps for compositional synthesis of strategies for controlled agents in a multi-agent system. It has three main parts. First the synthesis problem is automatically decomposed into subproblems by taking advantage of the structure in the multi-agent system and given objective. Then the subproblems are solved separately and their solutions are composed. The composition may restrict the possible actions that are available for agents at some states. The composition is then projected back to each subproblem and the subproblems are solved again with new restrictions. This process is repeated until either a subgame becomes unrealizable, or computed solutions for subproblems reach a fixed point. Finally, a set of strategies, one for each controlled agent, is extracted by decomposing the strategy obtained in the previous step. Next, we explain Algorithm 1 in more detail.

4.1 Decomposition of the Synthesis Problem

The synthesis problem is decomposed into subproblems in lines 2−9 of Algorithm 1. The main idea behind decomposition is that in many cases, each conjunct Φ_i of the objective Φ only refers to a small subset of agents. This observation is utilized to obtain a game structure from description of those agents that are *involved* in Φ_i, i.e., only agents are considered to form and solve a game with respect to Φ_i that are relevant. In other words, each subproblem corresponds to

Algorithm 1: Compositional Controller Synthesis

```
Input: A decoupled multi-agent system $\mathcal{M} = \{\mathtt{u}_1, \cdots, \mathtt{u}_m, \mathtt{c}_1, \cdots, \mathtt{c}_n\}$, $\phi_{init}$
       specifying initial state, and an objective $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_k$
Output: A set of strategies $(\mathcal{S}_1, \cdots, \mathcal{S}_n)$ one for each controlled agent, if one
        exists
1  /* Decompose the problem based on agents' involvement in conjuncts*/
2  for all $\Phi_i, 1 \leq i \leq k$ do
3      $\mathcal{INV}_i :=$ Involved($\Phi_i$);
4      $\mathcal{G}_i :=$ CreateGameStructure($\mathcal{INV}_i$);
5      $\mathcal{X}_i := \bigcup_{a \in \mathcal{INV}_i} \mathcal{O}_a$; /* the set of variables controlled by involved agents */
6      $\phi^i_{init} :=$ Project($\phi_{init}, \mathcal{X}_i$);
7      $\mathcal{G}^K_i :=$ CreateKnowledgeGameStructure($\mathcal{G}_i$);
8      $(\mathcal{G}^d_i, \phi^i_{init}) :=$ ToSafetyGame($\mathcal{G}^K_i, \phi^i_{init}, \Phi_i$);
9  /*Compositional synthesis*/
10 while true do
11     for $i = 1 \cdots k$ do
12         $\mathcal{S}^d_i :=$ SolveSafetyGame($\mathcal{G}^d_i, \phi^i_{init}$);
13     $\mathcal{S} := \bigotimes^m_{i=1} \mathcal{S}^d_i$; /* compose the strategies */
14     for $i = 1 \cdots k$ do
15         Let $\mathcal{Y}_i = \mathcal{V}^d_i \cup \Lambda^d_i$ be the set of variables and actions in $\mathcal{G}^d_i$;
16         $\mathcal{C}_i :=$ Project($\mathcal{S}, \mathcal{Y}_i$); /* project the strategies */
17     if $\forall 1 \leq i \leq k, \mathcal{S}^d_i = \mathcal{C}_i$ then
18         break; /* a fixed point is reached over strategies */
19     for $i = 1 \cdots k$ do
20         $\mathcal{G}^d_i := \mathcal{G}^d_i[\mathcal{C}_i]$; /* Restrict the subgames for the next iteration */
21 $(\mathtt{S}_1, \cdots, \mathtt{S}_n) :=$Extract($\mathcal{S}$);
22 return $(\mathtt{S}_1, \cdots, \mathtt{S}_n)$;
```

a conjunct Φ_i of the global objective Φ and the game structure obtained from agents involved in Φ_i.

For each conjunct Φ_i, $1 \leq i \leq k$, Algorithm 1 first obtains the set $\mathcal{INV}_i$ of involved agents using the procedure **Involved**. Formally, let $\mathcal{V}_{\Phi_i} \subseteq \mathcal{V}$ be the set of variables appearing in Φ_i's formula. The set of involved agents are those agents whose controlled variables appear in the conjunt's formula, i.e., **Involved**$(\Phi_i) = \{a \in \mathcal{M} \mid \mathcal{O}_a \cap \mathcal{V}_{\Phi_i} \neq \emptyset\}$.

