Information and Computation 261 (2018) 616–633

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Compositional and symbolic synthesis of reactive controllers for multi-agent systems

Rajeev Alur[a], Salar Moarref[b,*], Ufuk Topcu[c]

[a] 3330 Walnut Street, Philadelphia, PA 19104, United States
[b] 124 Hoy Rd, Ithaca, NY 14850, United States
[c] 210 East 24th Street, Austin, TX 78712, United States

ARTICLE INFO

Article history:
Received 30 December 2016
Available online 1 March 2018

Keywords:
Reactive synthesis
Compositional synthesis
Multi-agent systems
Formal methods

ABSTRACT

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 actions of uncontrolled agents in order to fulfill their objectives. Moreover, agents may be imperfect, i.e., only partially observe their environment. 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 alternative, both from time and memory perspective, and can solve problems where the centralized algorithm is infeasible.

1. Introduction

Complex systems often consist of multiple agents (or subsystems or components) interacting with each other and their environment to perform certain tasks and achieve specified objectives. For example, teams of robots are employed to perform tasks such as monitoring, surveillance, and disaster response in different domains including assembly planning [1], evacuation [2], search and rescue [3], localization [4], object transportation [5], and formation control [6]. With growing complexity of systems and guarantees they are required to provide, 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 such that satisfaction of the specified objectives is guaranteed. Given a model of the system describing the interaction of a controllable plant with its environment and an objective specified in a formal language such as linear temporal logic (LTL), controller synthesis problem seeks 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.

Unfortunately, high complexity of synthesis procedures has restricted their application to relatively small-sized problems. The pioneering work by Pnueli et al. [7] showed that reactive synthesis from LTL specifications is intractable which

* Corresponding author.
E-mail addresses: alur@cis.upenn.edu (R. Alur), sm945@cornell.edu (S. Moarref), utopcu@utexas.edu (U. Topcu).

https://doi.org/10.1016/j.ic.2018.02.021
0890-5401/© 2018 Elsevier Inc. All rights reserved.

prohibited the practitioners from utilizing synthesis algorithms. This issue becomes more evident for multi-agent systems, as adding each agent can often increase the size of the state space exponentially. Furthermore, distributed reactive synthesis [8] and multi-player games of incomplete information [9] are undecidable in general. Despite these discouraging results, recent advances in this growing research area have enabled automatic synthesis of interesting real-world systems [10], 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 [11–13]. 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, 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 *dynamically-decoupled* and are inspired by robot motion planning, decentralized control [14,15], and swarm robotics [16,17] literature. Intuitively, in a dynamically-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 own position, velocity, etc. However, dynamically-decoupled agents may be coupled through objectives, that is, 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 lacks 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 subproblems 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.

We propose a compositional framework for dynamically-decoupled multi-agent systems based on automatic decomposition of objectives and compositional reactive synthesis using maximally permissive strategies [18]. 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, solve them again with updated constraints, and repeat 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. Our findings show the potential of symbolic and compositional reactive synthesis methods as planning algorithms in presence of dynamically changing and possibly adversarial environment.

Related work. Synthesis problem was first recognized by Church [19]. The problem of synthesizing reactive systems from a specification given in linear temporal logic was considered by Pnueli et al. [7], where they propose a synthesis algorithm that first transforms the LTL specification into a Büchi automaton, which is then translated into a deterministic Rabin automaton using Safra's determinization procedure [20]. This double translation causes a doubly exponential time complexity which is unavoidable [21]. The high complexity of the synthesis process was discouraging, however, it was shown later that there are several interesting cases where the synthesis problem can be solved in polynomial time [22,23]. Bloem et al. [10] present polynomial time algorithms for the realizability and synthesis problems for a more general fragment of LTL known as Generalized Reactivity (1) (GR(1)). They show the efficiency and expressivity of GR(1) by applying their algorithms to a realistic industrial hardware case study of a medium size.

Compositional reactive synthesis has been considered in some recent works. Kupferman et al. [24] propose a Safraless approach that reduces the LTL realizability problem to Büchi games. Their approach is then extended to treat specifications that are conjunction of LTL properties compositionally [25]. There is no notion of maximally permissive strategy for Büchi games, and to our best knowledge their algorithms are not implemented. Baier et al. [26] give a compositional framework for treating multiple linear-time objectives inductively. To this end, they introduce the concept of most general strategies which generate all decision functions that guarantee the objective under consideration. Sohail et al. [27] propose an algorithm to compositionally construct a parity game from conjunctive LTL specifications. Alur et al. [28] show how local specifications of components can be refined compositionally to ensure satisfaction of a global specification. Lustig et al. [29] study the problem of LTL synthesis from libraries of reusable components. Alur et al. [30] propose a framework for compositional synthesis from a library of parametric and reactive controllers. Filiot et al. [18] present monolithic and compositional algorithms to solve the LTL realizability problem. They reduce the LTL realizability problem to solving safety games, and 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 [18] seems to be the most successful application of compositional synthesis in practice.

Two-player games of imperfect information are studied in [31–34], 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 [18,34] 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., [11,35,36]). 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., [37,38]) 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 [12], 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 [11,39] 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 reactive 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. Note that in this paper we assume that a finite-state abstraction of the system is given and we present compositional algorithms for synthesizing *discrete* controllers. Computed controllers can then be refined to controllers enforcing the specification over the original system using standard techniques in the literature [40].

Dynamically decoupled multi-agent systems are considered and studied in different research areas, e.g., in decentralized control [14,15], interpreted systems [41,42], and swarm robotics [16,17] literature. In this paper, we model dynamically-decoupled multi-agent systems in a way that best fits our solution approach and intended application. The problem considered in this paper is also related to the planning algorithms that are well-studied in artificial intelligence community, and a huge body of research has been developed over the last decades for planning using logic-based representations such as propositional and first order logics (see [43] and [44] for a comprehensive introduction). In this paper we use temporal logic to specify the system objectives which allows describing time-varying aspects of discrete planning problems. Furthermore, we consider environments that are partially-observable and dynamically changing.

Organization. The organization of the paper is as follows. In Section 2 we introduce some notation, background and definitions that are used in the rest of the paper. In Section 3 we explain how multi-agent systems are modeled in our framework. In Section 4 we present the compositional synthesis algorithm. In Section 5 we apply the centralized and compositional synthesis algorithms to a robot motion planning case study and compare their performance. Finally, in Section 6 we conclude the paper and point out some future directions. This paper is based on a conference publication [45] and contains detailed explanations of the methods and results presented there.

2. Preliminaries

Let $\mathbb{Z}$ be the set of integers. For $a, b \in \mathbb{Z}$, let $[a..b] = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$.

2.1. 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), $\wedge$ (conjunction), $\vee$ (disjunction), and $\rightarrow$ (implication)) and temporal operators (e.g., $\Box$ (always), $\bigcirc$ (next), $\mathcal{U}$ (until), and $\Diamond$ (eventually)). The formulas of LTL are defined over a set of atomic propositions (Boolean variables) $\mathcal{V}$. The syntax is given by the grammar:

$$\Phi := v \mid \Phi \vee \Phi \mid \neg\Phi \mid \bigcirc \Phi \mid \Phi\, \mathcal{U}\, \Phi \quad \text{for } v \in \mathcal{V}$$

We define $\texttt{true} = v \vee \neg v$, $\texttt{false} = v \wedge \neg v$, $\Diamond\Phi = \texttt{true}\ \mathcal{U}\ \Phi$, and $\Box\Phi = \neg\Diamond\neg\Phi$. 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. Formally, we define $s \models \phi$ inductively as

- for $v \in \mathcal{V}$, $s \models v$ if and only if $v \in s$,
- $s \models \neg\phi$ if and only if $s \not\models \phi$, and
- $s \models \phi \vee \psi$ if and only if $s \models \phi$ or $s \models \psi$.

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., $\mathcal{L}(\Phi) = \{w \in (\Sigma_{\mathcal{V}})^{\omega} \mid w \models \Phi\}$, where the satisfaction relation $w = \sigma_0\sigma_1\sigma_2 \cdots \models \Phi$ is inductively defined as follows:

1. $w \models v$ if $v \in \sigma_0$,
2. $w \models \Phi_1 \vee \Phi_2$ if $w \models \Phi_1$ or $w \models \Phi_2$,
3. $w \models \neg\Phi$ if $w \not\models \Phi$,
4. $w \models \bigcirc\Phi$ if $\sigma_1\sigma_2 \cdots \models \Phi$, and
5. $w \models \Phi_1 \mathcal{U} \Phi_2$ if there is $n \geq 0$ such that $\sigma_n\sigma_{n+1} \cdots \models \Phi_2$ and for all $0 \leq i < n, \sigma_i\sigma_{i+1} \cdots \models \Phi_1$.