A game structure $\mathcal{G}_i$ is then obtained from the description of the agents $\mathcal{INV}_i$ using the procedure **CreateGameStructure** as explained in Section 3. The projection ϕ^i_{init} of the predicate ϕ_{init} with respect to the involved agents is computed next. The procedure **Project** takes a predicate ϕ over variables $\mathcal{V}_\phi$ and a subset $\mathcal{X} \subseteq \mathcal{V}_\phi$ of variables as input, and projects the predicate with respect to the given subset. Formally, **Project**$(\phi, \mathcal{X}) = \left\{s_{|\mathcal{X}} \mid s \in \Sigma_{\mathcal{V}_\phi}\right\}$.

The knowledge game structure $\mathcal{G}^K_i$ corresponding to $\mathcal{G}_i$ is obtained at line 7. Note that this step is not required if the system only includes perfect agents that can observe the state of the game perfectly at any time-step. Finally, the objective Φ_i is transformed into a game structure using the algorithms in [15,33] and composed with $\mathcal{G}_i^K$ to obtain a safety game $(\mathcal{G}_i^d, \phi_{init}^i)$. The result of decomposition phase is k safety games $\big\{(\mathcal{G}_1^d, \phi_{init}^1), \cdots, (\mathcal{G}_k^d, \phi_{init}^k)\big\}$ that form the subproblems for the compositional synthesis phase.

Example 2. Let R_i for $i = 1, \cdots, 4$ be four robots in an $n \times n$ grid-world, where R_4 is uncontrolled and other robots are controlled. For simplicity, assume that all agents are perfect. At each time-step any robot R_i can move to one of its neighboring cells by taking an action from the set $\{up_i, down_i, right_i, left_i\}$ with their obvious meanings. Consider the following objective $\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \Phi_{12} \wedge \Phi_{23}$ where Φ_i for $i = 1, 2, 3$ specifies that R_i must not collide with R_4, and Φ_{12} (Φ_{23}) specifies that R_1 and R_2 (R_2 and R_3, respectively) must avoid collision with each other. Sub-formulas Φ_i, $i = 1, 2, 3$, only involve agents R_i and R_4, i.e., $\mathcal{INV}(\Phi_i) = \{R_i, R_4\}$. Therefore, the game structures $\mathcal{G}_i$ induced by agents R_i and R_4 are composed with the game structure computed for Φ_i to form a subproblem as a safety game. Similarly, we obtain safety games for objectives Φ_{12} and Φ_{23} with $\mathcal{INV}(\Phi_{12}) = \{R_1, R_2\}$ and $\mathcal{INV}(\Phi_{23}) = \{R_2, R_3\}$, respectively. $\square$

Remark 1. The decomposition method used here is not the only way to decompose the problem, neither it is necessarily optimal. More efficient decomposition technique can be used to obtain quicker convergence in Algorithm 1 for example by different grouping of conjuncts. Nevertheless, the decomposition technique explained above is simple and it was effective in our experiments.

4.2 Compositional Synthesis

The safety games obtained in decomposition phase are compositionally solved in lines $9-21$ of Algorithm 1. At each iteration of the main loop, the subproblems $(\mathcal{G}_i^d, \phi_{init}^i)$ are solved, and a maximally permissive strategy $\mathcal{S}_i^d$ is computed for them, if one exists. Computed strategies are then composed in line 11 of Algorithm 1 to obtain a strategy $\mathcal{S}$ for the whole system. The strategy $\mathcal{S}$ is then projected back to sub-games, and it is compared if all the projected strategies are equivalent to the strategies computed for the subproblems. If that is the case, the main loop terminates, while $\mathcal{S}$ is winning for the game $(\mathcal{G}^d, \phi_{init})$ where $(\mathcal{G}^d, \phi_{init})$ is the safety game associated with the multi-agent system $\mathcal{M}$ and objective Φ. Otherwise, at least one of the subproblems needs to be restricted. Each sub-game is restricted by the computed projection, and the process is repeated. The loop terminates either if at some iteration a subproblem becomes unrealizable, or if permissive strategies $\mathcal{S}_1, \cdots, \mathcal{S}_k$ reach a fixed point. In the latter case, a set of strategies, one for each controlled agent is extracted from $\mathcal{S}$ as explained below.

4.3 Computing Strategies for the Agents

Let $\mathcal{V}^{\otimes} = \bigcup_{i=1}^{k} \mathcal{V}_{\mathcal{G}_i^d}$ be the set of all variables used to encode the game structures $\mathcal{G}_i^d$, and $\Lambda^c = \Lambda_{\mathtt{c}_1} \times \cdots \times \Lambda_{\mathtt{c}_n}$ be the set of controlled agents' actions. Once a permissive strategy $\mathcal{S} : \Sigma_{\mathcal{V}^{\otimes}} \to 2^{\Lambda^c}$ is computed, a winning strategy $\mathtt{S}_d : \Sigma_{\mathcal{V}^{\otimes}} \to$

Λ^c is obtained from $\mathcal{S}$ by restricting the non-deterministic action choices of the controlled agents to a single action. The strategy $\mathtt{S}_d$ is then decomposed into strategies $\mathtt{S}_1 : \Sigma_{\mathcal{V}^\otimes} \to \Lambda_{\mathtt{c}_1}, \cdots, \mathtt{S}_n : \Sigma_{\mathcal{V}^\otimes} \to \Lambda_{\mathtt{c}_n}$ for the agents simply by projecting the actions over system transitions to their corresponding agents. Formally, for any $s \in \Sigma_{\mathcal{V}^\otimes}$ such that $\mathcal{S}(s)$ is defined, let $\mathtt{S}_d(s) = \sigma \in \mathcal{S}(s)$ where $\sigma = (\sigma_1, \cdots, \sigma_n) \in \Lambda^c$ is an arbitrary action chosen from possible actions permitted by $\mathcal{S}$ in the state s. Agents' strategies are defined as $\mathtt{S}_i(s) = \sigma_i$ for $i = 1, \cdots, n$. Note that we assume each controlled agent has perfect knowledge about other controlled agents' observations. The following theorem establishes the correctness of Algorithm 1.

Theorem 1. *Algorithm 1 is sound*[2]*.*

Proof. Note that Algorithm 1 always terminates, that is because either eventually a fixed point over strategies is reached, or a sub-game becomes unrealizable which indicates that the objective cannot be enforced. Consider the permissive strategies $\mathcal{S}_i^d$ and their projections $\mathcal{C}_i$. We have $\mathcal{C}_i(s) \subseteq \mathcal{S}_i^d(s)$ for any $s \in \Sigma_{\mathcal{V}}$, and as a result of composing and projecting intermediate strategies, we will obtain more restricted sub-games. As the state space and available actions in any state is finite, at some point, either a sub-game becomes unrealizable because the system player becomes too restricted and cannot win the game, or all strategies reach a fixed point. Therefore, the algorithm always terminates.