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 $\mathcal{X}$, i.e., $s_{|\mathcal{X}} = \{x \in \mathcal{X} \mid x \in s\}$. Given non-overlapping sets of Boolean variables $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_n$, we use the notation $\phi(\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_n)$ to indicate that ϕ is a predicate over $\mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_n$. We often use predicates over $\mathcal{V} \cup \mathcal{V}'$ where $\mathcal{V}'$ is the set of primed versions of the variables in $\mathcal{V}$, i.e., $\mathcal{V}' = \{v' \mid v \in \mathcal{V}\}$. Given a predicate $\phi(\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_n, \mathcal{V}'_1, \mathcal{V}'_2, \cdots, \mathcal{V}'_n)$ and assignments $s_i, t_i \in \Sigma_{\mathcal{V}_i}$, we use $(s_1, s_2, \cdots, s_n, t'_1, t'_2, \cdots, t'_n) \models \phi$ to indicate $s_1 \cup s_2 \cup \cdots \cup s_n \cup t'_1 \cup t'_2 \cup \cdots \cup t'_n \models \phi$ where $t'_i = \{v' \in \mathcal{V}'_i \mid v \in t_i\}$. For a predicate ϕ over variables $\mathcal{V}$, we let $[\![\phi]\!]$ be the set of valuations over $\mathcal{V}$ that make ϕ true, that is, $[\![\phi]\!] = \{s \in \Sigma_{\mathcal{V}} \mid s \models \phi\}$. 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. Formally, $Same(\mathcal{Z}, \mathcal{Z}') = \bigwedge_{z \in \mathcal{Z}} z \leftrightarrow z'$.

OBDDs (or BDDs for short) can be used for obtaining concise representations of sets and relations over finite domains [46]. If R is an n-ary relation over $\{0, 1\}$, then R can be represented by the BDD for its *characteristic function*:

$$f_R(x_1, \cdots, x_n) = 1 \text{ if and only if } R(x_1, \cdots, x_n) = 1.$$

With a little bit abuse of notation and when it is clear from the context, we treat sets and functions as their corresponding predicates.

Example 1. Consider a function $f : [0..3] \to [0..3]$ defined as $f(\mathtt{i}) = \mathtt{i}$ for $\mathtt{i} \in [0..3]$. Let $\mathtt{a}_1$ and $\mathtt{a}_0$ ($\mathtt{b}_1$ and $\mathtt{b}_0$) be Boolean variables used to encode input (output, respectively) of f. The function f can be represented by a predicate $\phi_f = \mathtt{a}_1 \leftrightarrow \mathtt{b}_1 \wedge \mathtt{a}_0 \leftrightarrow \mathtt{b}_0$, and symbolically encoded by a BDD.

2.2. 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 variables encoding 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 $\gamma : \Sigma_{\mathcal{OBS}} \to 2^{\Sigma_{\mathcal{V}}} \backslash \emptyset$ is an observation function that maps each observation to a set of states. We assume that the set $\{\gamma(o) \mid o \in \Sigma_{\mathcal{OBS}}\}$ partitions the state space $\Sigma_{\mathcal{V}}$.[1] 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 playing. Without loss of generality, we assume that player-1 always starts the game. Let $t \in \mathcal{V}$ be a special variable with domain $\{1, 2\}$ determining which player's turn it is during the game. For $i = 1, 2$, let $\Sigma^i_{\mathcal{V}} = \{s \in \Sigma_{\mathcal{V}} \mid s_{|t} = i\}$ denote player-i's states in the game structure. At any state $s \in \Sigma^i_{\mathcal{V}}$, player-$i$ chooses an action $\ell \in \Sigma_\Lambda$ 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 $\Gamma(s) = \{\ell \in \Sigma_\Lambda \mid \exists s' \in \Sigma_{\mathcal{V}'}.\ (s, \ell, s') \models \tau\}$ 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 $\pi = s_0s_1s_2 \cdots$ such that $s_0 = s_{init}$ and, for all $i > 0$, there is an action $\ell_i \in \Sigma_\Lambda$ 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 is finite and ends in a state $s \in \Sigma_{\mathcal{V}}$ where $\Gamma(s) = \emptyset$. The *observation sequence* of π is the unique sequence $Obs(\pi) = o_0o_1o_2 \cdots$ such that for all $i \geq 0$, we have $s_i \in \gamma(o_i)$. For $\ell \in \Sigma_\Lambda$ and $X \subseteq \Sigma_{\mathcal{V}}$, let $Post^{\mathcal{G}}_{\ell}(X) = \{r \in \Sigma_{\mathcal{V}} \mid \exists s \in X\ :\ (s, \ell, r') \models \tau\}$.

Strategies. A *strategy* $\mathtt{S}$ in $\mathcal{G}$ for player-i, $i \in \{1, 2\}$, is a function $\mathtt{S} : (\Sigma_{\mathcal{V}})^*.\Sigma^i_{\mathcal{V}} \to \Sigma_\Lambda$. A strategy $\mathtt{S}$ in $\mathcal{G}$ for player-2 is *observation-based* if for all prefixes $\rho_1, \rho_2 \in (\Sigma_{\mathcal{V}})^*.\Sigma^2_{\mathcal{V}}$, if $Obs(\rho_1) = Obs(\rho_2)$, then $\mathtt{S}(\rho_1) = \mathtt{S}(\rho_2)$. We are interested in the

[1] This assumption can be weakened to a covering of the state space where observations can overlap [34,33].

existence of observation-based strategies for player-2. Given two strategies $\mathtt{S}_1$ and $\mathtt{S}_2$ for player-1 and player-2, respectively, 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 $s_j \in \Sigma^i_{\mathcal{V}}$ and $(s_j, \mathtt{S}_i(s_0 \cdots s_j), s'_{j+1}) \models \tau$. A strategy $\mathtt{S}$ is *memory-less* (a.k.a. positional) if it is independent of the history of the game and only depends on the current state. A memory-less strategy for player-i can be represented as a function $\mathtt{S} : \Sigma^i_{\mathcal{V}} \to \Sigma_\Lambda$.

Winning condition. A game $(\mathcal{G}, \phi_{init}, \Phi)$ consists of a game structure $\mathcal{G}$, a predicate ϕ_{init} specifying a set of initial states, 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 and all initial states $s_{init} \models \phi_{init}$, we have $\Omega_{\mathtt{S}_1,\mathtt{S}_2}(s_{init}) \subseteq \Pi$, that is, all possible outcomes are winning for player-2. It is well known that for ω-regular objectives, the games are determined, i.e., either player-2 has a winning strategy or player-1 has a spoiling strategy [47]. We say the game $(\mathcal{G}, \phi_{init}, \Phi)$ is realizable if and only if player-2 has a winning strategy in the game $(\mathcal{G}, \phi_{init}, \Phi)$.

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 [31,34]. Formally, let $\mathcal{V}^K$ be a set of Boolean variables of size $2^{|\mathcal{V}|}$ and $\mathcal{H}$ be a bijective mapping that maps each state $v^K \in \Sigma_{\mathcal{V}^K}$ to a unique set $\mathcal{H}(v^K) \subseteq \Sigma_{\mathcal{V}}$ of states. We define the *knowledge-based subset construction* of $\mathcal{G}$ as the game structure $\mathcal{G}^K = (\mathcal{V}^K, \Lambda, \tau^K)$ of perfect information where $(v^K, \ell, w^K) \models \tau^K$ if and only if there exists $obs \in \Sigma_{\mathcal{OBS}}$ such that $\mathcal{H}(w^K) \neq \emptyset$ and

- $\mathcal{H}(w^K) = (\bigcup_{\sigma \in \Sigma_\Lambda} Post^{\mathcal{G}}_\sigma(\mathcal{H}(v^K))) \cap \gamma(obs)$ if $v^K \in \Sigma^1_{\mathcal{V}^K}$, and
- $\mathcal{H}(w^K) = Post^{\mathcal{G}}_\ell(\mathcal{H}(v^K)) \cap \gamma(obs)$ if $v^K \in \Sigma^2_{\mathcal{V}^K}$.

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. Note that here we assume actions of player-1 cannot be observed by player-2. That is, when it is player-2's turn to play, she does not know what action player-1 took at the previous step. That is why player-1 and player-2 states are treated differently in the transition relation of the knowledge game structure. 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 $\mathcal{G}$. Given an initial state, we construct the knowledge game structure symbolically, and only part of it that is *reachable* from the initial state (see Appendix A for details of constructing the knowledge game structure). In the rest of this section, we only consider game structures of perfect information.

Safety games. In this paper, we use the bounded synthesis approach [48,18] to solve the synthesis problems from LTL specifications. In bounded synthesis approach, the LTL specification is translated into a universal co-Büchi word or tree automaton, which is then, together with a bound $k \in \mathbb{N}$, is used to build a *safety game* such that winning strategies in the game correspond to implementations satisfying the specification. Intuitively, the bound indicates the maximum number of times that rejecting states in the co-Büchi automaton can be visited, so it implicitly defines a safety game. 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 any 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. Next, we summarize how LTL specifications are transformed into safety games following the construction proposed in [18].

An *infinite word automaton* over the finite alphabet Σ is a tuple $A = (Q, \Sigma, q_0, \delta, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of rejecting states and $\delta : Q \times \Sigma \to 2^Q$ is a transition function. We say A is *deterministic* if $\forall q \in Q. \forall \sigma \in \Sigma. |\delta(q, \sigma)| \leq 1$. A run of A on a word $w = w_0 w_1 \cdots \in \Sigma^\omega$ is an infinite sequence of states $\sigma = \sigma_0 \sigma_1 \cdots \in Q^\omega$ such that $\sigma_0 = q_0$ and for all $i \geq 0. \sigma_{i+1} \in \delta(\sigma_i, w_i)$. Let $Runs_A(w)$ be the set of all runs of A on w. We denote by $Visit(\sigma, q)$ the number of times the state q occurs along the run σ. A word w is accepted by A with *universal co-Büchi accepting condition* if $\forall \sigma \in Runs_A(w).\ \forall q \in F.\ Visit(\sigma, q) < \infty$, i.e., rejecting states are visited only finitely often. Similarly, w is accepted by A with *universal K-co-Büchi* accepting condition if $\forall \sigma \in Runs_A(w).\ \forall q \in F.\ Visit(\sigma, q) \leq K$. We denote by $L_{uc}(A)$ and $L_{uc,K}(A)$ the set of words accepted by A with the universal co-Büchi and universal K-co-Büchi accepting condition, respectively. With these interpretations in mind, we say A is a universal co-Büchi automaton (UCW) and that the pair (A, K) is a universal K-co-Büchi automaton (UKCW).