We now show that Algorithm 1 is sound, i.e., if it computes strategies $(\mathtt{S}_1, \cdots, \mathtt{S}_n)$, then the strategy $\mathtt{S} = \bigotimes_{i=1}^n \mathtt{S}_i$ is a winning strategy in the game $(\mathcal{G}^{\mathcal{M}}, \phi_{init}, \Phi)$, where $\mathcal{G}^{\mathcal{M}}$ is the game structure induced by $\mathcal{M}$. Let $\mathcal{S}^* = \bigotimes_{i=1}^k \mathcal{S}_i^d$ be the fixed point reached over the strategies. First note that any run in $\mathcal{G}_i^d[\mathcal{S}_i^d]$ starting from a state $s \models \phi_{init}^i$ for $1 \leq i \leq k$ satisfies the conjunct Φ_i since $\mathcal{S}_i^d$ is winning in the corresponding safety game. That is, the restriction of the game structure $\mathcal{G}_i^d$ to the strategy $\mathcal{S}_i^d$ satisfies Φ_i. Consider any run $\pi = s_0 s_1 s_2 \cdots$ in the restricted game structure $\mathcal{G}^d[\mathcal{S}^*]$ starting from the initial state $s_0 \models \phi_{init}$ where $\mathcal{G}^d = \bigotimes_{i=1}^k \mathcal{G}_i^d$. Let $\pi^i = s_0^i s_1^i s_2^i \cdots$ for $1 \leq i \leq k$ be the projection of π with respect to variables $\mathcal{V}_i^d$ of the game structure $\mathcal{G}_i^d$, i.e., $s_j^i = s_{j_{|\mathcal{V}_i^d}}$ for $j \geq 0$. Since $s_0^i \models \phi_{init}^i$ and $\mathcal{S}_i^d$ is equivalent to the projection of $\mathcal{S}^*$ with respect to variables and actions in the game structure $\mathcal{G}_i^d$, it follows that π^i is a winning run in the safety game $(\mathcal{G}_i^d[\mathcal{S}_i^d], \Phi_i)$, i.e., $\pi^i \models \Phi_i$. As $\pi^i \models \Phi_i$ for $1 \leq i \leq k$, we have $\pi \models \Phi = \bigwedge_{i=1}^k \Phi_i$. It follows that $\mathcal{S}^*$ is winning in the safety game $(\mathcal{G}^d, \phi_{init})$. Moreover, $\mathcal{S}^*$ is also winning with respect to the original game as $(\mathcal{G}^d, \phi_{init})$ is the safety game associated with $(\mathcal{G}^{\mathcal{M}}, \phi_{init}, \Phi)$ [15]. It is easy to see that the set $(\mathtt{S}_1, \cdots, \mathtt{S}_n)$ of strategies extracted from $\mathcal{S}^*$ by Algorithm 1 is winning for the game $(\mathcal{G}^{\mathcal{M}}, \phi_{init}, \Phi)$. $\square$

[2] In [15] it is shown that bounded synthesis is complete by proving the existence of a sufficiently large bound. Following their result, it can be shown that Algorithm 1 is also complete. However, in practice, the required bound is rather high and instead an incremental approach is used for synthesis.

Remark 2. Algorithm 1 is different from compositional algorithm proposed in [15] in two ways. First, it composes maximally permissive strategies in contrast to composing game structures as proposed in [15]. The advantage is that strategies usually have more compact symbolic representations compared to game structures[3]. Second, in the compositional algorithm in [15], sub-games are composed and a symbolic step, i.e., a post or pre-image computation, is performed over the composite game. In our experiments, performing a symbolic step over composite game resulted in a poor performance, often worse than the centralized algorithm. Algorithm 1 removes this bottleneck as it is not required in our setting. This leads to a significant improvement in algorithm's performance since image and pre-image computations are typically the most expensive operations performed by symbolic algorithms [35].

5 Case Study

We now demonstrate the techniques on a robot motion planning case study similar to those that can be found in the related literature (e.g., [8,9,10].) Consider a square grid-world with some static obstacles similar to the one depicted in Figure 1. We consider a multi-agent system $\mathcal{M} = \{\mathtt{u}_1, \cdots, \mathtt{u}_m, \mathtt{c}_1, \cdots, \mathtt{c}_n\}$ with uncontrolled robots $\mathcal{M}^u = \{\mathtt{u}_1, \cdots, \mathtt{u}_m\}$ and controlled ones $\mathcal{M}^c = \{\mathtt{c}_1, \cdots, \mathtt{c}_n\}$. At any time-step, any controlled robot $\mathtt{c}_i$ for $1 \leq i \leq n$ can move to one of its neighboring cells by taking actions $up_i, down_i, left_i$, and $right_i$, or it can stay put by taking the action *stop*. Any uncontrolled robot $\mathtt{u}_j$ for $1 \leq j \leq m$ stays on the same row where they are initially positioned, and at any time-step can move to their left or right neighboring cells by taking actions $left_j$ and $right_j$, respectively. We consider the following objectives for the systems, (Φ_1) collision avoidance, i.e., controlled robots must avoid collision with static obstacles and other robots, (Φ_2) formation maintenance, i.e., each controlled robot $\mathtt{c}_i$ must keep a linear formation (same horizontal or vertical coordinate) at all times with the subsequent controlled robot $\mathtt{c}_{i+1}$ for $1 \leq i < n$, (Φ_3) bounded reachability, i.e., controlled robots must reach the bottom row in a pre-specified number of steps. We consider two settings. First we assume all agents are perfect, i.e., all agents have full-knowledge of the state of the system at any time-step. Then we assume controlled agents are imperfect and can observe uncontrolled robots only if they are nearby and occupying an adjacent cell, similar to Example 1.

We apply two different methods to synthesize strategies for the agents. In the *Centralized* method, a game structure for the whole system is obtained first, and then a winning strategy is computed with respect to the considered objective. In the *Compositional* approach, the strategy is computed compositionally using Algorithm 1. We implemented the algorithms in Java using the BDD package JDD [36]. The experiments are performed on an Intel core i7 3.40 GHz machine with 16GB memory. In our experiments, we vary the number of uncontrolled and

[3] Strategies are mappings from states to actions while game structures include more variables and typically have more complex BDD representation as they refer to states, actions, and next states.

controlled agents, size of the grid-world, and the objective of the system as shown in Tables 1 and 2. The columns show the number of uncontrolled and controlled robots, considered objective, size of the grid-world, number of variables in the system, and the time and memory usage for different approaches, respectively. Furthermore, we define $\Phi_{12} = \Phi_1 \wedge \Phi_2$, $\Phi_{13} = \Phi_1 \wedge \Phi_3$, and $\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3$.

Table 1: Experimental results for systems with perfect agents

					Centralized		Compositional	
$\lvert\mathcal{M}^u\rvert$	$\lvert\mathcal{M}^c\rvert$	objective	size	$\lvert\mathcal{V}\rvert$	time	mem (MB)	time	mem (MB)
1	1	Φ_1	64×64	52	**72 ms**	6.6	105 *ms*	6.6
1	1	Φ_1	128×128	60	**93 ms**	6.6	101 *ms*	6.6
1	2	Φ_{13}	16×16	79	14.9 *min*	365.5	**4.2 s**	**19.3**
1	2	Φ_{13}	32×32	95	mem out	mem out	**34.4 s**	**50.8**
1	2	Φ	16×16	79	400.3 *s*	239.7	**5.1 s**	**19.4**
1	2	Φ	32×32	95	155.8 *min*	1209	**33.1 s**	**38.3**
1	3	Φ_{13}	4×4	66	22 *s*	50.8	**0.8 s**	**6.8**
1	3	Φ_{13}	8×8	88	mem out	mem out	**98.4 s**	**101.2**
2	1	Φ	8×8	51	106.4 *s*	322	**33 ms**	**6.6**
2	1	Φ	128×128	107	mem out	mem out	**3.5s**	**6.7**
2	2	Φ	4×4	56	3.2 *s*	19.4	**201 ms**	**6.6**
2	2	Φ	8×8	76	10.6 *min*	460	**14.4 s**	**19.4**
2	3	Φ_{13}	4×4	75	19.1 *min*	497.8	**8.4 s**	**25.9**
2	3	Φ_{13}	8×8	101	mem out	mem out	**30.2 min**	**800.2**
2	3	Φ	8×8	101	mem out	mem out	**12.7 min**	**302.6**

Multi-agent systems with perfect agents. Table 1 shows some of our experimental results for the setting where all agents are perfect (more experimental data is provided in the technical report.) Note that the compositional algorithm does not always perform better than the centralized alternative. Indeed, if the conjuncts of objectives involve a large subset of agents, compositional algorithm comes closer to the centralized algorithm. Intuitively, if the agents are "strongly" coupled, the overhead introduced by compositional algorithm is not helpful, and the central algorithm performs better. For example, when the system consists of a controlled robot and an uncontrolled one along with a single safety objective, compositional algorithm coincides with the centralized one, and centralized algorithm performs slightly better. However, if the sub-problems are "loosely" coupled, which is the case in many practical problems, the compositional algorithm significantly outperforms the centralized one, both from time and memory perspective, as we increase the number of agents and make the objectives more complex, and it can solve problems where the centralized algorithm is infeasible.