To reflect the game point of view of synthesis, the turn-based extension of infinite word automaton is defined as follows. A *turn-based automaton* A over the input alphabet $\Sigma_{\mathcal{I}}$ and output alphabet $\Sigma_{\mathcal{O}}$ is a tuple $A = (\Sigma_{\mathcal{I}}, \Sigma_{\mathcal{O}}, Q_1, Q_2, q_0, \delta_1, \delta_2, F)$ where Q_1, Q_2 are finite set of input and output states, respectively, $q_0 \in Q_1$ is the initial state, $F \subseteq Q_1 \cup Q_2$ is the set of rejecting states, and $\delta_1 : Q_1 \times \Sigma_{\mathcal{I}} \to 2^{Q_2}$ and $\delta_2 : Q_2 \times \Sigma_{\mathcal{O}} \to 2^{Q_1}$ are the player-1 and player-2 transition relations, respectively. Let $\Sigma = \Sigma_{\mathcal{I}} \cup \Sigma_{\mathcal{O}}$. A run on a word $w = (i_0 \cup o_0)(i_1 \cup o_1) \cdots \in \Sigma^\omega$ is a word $\sigma = \sigma_0 \sigma_1 \cdots \in (Q_1 Q_2)^\omega$ such that $\sigma_0 = q_0$ and for all $j \geq 0$, $(\sigma_{2j}, i_j, \sigma_{2j+1}) \in \delta_1$ and $(\sigma_{2j+1}, o_j, \sigma_{2j+2}) \in \delta_2$. All the accepting conditions defined for infinite word automaton carry over to turn-based automata in a straightforward manner. Turn-based automata with universal co-Büchi (universal K-co-Büchi) accepting condition is denoted by tbUCW (tbUKCW, respectively).

Associating a safety game with an LTL formula Φ is done as follows [18]: 1) construct a tbUCW A_Φ such that $L_{uc}(A_\Phi) = \mathcal{L}(\Phi)$, 2) given a bound $k \in \mathbb{N}$, construct a turn-based deterministic 0-co-Büchi automaton $det(A_\Phi, k) = (\Sigma_{\mathcal{I}}, \Sigma_{\mathcal{O}}, Q_1, Q_2, q_0, \delta_1, \delta_2, F)$ such that $L_{uc,0}(det(A_\Phi, k)) = L_{uc,k}(A_\Phi)$, and 3) obtain a safety game from $det(A_\Phi, k)$ as follows. Let $\mathcal{Q}$, $\mathcal{I}$, and $\mathcal{O}$ be sets of Boolean variables encoding sets $Q_1 \cup Q_2$, $\Sigma_{\mathcal{I}}$ and $\Sigma_{\mathcal{O}}$, respectively. Let predicates $\phi^{\Phi}_{init}(\mathcal{Q})$, $\phi^{\delta_1}(\mathcal{Q}, \mathcal{I}, \mathcal{Q}')$, $\phi^{\delta_2}(\mathcal{Q}, \mathcal{O}, \mathcal{Q}')$, and $\phi^F(\mathcal{Q})$ represent the initial state $\{q_0\}$, transition relations δ_1, δ_2, and the set of rejecting states F, respectively. Assume $t \notin \mathcal{Q} \cup \mathcal{I} \cup \mathcal{O}$ is a special *turn* variable with domain $\{1, 2\}$, representing which player's turn it is. The associated safety game with Φ is defined as $(\mathcal{G}^\Phi, \phi^\Phi_{init})$ where $\mathcal{G}^\Phi = (\mathcal{Q}, \mathcal{I} \cup \mathcal{O}, \tau)$ with

$$\tau = ((t = 1 \wedge t' = 2 \wedge \phi^{\delta_1}) \vee (t = 2 \wedge t' = 1 \wedge \phi^{\delta_2})) \wedge \neg\phi^F.$$

Intuitively, τ specifies that when its player-i's turn (i.e., $t = i$) for $i \in \{1, 2\}$, she makes a move according to δ_i. The conjunct $\neg\phi^F$ in definition of τ ensures that there is no outgoing transition from rejecting states, i.e., all rejecting states are dead-end. A winning strategy for player-2 in the safety game $(\mathcal{G}_\Phi, \phi^\Phi_{init})$ must avoid reaching dead-end states. We refer the readers to [18,49] for further details on reducing LTL formulas to safety games.

Maximally permissive strategies. Safety games are *memory-less determined*, i.e., player-2 wins the safety game $(\mathcal{G}, \phi_{init})$ where $\mathcal{G} = (\mathcal{V}, \Lambda, \tau)$ if and only if there exists a memory-less winning strategy $\mathtt{S} : \Sigma^2_{\mathcal{V}} \to \Sigma_\Lambda$. Let $W \subseteq \Sigma_{\mathcal{V}}$ 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* $\mathcal{S} : \Sigma^2_{\mathcal{V}} \to 2^{\Sigma_\Lambda}$ for player-2 is defined as follows: for any $s \in \Sigma^2_{\mathcal{V}}$, $\mathcal{S}(s) = \{\ell \in \Sigma_\Lambda \mid \forall r \in \Sigma_{\mathcal{V}'}.\ (s, \ell, r) \models \tau_s \to r \in W\}$, 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 $(\mathcal{G}, \Phi_{init})$.

Composition of two maximally permissive strategies $\mathcal{S}_1, \mathcal{S}_2 : \Sigma^2_{\mathcal{V}} \to 2^{\Sigma_\Lambda}$, 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 $s \in \Sigma^2_{\mathcal{V}}$, i.e., the set of allowed actions by $\mathcal{S}$ at any state $s \in \Sigma^2_{\mathcal{V}}$ is the intersection of the allowed actions by $\mathcal{S}_1$ and $\mathcal{S}_2$. Let $\phi_{\mathcal{S}_1}$ ($\phi_{\mathcal{S}_2}$) be a predicate encoding $\mathcal{S}_1$ ($\mathcal{S}_2$, respectively), i.e., for all $(s, \ell) \in \Sigma^2_{\mathcal{V}} \times \Sigma_\Lambda$, $(s, \ell) \models \phi_{\mathcal{S}_1}$ if and only if $\ell \in \mathcal{S}_1(s)$. The predicate $\phi_{\mathcal{S}}$ corresponding to $\mathcal{S}$ can be computed symbolically as $\phi_{\mathcal{S}} = \phi_{\mathcal{S}_1} \wedge \phi_{\mathcal{S}_2}$. 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 $\mathcal{S}$. Intuitively, $\mathcal{G}[\mathcal{S}]$ is the same as $\mathcal{G}$ but player-2's actions are restricted according to $\mathcal{S}$.

Solving games. To solve a game $(\mathcal{G}, \phi_{init}, \Phi)$ using bounded synthesis approach, we first obtain the safety game $(\mathcal{G}^\Phi, \phi^\Phi_{init})$ corresponding to Φ using the methods proposed in [18]. The game structure $\mathcal{G}^\Phi$ is then *composed* with $\mathcal{G}$. 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 $\tau^\otimes = \tau^1 \wedge \tau^2$. The safety game $(\mathcal{G} \otimes \mathcal{G}^\Phi, \phi_{init} \wedge \phi^\Phi_{init})$ is solved to determine the winner of the game and compute a winning strategy for player-2, if one exists.

Symbolic algorithms for solving the realizability and synthesis problems are based on the *controllable predecessor* operator [50]. The (player-2) controllable predecessor operator $CPre : 2^{\Sigma_{\mathcal{V}}} \to 2^{\Sigma_{\mathcal{V}}}$ maps a set $X \subseteq \Sigma_{\mathcal{V}}$ of states to the states from which player-2 can force the game into X in one step. Player-2 can force the game into X from a state $s \in \Sigma^1_{\mathcal{V}}$ if and only if for all available moves ℓ, all ℓ-successors of s are in X, and she can force the game into X from a state $s \in \Sigma^2_{\mathcal{V}}$ if and only if there is *some* available action ℓ such that all ℓ-successors of v are in X. Formally, let operator $Epre_\Lambda : 2^{\Sigma_{\mathcal{V}}} \to 2^{\Sigma_{\mathcal{V}}}$ maps a set $X \subseteq \Sigma_{\mathcal{V}}$ of states to the states for which there exists an action $\ell \in \Sigma_\Lambda$ such that all ℓ-successors belong to the set X, and is formally defined as follows:

$$Epre_\Lambda(X) = \{v \in \Sigma_{\mathcal{V}} \mid \exists \ell \in \Sigma_\Lambda \forall w \in \Sigma_{\mathcal{V}}.(v, \ell, w') \models \tau \to w \in X\}$$

Similarly, operator $Apre_\Lambda : 2^{\Sigma_{\mathcal{V}}} \to 2^{\Sigma_{\mathcal{V}}}$ maps a set $X \subseteq \Sigma_{\mathcal{V}}$ of states to the states for which for all actions $\ell \in \Lambda$ all ℓ-successors belong to the set X, and is formally defined as

$$Apre_\Lambda(X) = \{v \in \Sigma_{\mathcal{V}} \mid \forall \ell \in \Sigma_\Lambda \forall w \in \Sigma_{\mathcal{V}}.(v, \ell, w') \models \tau \to w \in X\}$$

The controllable predecessor is formally defined as

$$CPre(X) = (t = 2 \wedge Epre_\Lambda(X)) \vee (t = 1 \wedge Apre_\Lambda(X)).$$

The set of winning states from which player-2 can avoid dead-end states in the safety game $(\mathcal{G}, \phi_{init})$ is the greatest fixed point $W = \nu X.CPre(X)$. Player-2 is winning in $(\mathcal{G}, \phi_{init})$ if $[\![\phi_{init}]\!] \subseteq W$, i.e., any initial state is winning. Algorithm 1 summarizes the steps for solving games using bounded-synthesis approach. The procedure **ToSafetyGame** takes an LTL formula as an input and returns its corresponding safety game [18,49]. The procedure **SolveSafetyGame** computes the maximally permissive strategy for player-2 in the safety game $(\mathcal{G} \otimes \mathcal{G}^\Phi, \phi_{init} \wedge \phi^\Phi_{init})$ [18], if one exists.