Multi-agent systems with imperfect controlled agents. Not surprisingly, scalability is a bigger issue when it comes to games with imperfect information due to the subset construction procedure, which leads to yet another reason for compositional algorithm to perform better than the centralized al-

Table 2: Experimental results for systems with imperfect agents

					Centralized		Compositional	
$\lvert\mathcal{M}^u\rvert$	$\lvert\mathcal{M}^c\rvert$	objective	size	$\lvert\mathcal{V}\rvert$	time	mem (MB)	time	mem (MB)
1	2	Φ_{12}	4×4	127	1.7 *s*	6.7	**0.6 s**	6.7
1	2	Φ_{12}	6×6	235	28.6 *s*	31.9	**10.2 s**	**19.3**
1	2	Φ_{12}	8×8	235	229.7 *s*	126.6	**95 s**	**57.1**
1	2	Φ_{12}	9×9	375	time out	time out	**306 s**	**94.9**
1	2	Φ_{12}	10×10	375	time out	time out	**9.7 min**	**176.7**
1	2	Φ_{13}	4×4	143	1.4 *s*	6.7	**303 ms**	6.7
1	2	Φ_{13}	6×6	255	38.2 *s*	57.1	**5 s**	**13**
1	2	Φ_{13}	8×8	255	8.9 *min*	252.2	**38.3 s**	**51**
1	2	Φ_{13}	9×9	395	time out	time out	**114.9 s**	**88.6**
1	2	Φ_{13}	10×10	395	time out	time out	**279.9 s**	**157.8**
1	2	Φ	4×4	143	2.3 *s*	6.7	**0.7 s**	**6.7**
1	2	Φ	6×6	255	46.2 *s*	50.8	**10 s**	**19.3**
1	2	Φ	8×8	255	344.5 *s*	202.1	**129.9 s**	**57.1**
1	2	Φ	9×9	395	time out	time out	**309.9 s**	**101.2**
1	2	Φ	10×10	395	time out	time out	**9.6 min**	**176.7**
1	3	Φ_1	4×4	186	144.3 *s*	69.7	**0.9 s**	**6.7**
1	3	Φ_1	6×6	346	time out	time out	**17.7 s**	**38.2**
1	3	Φ_1	8×8	346	time out	time out	**190.9 s**	**176.7**
1	3	Φ_1	10×10	554	time out	time out	**24.6 min**	**730.6**
1	3	Φ_{13}	4×4	210	265.8 *s*	214.5	**0.9 s**	**6.7**
1	3	Φ_{13}	6×6	376	time out	time out	**49.2 s**	**57.1**
1	3	Φ_{13}	8×8	376	time out	time out	**483.9 s**	**214.5**
1	3	Φ_{13}	9×9	584	time out	time out	**31.7 min**	**441.1**
1	3	Φ	6×6	376	time out	time out	**36 s**	**50.8**
1	3	Φ	8×8	376	time out	time out	**343.4 s**	**201.9**
1	3	Φ	10×10	584	time out	time out	**39.6 min**	**774.7**

ternative. Table 2 shows some of our experimental results for the setting where controlled agents are imperfect. While the centralized approach fails to compute the knowledge game structure due to the state explosion problem, the compositional algorithm performs significantly better by decomposing the problem and performing subset construction on smaller and more manageable game structures of imperfect information.

6 Conclusions and Future Work

We proposed a framework for controller synthesis for multi-agent systems. We showed that by taking advantage of the structure in the system to compositionally synthesize the controllers, and by representing and exploring the state space symbolically, we can achieve better scalability and solve more realistic problems. Our preliminary results shows the potential of reactive synthesis as planning algorithms in the presence of dynamically changing and adversarial environment.

In our implementation, we performed the subset construction procedure symbolically and we only constructed the part of it that is reachable from the initial state. One of our observations was that by considering more structured observation functions for game structures of imperfect information, such as the ones considered in our case study where the robots show a "local" observation behavior, the worst case exponential blow-up in the constructed knowledge game structure does not occur in practice. In future, we plan to investigate how considering more restricted yet practical observation functions can enable us to handle systems with imperfect agents of larger size.

References

1. Jennings, J.S., Whelan, G., Evans, W.F.: Cooperative search and rescue with a team of mobile robots. In: 8th International Conference on Advanced Robotics, IEEE (1997)
2. Rus, D., Donald, B., Jennings, J.: Moving furniture with teams of autonomous robots. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE (1995)
3. Balch, T., Arkin, R.C.: Behavior-based formation control for multirobot teams. IEEE Transactions on Robotics and Automation (1998)
4. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM (1989)
5. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: 31st Annual Symposium on Foundations of Computer Science, IEEE (1990)
6. Peterson, G., Reif, J., Azhar, S.: Lower bounds for multiplayer noncooperative games of incomplete information. Computers & Mathematics with Applications (2001)
7. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. Journal of Computer and System Sciences (2012)
8. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE Transactions on Robotics (2009)
9. Saha, I., Ramaithitima, R., Kumar, V., Pappas, G.J., Seshia, S.A.: Automated composition of motion primitives for multi-robot systems from safe ltl specifications. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE (2014)
10. Ayanian, N., Kallem, V., Kumar, V.: Synthesis of feedback controllers for multiple aerial robots with geometric constraints. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE (2011)
11. Keviczky, T., Borrelli, F., Balas, G.J.: Decentralized receding horizon control for large scale dynamically decoupled systems. Automatica (2006)
12. Dunbar, W.B., Murray, R.M.: Distributed receding horizon control for multi-vehicle formation stabilization. Automatica (2006)
13. Sahin, E., Girgin, S., Bayindir, L., Turgut, A.E.: Swarm robotics. Swarm Intelligence (2008)
14. Shi, Z., Tu, J., Zhang, Q., Liu, L., Wei, J.: A survey of swarm robotics system. In: Advances in Swarm Intelligence. Springer (2012)
15. Filiot, E., Jin, N., Raskin, J.F.: Antichains and compositional algorithms for ltl synthesis. Formal Methods in System Design (2011)

16. Kupferman, O., Piterman, N., Vardi, M.: Safraless compositional synthesis. In: Computer Aided Verification, Springer (2006)
17. Baier, C., Klein, J., Klüppelholz, S.: A compositional framework for controller synthesis. In: Concurrency Theory. Springer (2011)
18. Sohail, S., Somenzi, F.: Safety first: A two-stage algorithm for ltl games. In: Formal Methods in Computer-Aided Design, IEEE (2009)
19. Alur, R., Moarref, S., Topcu, U.: Pattern-based refinement of assume-guarantee specifications in reactive synthesis. 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (2015)
20. Lustig, Y., Vardi, M.Y.: Synthesis from component libraries. International Journal on Software Tools for Technology Transfer (2013)
21. Alur, R., Moarref, S., Topcu, U.: Compositional synthesis with parametric reactive controllers. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, ACM (2016)
22. Reif, J.H.: The complexity of two-player games of incomplete information. Journal of computer and system sciences (1984)
23. Chatterjee, K., Henzinger, T.A.: Semiperfect-information games. In: Foundations of Software Technology and Theoretical Computer Science. Springer (2005)
24. De Wulf, M., Doyen, L., Raskin, J.F.: A lattice theory for solving games of imperfect information. In: Hybrid Systems: Computation and Control. Springer (2006)
25. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.F.: Algorithms for omega-regular games with imperfect information. In: Computer Science Logic, Springer (2006)
26. Wongpiromsarn, T., Topcu, U., Murray, R.M.: Receding horizon temporal logic planning. IEEE Transactions on Automatic Control (2012)
27. Kress-gazit, H., Wongpiromsarn, T., Topcu, U.: Correct, reactive robot control from abstraction and temporal logic specifications
28. Wongpiromsarn, T., Ulusoy, A., Belta, C., Frazzoli, E., Rus, D.: Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In: IEEE International Conference on Robotics and Automation, IEEE (2013)
29. Kloetzer, M., Belta, C.: Automatic deployment of distributed teams of robots from temporal logic motion specifications. IEEE Transactions on Robotics (2010)
30. Ozay, N., Topcu, U., Murray, R.M.: Distributed power allocation for vehicle management systems. In: 50th IEEE Conference on Decision and Control and European Control Conference, IEEE (2011)
31. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT press (1999)
32. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Automated Technology for Verification and Analysis. Springer (2007)
33. Ehlers, R.: Symbolic bounded synthesis. Formal Methods in System Design (2012)
34. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM (2002)
35. Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in n log n symbolic steps. In: Formal Methods in Computer-Aided Design, Springer (2000)
36. Vahidi, A.: Jdd. http://javaddlib.sourceforge.net/jdd/index.html