Algorithm 1: SolveGame.

```
Input: a game $(\mathcal{G}, \phi_{init}, \Phi)$
Output: a winning strategy $\mathcal{S}$, if one exists
1 $(\mathcal{G}^{\Phi}, \phi^{\Phi}_{init}) :=$ ToSafetyGame$(\Phi)$;
2 $\mathcal{S} :=$ SolveSafetyGame$(\mathcal{G} \otimes \mathcal{G}^{\Phi}, \phi_{init} \wedge \phi^{\Phi}_{init})$;
3 return $\mathcal{S}$;
```

3. Dynamically-decoupled multi-agent systems

In this section we describe how we model dynamically-decoupled multi-agent systems and formally state the problem that is considered in this paper. 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 is 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 variables encoding actions of 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 $s \in \Sigma_{\mathcal{I}\cup\mathcal{O}}$ 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 $i \in \Sigma_{\mathcal{I}}$, $o \in \Sigma_{\mathcal{O}}$, $\ell \in \Sigma_{\Lambda}$, and $o' \in \Sigma_{\mathcal{O}'}$ indicates 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 any state s' where $s'_{|\mathcal{O}'} = o'$ is a possible successor. A *perfect agent* is an agent with $\mathcal{OBS} = \mathcal{I} \cup \mathcal{O}$ and $\gamma(s) = \{s\}$ for all $s \in \Sigma_{\mathcal{I}\cup\mathcal{O}}$, i.e., a perfect agent can observe the valuation over its input and output variables perfectly. We omit $(\mathcal{OBS}, \gamma)$ in the description of perfect agents.

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 $1 \le i \le n$. 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., $\forall 1 \le i \le n.\ \mathcal{O}_i \cap \mathcal{O}_j = \emptyset$, and the set of input variables $\mathcal{I}_i$ for each agent $\mathtt{a}_i \in \mathcal{M}$ is the set of variables controlled by other agents, i.e., $\mathcal{I}_i = \mathcal{V} \backslash \mathcal{O}_i$. We further make some simplifying assumptions. 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. Moreover, we assume that the observation variables for controlled agents are pairwise disjoint, i.e., $\forall 1 \le i \le n.\ \mathcal{OBS}_i \cap \mathcal{OBS}_j = \emptyset$, 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 multi-player games of incomplete information are undecidable in general [9].

In this paper we focus on a special class of multi-agent systems where all agents are *local*. An agent $\mathtt{a} = (\mathtt{type}, \mathcal{I}, \mathcal{O}, \Lambda, \tau, \mathcal{OBS}, \gamma)$ 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 $v \in \mathcal{I}$. We say a multi-agent system $\mathcal{M} = \{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_n\}$ is *dynamically-decoupled* if all agents $\mathtt{a} \in \mathcal{M}$ are local. Intuitively, agents in a dynamically-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 that 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 [12]. 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 also 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 dynamically-decoupled multi-agent system $\mathcal{M} = \mathcal{M}^{\mathtt{u}} \biguplus \mathcal{M}^{\mathtt{c}}$ partitioned into a set $\mathcal{M}^{\mathtt{u}} = \{\mathtt{u}_1, \cdots, \mathtt{u}_m\}$ of uncontrolled agents and a set $\mathcal{M}^{\mathtt{c}} = \{\mathtt{c}_1, \cdots, \mathtt{c}_n\}$ of controlled agents, the turn-based game structure $\mathcal{G}^{\mathcal{M}}$ induced by $\mathcal{M}$ is defined as $\mathcal{G}^{\mathcal{M}} = (\mathcal{V}, \Lambda, \tau, \mathcal{OBS}, \gamma)$ where $\mathcal{V} = \{t\} \cup \bigcup_{\mathtt{a}\in\mathcal{M}} \mathcal{O}_{\mathtt{a}}$ is the set of all variables in $\mathcal{M}$ with t as a turn variable, $\Lambda = \bigcup_{\mathtt{a}\in\mathcal{M}} \Lambda_{\mathtt{a}}$ is the set of action variables, $\mathcal{OBS} = \bigcup_{\mathtt{c}\in\mathcal{M}^{\mathtt{c}}} \mathcal{OBS}_{\mathtt{c}}$ is the set of all observation variables of controlled agents,[2] and τ and γ are defined as follows:

$$\tau = \tau_e \vee \tau_s,$$
$$\tau_e = t = 1 \wedge t' = 2 \wedge \bigwedge_{\mathtt{u}\in\mathcal{M}^{\mathtt{u}}} \tau_{\mathtt{u}} \wedge \bigwedge_{\mathtt{c}\in\mathcal{M}^{\mathtt{c}}} Same(\mathcal{O}_{\mathtt{c}}, \mathcal{O}'_{\mathtt{c}}),$$

[2] For uncontrolled agents, it does not matter if they are perfect or imperfect since spoiling strategies in turn-based games can even be blind [34].

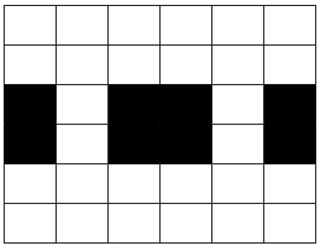


Fig. 1. A grid-world with static obstacles.

$$\tau_s = t = 2 \wedge t' = 1 \wedge \bigwedge_{\mathrm{c} \in \mathcal{M}^{\mathrm{c}}} \tau_{\mathrm{c}} \wedge \bigwedge_{\mathrm{u} \in \mathcal{M}^{\mathrm{u}}} Same(\mathcal{O}_{\mathrm{u}}, \mathcal{O}'_{\mathrm{u}}), \text{ and}$$

$$\gamma = \bigwedge_{\mathrm{c} \in \mathcal{M}^{\mathrm{c}}} \gamma_{\mathrm{c}}$$

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 2. Let R_1 and R_2 be two robots in an $n \times n$ grid-world similar to the one shown in Fig. 1. Assume R_1 is an uncontrolled robot, whereas R_2 is 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 $\Sigma_{\Lambda_i} = \{up_i, down_i, right_i, left_i\}$. Furthermore, assume that R_2 has imperfect sensors and can only observe R_1 if 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.[3] 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 $\ell \in \Sigma_{\Lambda_i}$ 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,

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

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. The observation function γ_2 can be symbolically encoded as

$$\gamma_2 = \bigvee_{o \in \Sigma_{\mathcal{OBS}}} (o \wedge \phi_{\gamma(o)})$$

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

[3] 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.

Algorithm 2: Compositional controller synthesis.

```
Input: A dynamically-decoupled multi-agent system $\mathcal{M} = \{u_1, \cdots, u_m, c_1, \cdots, c_n\}$, $\phi_{init}$ specifying an initial state, and an objective $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_k$
Output: A set of strategies $(s_1, \cdots, s_n)$ one for each controlled agent, if one exists
1  /* Decompose the problem */
2  for all $\Phi_i, 1 \le i \le k$ do
3      $\mathcal{INV}_i :=$ Involved($\Phi_i$);
4      $\mathcal{G}_i :=$ CreateGameStructure($\mathcal{INV}_i$);
5      $\mathcal{X}_i := \bigcup_{a \in \mathcal{INV}_i} \mathcal{O}_a$; /*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      $(\mathcal{G}^{\Phi_i}, \phi^{\Phi_i}_{init}) :=$ ToSafetyGame($\Phi_i$);
9      $(\mathcal{G}^d_i, \phi^{d_i}_{init}) := (\mathcal{G}^K_i \otimes \mathcal{G}^{\Phi_i}, \phi^i_{init} \wedge \phi^{\Phi_i}_{init})$;
10 /*Compositional synthesis*/
11 while true do
12     for $i = 1 \cdots k$ do
13         $\mathcal{S}^d_i :=$ SolveSafetyGame($\mathcal{G}^d_i, \phi^{d_i}_{init}$);
14     $\mathcal{S} := \bigotimes_{i=1}^k \mathcal{S}^d_i$; /* compose the strategies */
15     for $i = 1 \cdots k$ do
16         Let $\mathcal{Y}_i = \mathcal{V}^d_i \cup \Lambda^d_i$ be the set of variables and actions in $\mathcal{G}^d_i$;
17         $\mathcal{C}_i :=$ Project($\mathcal{S}, \mathcal{Y}_i$); /* project the strategies */
18     if $\forall 1 \le i \le k, \mathcal{S}^d_i = \mathcal{C}_i$ then
19         break; /* a fixed point is reached over strategies */
20     for $i = 1 \cdots k$ do
21         $\mathcal{G}^d_i := \mathcal{G}^d_i[\mathcal{C}_i]$; /* Restrict the subgames for the next iteration */
22 $(s_1, \cdots, s_n)$ :=Extract($\mathcal{S}$);
23 return $(s_1, \cdots, s_n)$;
```

$$\mathcal{V} = \{t\} \cup \mathcal{O}_1 \cup \mathcal{O}_2,$$
$$\Lambda = \Lambda_1 \cup \Lambda_2,$$
$$\tau = \tau_e \vee \tau_s,$$
$$\tau_e = t = 1 \wedge t' = 2 \wedge \tau_1 \wedge Same(\mathcal{O}_2, \mathcal{O}'_2),$$
$$\tau_s = t = 2 \wedge t' = 1 \wedge \tau_2 \wedge Same(\mathcal{O}_1, \mathcal{O}'_1),$$
$$\mathcal{OBS} = \mathcal{OBS}_2, and$$
$$\gamma = \gamma_2. \qquad \square$$

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

Problem 1. Given a dynamically-decoupled multi-agent system $\mathcal{M} = \mathcal{M}^u \uplus \mathcal{M}^c$ partitioned into uncontrolled $\mathcal{M}^u = \{u_1, \cdots, u_m\}$ and controlled agents $\mathcal{M}^c = \{c_1, \cdots, c_n\}$, a predicate ϕ_{init} specifying an initial state, and an objective $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_k$ as conjunction of $k \ge 1$ LTL formulas Φ_i, compute strategies $s_1, \cdots, s_n$ for controlled agents such that the strategy $s = s_1 \otimes \cdots \otimes s_n$ 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 2 summarizes the steps for compositional synthesis of strategies for controlled agents in a dynamically-decoupled 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. 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 2 in more detail.

4.1. Decomposition of the synthesis problem

The synthesis problem is decomposed into subproblems in lines 2–9 of Algorithm 2. The main idea behind this 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. Each subproblem corresponds to 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, $1 \le i \le k$, Algorithm 2 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 conjunct's formula, i.e., $\textbf{Involved}(\Phi_i) = \{a \in \mathcal{M} \mid \mathcal{O}_a \cap \mathcal{V}_{\Phi_i} \neq \emptyset\}$. 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, $\textbf{Project}(\phi, \mathcal{X}) = \{s_{|\mathcal{X}} \mid s \in \Sigma_{\mathcal{V}_\phi} \text{ and } s \models \phi\}$.[4]

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 agents that can observe the state of the game perfectly at any time-step. Finally, the objective Φ_i is transformed into a safety game using the algorithms in [18,49] and composed with $\mathcal{G}^K_i$ to obtain a safety game $(\mathcal{G}^d_i, \phi^{d_i}_{init})$. The result of decomposition phase is k safety games $\left\{(\mathcal{G}^d_1, \phi^{d_1}_{init}), \cdots, (\mathcal{G}^d_k, \phi^{d_k}_{init})\right\}$ that form the subproblems for the compositional synthesis phase.

Example 3. 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. □

Remark 1. The decomposition method used here is neither the only way to decompose the problem, nor necessarily the optimal. More efficient decomposition techniques can be used to obtain quicker convergence in Algorithm 2 for example by different grouping of conjuncts. Nevertheless, the decomposition technique explained above is simple and proved effective in our experiments.

4.2. Compositional synthesis

The safety games obtained in the decomposition phase are compositionally solved in lines 10–21 of Algorithm 2. At each iteration of the main loop, the subproblems $(\mathcal{G}^d_i, \phi^{d_i}_{init})$ are solved, and a maximally permissive strategy $\mathcal{S}^d_i$ is computed for them, if one exists. Intuitively, maximally permissive strategies $\mathcal{S}^d_i$'s allow *all* winning actions for involved controlled agents at each winning state, restricting the agents as little as possible and letting them to choose from a set of possible actions, in contrast to a single action permitted in deterministic strategies. In other words, the agents are not yet "committed" to any action. This *least-commitment* principle captured naturally by maximally permissive strategies is useful when composing the strategies: at each global state, the controlled agents can choose from those joint actions that are allowed by all maximally permissive strategies.

Computed maximally permissive strategies are composed in line 14 of Algorithm 2 to obtain a strategy $\mathcal{S}$ for the whole system. The strategy $\mathcal{S}$ is then projected back to sub-games, and it is checked whether all the projected strategies are equivalent to the strategies computed for the subproblems. If that is the case, the main loop terminates and $\mathcal{S}$ is winning for the game $(\mathcal{G}^d, \phi^d_{init})$ where $(\mathcal{G}^d, \phi^d_{init})$ 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 subgame is restricted by the computed projection, and the process is repeated. The loop terminates either if a subproblem becomes unrealizable at some iteration, 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 $\mathcal{V}^\otimes = \bigcup_{i=1}^k \mathcal{V}_{\mathcal{G}^d_i}$ be the set of all variables used to encode the game structures $\mathcal{G}^d_i$, and $\Lambda^c = \Lambda_{c_1} \cup \cdots \cup \Lambda_{c_n}$ be the set of variables encoding actions of the controlled agents. Once a permissive strategy $\mathcal{S} : \Sigma^2_{\mathcal{V}^\otimes} \to 2^{\Sigma_{\Lambda^c}}$ is computed, a winning strategy $\mathtt{S}_d : \Sigma^2_{\mathcal{V}^\otimes} \to \Sigma_{\Lambda^c}$ is obtained from $\mathcal{S}$ by restricting the non-deterministic action choices of the controlled agents to a single action at each state $s \in \Sigma^2_{\mathcal{V}^\otimes}$ where $\mathcal{S}(s)$ is non-empty. The strategy $\mathtt{S}_d$ is then decomposed into strategies

[4] The procedure **Project** can be implemented symbolically using BDD operations by existentially quantifying the variables $\mathcal{Y} = \mathcal{V}_\phi \backslash \mathcal{X}$ in ϕ, i.e., $\textbf{Project}(\phi, \mathcal{X}) = \exists \mathcal{Y}.\phi$.

$\mathtt{S}_1 : \Sigma^2_{\mathcal{V}^\otimes} \to \Sigma_{\Lambda_{c_1}}, \cdots, \mathtt{S}_n : \Sigma^2_{\mathcal{V}^\otimes} \to \Sigma_{\Lambda_{c_n}}$ for the agents simply by projecting the actions with respect to their corresponding agents. Formally, for each $s \in \Sigma^2_{\mathcal{V}^\otimes}$ such that $\mathcal{S}(s) \neq \emptyset$, let $\mathtt{S}_d(s) = \sigma \in \mathcal{S}(s)$ where $\sigma = (\sigma_1, \cdots, \sigma_n) \in \Sigma_{\Lambda^c}$ is an arbitrary action chosen from possible actions permitted by $\mathcal{S}$ in state s. Individual 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 and completeness of Algorithm 2.

Theorem 1. *Algorithm 2 is sound and complete.*

Proof. See Appendix B. □

Remark 2. Filiot et al. in [18] show that bounded synthesis is complete by proving the existence of a sufficiently large bound. Completeness of Algorithm 2 is based on completeness of bounded synthesis. Note that in practice, the required bound is rather high and instead an incremental approach is used for synthesis.

Remark 3. Algorithm 2 is different from the compositional algorithm proposed in [18] in two ways. First, it composes maximally permissive strategies in contrast to composing game structures as proposed in [18]. The advantage is that strategies usually have more compact symbolic representations compared to game structures. That is because 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. Second, in the compositional algorithm in [18], 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 the composite game resulted in a poor performance, often worse than the centralized algorithm. Algorithm 2 removes this bottleneck as it is not required in our setting. This leads to a significant improvement in performance since image and pre-image computations are typically the most expensive operations performed by symbolic algorithms [51].

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., [11–13]). Consider a square grid-world similar to the one depicted in Fig. 1 where some static obstacles are positioned in the middle rows to create two narrow corridors. 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 using 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. For each agent $\mathtt{a} \in \mathcal{M}$, let $(x^{\mathtt{a}}, y^{\mathtt{a}})$ denote its position on the grid-world. Assume **SO** is the set of static obstacles. We denote the position of each static obstacle $\mathbf{o} \in \mathbf{SO}$ by $(x^{\mathbf{o}}, y^{\mathbf{o}})$. We consider the following objectives:

- Collision avoidance (Φ_1): controlled robots must avoid collision with static obstacles and other robots. This can be specified as $\Phi_1 = \Phi_{\mathtt{c}\mathbf{o}} \wedge \Phi_{\mathtt{cu}} \wedge \Phi_{\mathtt{cc}}$ where

$$\Phi_{\mathtt{c}\mathbf{o}} = \bigwedge_{\mathtt{c}\in\mathcal{M}^c} \bigwedge_{\mathbf{o}\in\mathbf{SO}} \Box(\neg(x^{\mathtt{c}} \leftrightarrow x^{\mathbf{o}} \wedge y^{\mathtt{c}} \leftrightarrow y^{\mathbf{o}})),$$

$$\Phi_{\mathtt{cu}} = \bigwedge_{\mathtt{c}\in\mathcal{M}^c} \bigwedge_{\mathtt{u}\in\mathcal{M}^{\mathtt{u}}} \Box(\neg(x^{\mathtt{c}} \leftrightarrow x^{\mathtt{u}} \wedge y^{\mathtt{c}} \leftrightarrow y^{\mathtt{u}})), \text{ and}$$

$$\Phi_{\mathtt{cc}} = \bigwedge_{\mathtt{c}_i\in\mathcal{M}^c} \bigwedge_{\mathtt{c}_j\in\mathcal{M}^c, j>i} \Box(\neg(x^{\mathtt{c}_i} \leftrightarrow x^{\mathtt{c}_j} \wedge y^{\mathtt{c}_i} \leftrightarrow y^{\mathtt{c}_j})).$$

  Intuitively, $\Phi_{\mathtt{c}\mathbf{o}}$ says controlled robots must never occupy cells where static obstacles are, $\Phi_{\mathtt{cu}}$ specifies that no controlled robot must be at the same cell where an uncontrolled robot is, and $\Phi_{\mathtt{cc}}$ requires that there must be no collision between the controlled robots.
- Formation maintenance (Φ_2): 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 $1 \leq i < n$. Formally,

$$\Phi_2 = \bigwedge_{1\leq i<n} \Box(x^{\mathtt{c}_i} \leftrightarrow x^{\mathtt{c}_{i+1}} \vee y^{\mathtt{c}_i} \leftrightarrow y^{\mathtt{c}_{i+1}})$$

- Bounded reachability (Φ_3): controlled robots must reach the bottom row in a pre-specified number of steps k. This can be specified as

$$\Phi_3 = \bigwedge_{\mathtt{c}\in\mathcal{M}^c} \Diamond(y^{\mathtt{c}} \leftrightarrow \mathbf{bottom\_row})$$

  The bound k is used to associate a safety game for each conjunct of Φ_3 as explained in Section 2.

Table 1
Evaluation of approaches on a robot motion planning case study with perfect agents.

Ex. #	$\|\mathcal{M}^u\|$	$\|\mathcal{M}^c\|$	Objective	Size	$\|\mathcal{V}\|$	Centralized		Compositional	
						time	mem (MB)	time	mem (MB)
5	1	1	Φ_1	64×64	52	**72 ms**	6.6	105 ms	6.6
6	1	1	Φ_1	128×128	60	**93 ms**	6.6	101 ms	6.6
39	1	2	Φ_{13}	16×16	79	14.9 min	365.5	**4.2 s**	**19.3**
40	1	2	Φ_{13}	32×32	95	mem out	mem out	**34.4 s**	**50.8**
43	1	2	Φ	16×16	79	400.3 s	239.7	**5.1 s**	**19.4**
44	1	2	Φ	32×32	95	155.8 min	1209	**33.1 s**	**38.3**
47	1	3	Φ_1	16×16	74	38.6 min	1391.5	**181.9 s**	**201.8**
51	1	3	Φ_{12}	16×16	74	9.6 min	648.7	**168.1 s**	**271**
53	1	3	Φ_{13}	4×4	66	22 s	50.8	**0.8 s**	**6.8**
54	1	3	Φ_{13}	8×8	88	mem out	mem out	**98.4 s**	**101.2**
56	1	3	Φ	8×8	88	88.9 min	1227.8	**28.4 s**	**44.5**
82	2	1	Φ	8×8	51	106.4 s	322	**33 ms**	**6.6**
86	2	1	Φ	128×128	107	mem out	mem out	**3.5 s**	**6.7**
93	2	2	Φ	4×4	56	3.2 s	19.4	**201 ms**	**6.6**
94	2	2	Φ	8×8	76	10.6 min	460	**14.4 s**	**19.4**
96	2	3	Φ_1	8×8	71	53.8 min	1423	**421.4 s**	**264.8**
98	2	3	Φ_{12}	8×8	71	14.2 min	642.4	**481.3 s**	**277.4**
99	2	3	Φ_{13}	4×4	75	19.1 min	497.8	**8.4 s**	**25.9**
100	2	3	Φ_{13}	8×8	101	mem out	mem out	**30.2 min**	**800.2**
102	2	3	Φ	8×8	101	mem out	mem out	**12.7 min**	**302.6**

We consider two settings. First we assume all agents are perfect and 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 the setting described in Example 2.

We apply two different methods to synthesize strategies for the agents. In the *Centralized* method, first a game structure for the whole system is obtained, and then a winning strategy is computed with respect to the considered objective. In the *Compositional* approach, the strategy is computed compositionally using Algorithm 2. We implemented the algorithms in Java using the BDD package JDD.[5] The experiments are performed on an Intel core i7 3.40 GHz machine with 16 GB memory. In our experiments, we vary the number of uncontrolled and controlled agents, size of the grid-world, and the objective of the system[6] as shown in Tables 1 and 2. The columns show the experiment number, 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 (in MB) for different approaches, respectively. Furthermore, we define $\Phi_{12} = \Phi_1 \wedge \Phi_2$, $\Phi_{13} = \Phi_1 \wedge \Phi_3$, and $\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3$.

Multi-agent systems with perfect agents. Table 1 shows some of our experimental results for the setting where all agents are perfect.[7] 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 centralized 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 subproblems 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. Fig. 2 shows the computation time and memory usage in some of our experiments (see Table 1 for details of the experiments) where both centralized and compositional algorithms successfully computed strategies.

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 alternative. 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. Fig. 3 shows the computation time and memory usage in some of our experiments (see Table 2 for details of the experiments) where both centralized and compositional algorithms successfully computed strategies.

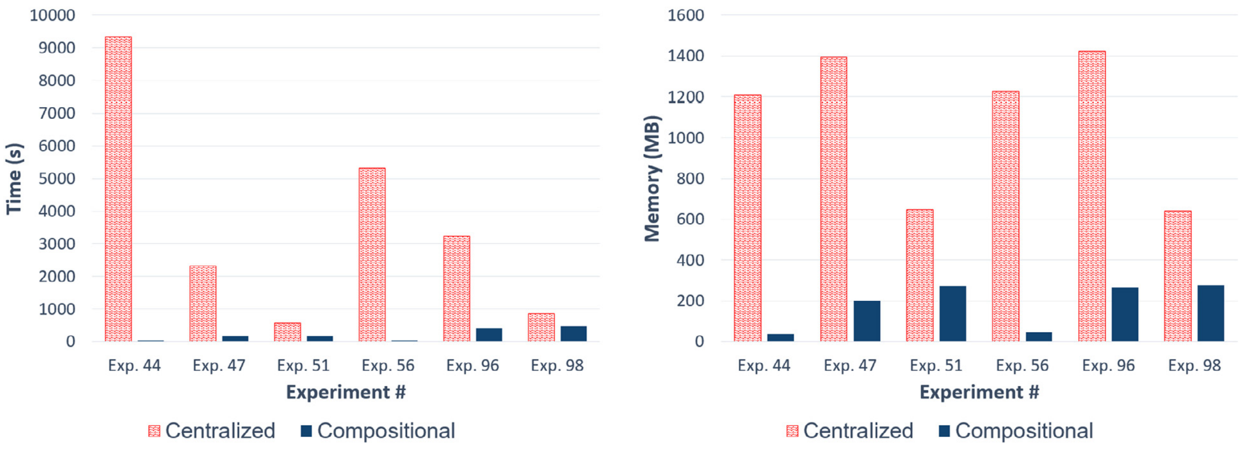
[5] http://javaddlib.sourceforge.net/jdd/index.html.

[6] JDD does not support dynamic variable reordering [52]. In our experiments, we fixed a variable ordering that had the best performance for the centralized approach and used the same variable ordering for the compositional algorithm.

[7] More experimental data for both perfect and imperfect set-ups is provided in [53].

Table 2
Evaluation of approaches on a robot motion planning case study with imperfect agents.

Ex. #	$\lvert\mathcal{M}^u\rvert$	$\lvert\mathcal{M}^c\rvert$	Objective	Size	$\lvert\mathcal{V}\rvert$	Centralized		Compositional	
						time	mem (MB)	time	mem (MB)
137	1	2	Φ_{12}	4×4	127	1.7 s	6.7	**0.6 s**	6.7
139	1	2	Φ_{12}	6×6	235	28.6 s	31.9	**10.2 s**	**19.3**
141	1	2	Φ_{12}	8×8	235	229.7 s	126.6	**95 s**	**57.1**
142	1	2	Φ_{13}	4×4	143	1.4 s	6.7	**303 ms**	6.7
144	1	2	Φ_{13}	6×6	255	38.2 s	57.1	**5 s**	**13**
145	1	2	Φ_{13}	7×7	255	159.5 s	132.6	**16.7 s**	**25.6**
146	1	2	Φ_{13}	8×8	255	8.9 min	252.2	**38.3 s**	**51**
147	1	2	Φ	4×4	143	2.3 s	6.7	**0.7 s**	6.7
149	1	2	Φ	6×6	255	46.2 s	50.8	**10 s**	**19.3**
151	1	2	Φ	8×8	255	344.5 s	202.1	**129.9 s**	**57.1**
152	1	2	Φ_1	9×9	375	27.8 min	390.7	**113.2 s**	**82.3**
154	1	2	Φ_{12}	9×9	375	time out	time out	**306 s**	**94.9**
155	1	2	Φ_{12}	10×10	375	time out	time out	**9.7 min**	**176.7**
156	1	2	Φ_{13}	9×9	395	time out	time out	**114.9 s**	**88.6**
157	1	2	Φ_{13}	10×10	395	time out	time out	**279.9 s**	**157.8**
158	1	2	Φ	9×9	395	time out	time out	**309.9 s**	**101.2**
159	1	2	Φ	10×10	395	time out	time out	**9.6 min**	**176.7**
160	1	3	Φ_1	4×4	186	144.3 s	69.7	**0.9 s**	**6.7**
162	1	3	Φ_1	6×6	346	time out	time out	**17.7 s**	**38.2**
164	1	3	Φ_1	8×8	346	time out	time out	**190.9 s**	**176.7**
166	1	3	Φ_1	10×10	554	time out	time out	**24.6 min**	**730.6**
174	1	3	Φ_{13}	4×4	210	265.8 s	214.5	**0.9 s**	**6.7**
176	1	3	Φ_{13}	6×6	376	time out	time out	**49.2 s**	**57.1**
178	1	3	Φ_{13}	8×8	376	time out	time out	**483.9 s**	**214.5**
179	1	3	Φ_{13}	9×9	584	time out	time out	**31.7 min**	**441.1**
181	1	3	Φ	6×6	376	time out	time out	**36 s**	**50.8**
183	1	3	Φ	8×8	376	time out	time out	**343.4 s**	**201.9**
185	1	3	Φ	10×10	584	time out	time out	**39.6 min**	**774.7**



(a) Time (b) Memory

Fig. 2. Comparison of centralized and compositional approaches on a robot motion planning case study with perfect agents.

6. Conclusions and future work

We considered the problem of automated synthesis of controllers for multi-agent systems from high-level temporal logic specifications. The key insight was to consider more restricted yet practically useful subclasses of the general problem. The overall theme of the solution approach was to take advantage of the existing structure in systems in order to decompose the synthesis problem into smaller and more manageable subproblems, and to achieve more efficient synthesis algorithms through compositional synthesis techniques. We proposed a framework for modular specification and controller synthesis for dynamically-decoupled multi-agent systems. We showed that, by taking advantage of the structure in system to compositionally synthesize controllers, and by representing and exploring the state space symbolically, we can achieve better scalability. Our preliminary results show the potential of reactive synthesis as planning algorithms in the presence of dynamically changing and adversarial environment.

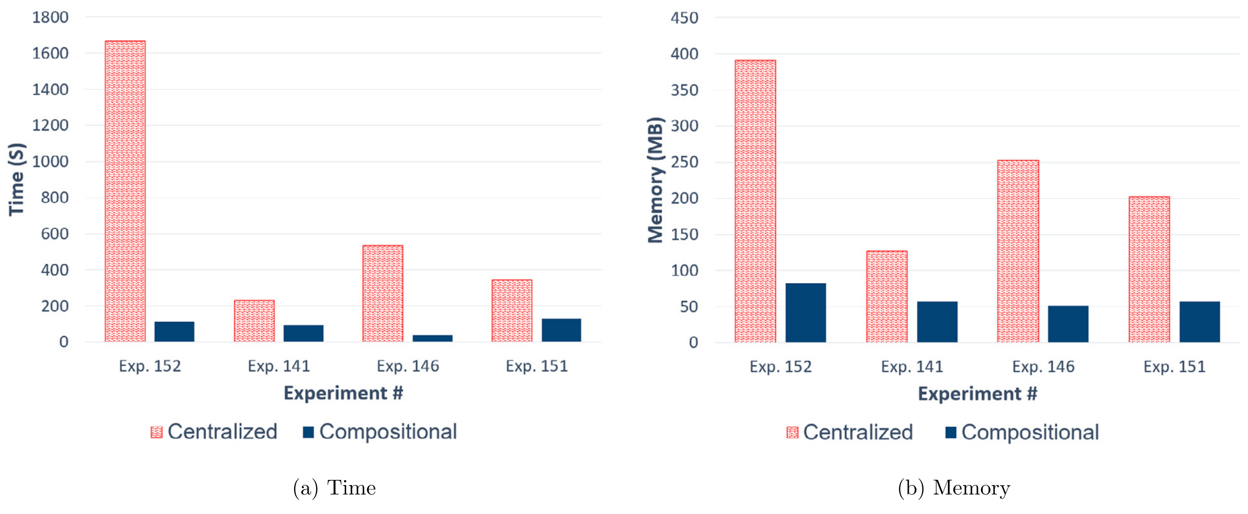


Fig. 3. Comparison of centralized and compositional approaches on a robot motion planning case study with imperfect agents.

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 observation functions can enable us to handle systems with imperfect agents of larger size. Besides, in multi-agent systems we consider, the agents are either cooperative or adversarial. An interesting future direction is to extend this model and allow the user to specify which agents can cooperate to fulfill a specific objective.

Acknowledgments

This research was partially supported by awards NSF Expeditions in Computing CCF 1138996, AFRL $FA8650$-15-C-2546, AFRL 8650-16-C-2610 and DARPA $W911NF$-16-1-0001.

Appendix A. Constructing the knowledge game structure

A turn-based game structure of imperfect information is first transformed into its corresponding knowledge game structure through subset construction. Given an initial state, we construct the knowledge game structure symbolically, and only part of it that is *reachable* from the initial state. Intuitively, only "relevant" part of the knowledge game structure is constructed. Algorithm 3 summarizes the steps for constructing the knowledge game structure. Queues Q_1 and Q_2 store sets of states in the knowledge game structure that are reachable from the initial state by player-1 and player-2, respectively, and their outgoing transitions are yet to be computed. Since we assume player-1 starts the game, Q_1 initially contains the initial state, and Q_2 is empty. Let k be a bounded integer variable that is used to encode states of the knowledge game structure, and $\mathcal{H}$ be a mapping that maps knowledge game states to their corresponding sets of states. Note that since the domain of k is bounded, it can be encoded using a finite number of Boolean variables. To keep the notation simple, we use its bounded integer representation.

Sets Vis_1 and Vis_2 keep track of sets of states for player-1 and player-2, respectively, that are already visited during the search and for which the outgoing transitions in the knowledge game structure is computed. Transition relations τ_1^K and τ_2^K represent player-1 and player-2's transition relations in the knowledge game structure, and initially are set to `false`. Algorithm 3 iteratively chooses a non-empty queue and picks a set of states $[\![\phi]\!]$ to compute its corresponding transitions in the knowledge game structure. ϕ is marked as visited in the corresponding set. The post-image of ϕ, i.e., the set of states that can be reached in game structure $\mathcal{G}$ in one step from any state $s \in [\![\phi]\!]$, is computed next. The transition relation of the corresponding player is updated using the procedure **UpdateKGSTransitionRelation** as shown in Algorithm 4.

Procedure **UpdateKGSTransitionRelation** first obtains the "relevant" observations $Obs \subseteq \Sigma_{\mathcal{OBS}}$ by partitioning the set of states $[\![\psi]\!]$ using the observation function γ. Then for each observation $o \in Obs$, the knowledge game state $b \in \Sigma_k$ corresponding to $\alpha = \gamma(o) \wedge \psi$ is obtained. If b is undefined, a new state for the knowledge game structure is defined by mapping α to the current value of the counter and then incrementing the counter. Next, if the set of states α has not already been explored, it is added to Q_1 or Q_2 (depending on what player's transition relation is being updated) to be processed later. Finally, the transition relation of the player-p is updated by adding a transition between $\beta = \mathcal{H}^{-1}(\phi)$. Note that since player-1 actions cannot be observed by player-2, actions do not appear in the transition relation formula of player-1.

Algorithm 3: CreateKnowledgeGameStructure.

```
Input: a turn-based game structure $\mathcal{G} = (\mathcal{V}, \Lambda, \tau, \mathcal{OBS}, \gamma)$ of imperfect information and a predicate $\phi_{init}$ specifying an initial state
Output: the knowledge game structure $\mathcal{G}^K$ for $\mathcal{G}$
1  Let $Q_1$ and $Q_2$ be two initially empty queues;
2  $Q_1$.Enqueue($\phi_{init}$);
3  Let $k$ be a variable with domain $\Sigma_k \in [0..2^{2^{|\mathcal{V}|}}]$;
4  $counter := 0$;
5  Let $\mathcal{H} : \Sigma_k \to 2^{\Sigma_\mathcal{V}}$ be a mapping that maps knowledge game states to sets of states. Initially $\mathcal{H}(a)$ is undefined for any $a \in \Sigma_k$;
6  $Vis_1 := \emptyset$ /*sets of states visited by player-1, initially empty*/;
7  $Vis_2 := \emptyset$ /*sets of states visited by player-2, initially empty*/;
8  $\tau_1^K :=$ false /*player-1 transition relation in $\mathcal{G}^K$*/;
9  $\tau_2^K :=$ false /*player-2 transition relation in $\mathcal{G}^K$*/;
10 while $Q_1$ or $Q_2$ is not empty do
11     if $Q_1$ is not empty then
12         $\phi := Q_1$.Dequeue();
13         $Vis_1 := Vis_1 \cup \{\phi\}$;
14         Let $\psi$ be a predicate specifying the set $\bigcup_{\ell \in \Lambda} Post_\ell^{\mathcal{G}}([\![\phi]\!])$;
15         UpdateKGSTransitionRelation($\phi$, $\psi$, true, player-1);
16     if $Q_2$ is not empty then
17         $\phi := Q_2$.Dequeue();
18         $Vis_2 := Vis_2 \cup \{\phi\}$;
19         for each action $\ell \in actions$ do
20             Let $\psi$ be a predicate specifying the set $Post_\ell^{\mathcal{G}}(\phi)$;
21             UpdateKGSTransitionRelation($\phi$, $\psi$, $\ell$, player-2);
22 Let $t_K$ be a variable with domain $\Sigma_{t_K} = \{1, 2\}$;
23 Define $\tau^K = (t_K = 1 \wedge t'_K = 2 \wedge \tau_1^K) \vee (t_K = 2 \wedge t'_K = 1 \wedge \tau_2^K)$;
24 return $\mathcal{G}^K = (\{k, t_k\}, \Lambda, \tau^K)$;
```

Algorithm 4: UpdateKGSTransitionRelation.

```
Input: a predicate $\phi$ over $\mathcal{V}$ specifying a set of states, a predicate $\psi$ over variables $\mathcal{V}$ specifying a set of next states, an action $\sigma \in \Lambda$, and
       player-$p \in$ {player-1, player-2} whose transition relation is being updated (assumes access to local variables of the invoker)
1  $Obs := \{o \in \Sigma_{\mathcal{OBS}} \mid \exists s \in \Sigma_\mathcal{V}.\ s \models \psi \wedge o = \gamma^{-1}(s)\}$;
2  for each observation $o \in Obs$ do
3      $\alpha := \gamma(o) \wedge \psi$;
4      $b := \mathcal{H}^{-1}(\alpha)$;
5      if $b$ is undefined then
6          $b := counter$;
7          $counter := counter + 1$;
8          $\mathcal{H}(b) := \alpha$;
9      if $p = player$-1 then
10         if $\alpha \notin Vis_2$ then
11             $Q_2$.Enqueue($\alpha$);
12     else
13         if $\alpha \notin Vis_1$ then
14             $Q_1 =$ Enqueue($\alpha$);
15     $\beta := \mathcal{H}^{-1}(\phi)$;
16     if $p = player$-1 then
17         $\tau_1^K := \tau_1^K \vee (\beta \wedge \alpha')$;
18     else
19         $\tau_2^K := \tau_2^K \vee (\beta \wedge \sigma \wedge \alpha')$;
```

Appendix B. Proof of Theorem 1

Proof. Note that Algorithm 2 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 all $s \in \Sigma_\mathcal{V}$, and by composing and projecting intermediate strategies, we obtain more restricted subgames. As the state space and available actions at any state is finite, eventually, either a subgame 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 2 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 $s \models \phi_{init}^{d_i}$ 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 $s_0 \models \phi_{init}^d$ where $\mathcal{G}^d = \bigotimes_{i=1}^k \mathcal{G}_i^d$ and $\phi_{init}^d = \bigwedge_{i=1}^k \phi_{init}^{d_i}$. 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 $s_0^i \models \phi_{init}^{d_i}$ 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 $(\mathcal{G}_i^d[\mathcal{S}_i^d], \phi_{init}^{d_i})$, 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 $(\mathcal{G}^d, \phi_{init}^d)$. Moreover, $\mathcal{S}^*$ is also winning with respect to the original game as $(\mathcal{G}^d, \phi_{init}^d)$ is the safety game associated with $(\mathcal{G}^\mathcal{M}, \phi_{init}, \Phi)$ [18]. It is easy to see that the set $(\mathtt{S}_1, \cdots, \mathtt{S}_n)$ of strategies extracted from $\mathcal{S}^*$ by Algorithm 2 is winning for the game $(\mathcal{G}^\mathcal{M}, \phi_{init}, \Phi)$.

We now show that Algorithm 2 is also complete, i.e., if there exists a winning strategy $\mathtt{S} = \bigotimes_{i=1}^n \mathtt{S}_i$ in the game $(\mathcal{G}^\mathcal{M}, \phi_{init}, \Phi)$ given as composition of strategies for controlled agents, then Algorithm 2 computes such a strategy. Let $\mathtt{S} : \Sigma_{\mathcal{V}^\otimes}^2 \to \Sigma_\Lambda$ be a winning strategy for the central safety game $(\mathcal{G}^d, \phi_{init}^d)$. Assume $\mathcal{C}_i = \mathbf{Project}(\mathtt{S}, \mathcal{Y}_i)$ is the projection of $\mathtt{S}$ into variables of the game structure $\mathcal{G}_i^d$ for $1 \leq i \leq k$.

Lemma 1. *$\mathcal{C}_i$ is the maximally permissive strategy in the safety game $(\mathcal{G}_i^d[\mathcal{C}_i], \phi_{init}^{d_i})$.*

Proof. We show that any run in $\mathcal{G}_i^d[\mathcal{C}_i]$ starting from the initial state $r_0 \models \phi_{init}^{d_i}$ is infinite and therefore winning. That is, the projection of the winning strategy $\mathtt{S}$ of the centralized game structure $\mathcal{G}^d = (\mathcal{V}^\otimes, \Lambda, \tau)$, is winning for the game structure $\mathcal{G}_i^d = (\mathcal{V}_i^d, \Lambda_i^d, \tau_i^d)$. Let $W \subseteq \Sigma_{\mathcal{V}^\otimes}$ be the set of winning states in the central safety game $(\mathcal{G}^d, \phi_{init}^d)$. Consider any run $\pi = r_0, r_1, r_2, \cdots$ in the game structure $\mathcal{G}_i^d[\mathcal{C}_i]$ where $r_j \in \Sigma_{\mathcal{V}_i^d}$ and $r_0 \models \phi_{init}^{d_i}$ for $j \geq 0$. We show that for any $r_j \in \Sigma_{\mathcal{V}_i^d}$, $j \geq 0$, there exists a state $s_j \in W$ such that $s_{|\mathcal{V}_i^d} = r_j$. It follows that no r_j is dead-end, i.e., there is at least one available action at r_j, and thus the run π is infinite (because otherwise the corresponding state s_j of $\mathcal{G}^d$ must also be dead-end which is a contradiction with s_j being a winning state).

Proof is by induction. For $r_0 \models \phi_{init}^{d_i}$, there exists $s_0 \in \Sigma_{\mathcal{V}^\otimes} \models \phi_{init}$ such that $s_{0_{|\mathcal{V}_i^d}} = r_0$ and $s_0 \in W$. That is because $(\mathcal{G}^d, \phi_{init}^d)$ is realizable, and therefore s_0 must be a winning state. For $j \geq 0$, let $\sigma_j \in \Sigma_{\Lambda_i^d}$ be an arbitrary action available at r_j in $\mathcal{G}_i^d$, and r_{j+1} a possible successor, i.e., $(r_j, \sigma_j, r_{j+1}) \models \tau_i^d$. There are two cases. If r_j is a player-1 state, there exists a state $s_j \in W$ where $s_{j_{|\mathcal{V}_i^d}} = r_j$ (inductive assumption) and $\ell_j \in \Sigma_\Lambda$ such that $\ell_{j_{|\Lambda_i^d}} = \sigma_j$. That is because the set of available actions at r for uncontrolled agents involved in $\mathcal{G}_i^d$ is same as the set of available actions for them at s in $\mathcal{G}^d$ since the agents are dynamically decoupled and actions of uncontrolled agents are not restricted during composition. Otherwise if r_j is a player-2 state, there exists a state $s_j \in W$ and $\ell \in \mathtt{S}(s_j)$ such that $\ell_{|\Lambda_i^d} = \sigma_j \in \mathcal{C}_i$. Given s_j and chosen actions of the agents ℓ and since the agents are dynamically-decoupled, there must exist $s_{j+1} \in W$ such that $s_{j+1_{|\mathcal{V}_i^d}} = r_{j+1}$. □

The following lemma shows that composing the permissive strategies $\mathcal{C}_i$, $1 \leq i \leq k$ and then projecting them with respect to the variables and actions of the sub-games will lead to the same permissive strategy $\mathcal{C}_i$ for each sub-game, i.e., $\{\mathcal{C}_i \mid 1 \leq i \leq k\}$ is a fixed point over strategies.

Lemma 2. ***Project***$(\bigotimes_{i=1}^k \mathcal{C}_i, \mathcal{Y}_i) = \mathcal{C}_i$.

Proof. We prove the lemma for $k = 2$. Extension to the general case is straightforward. We have $\mathcal{V}^\otimes = \mathcal{V}_{\mathcal{G}_1^d} \cup \mathcal{V}_{\mathcal{G}_2^d}$. We define three (possibly empty) sets $\mathcal{X}_1 = \mathcal{V}_{\mathcal{G}_1^d} \backslash \mathcal{V}_{\mathcal{G}_2^d}$, $\mathcal{X}_2 = \mathcal{V}_{\mathcal{G}_2^d} \backslash \mathcal{V}_{\mathcal{G}_1^d}$, and $\mathcal{X}_3 = \mathcal{V}_{\mathcal{G}_1^d} \cap \mathcal{V}_{\mathcal{G}_2^d}$. That is, $\mathcal{X}_1$ is the set of variables that only appear in $\mathcal{G}_1^d$, $\mathcal{X}_2$ is the set of variables only appearing in $\mathcal{G}_2^d$, and $\mathcal{X}_3$ is the set of common variables. We have $\mathcal{V}^\otimes = \mathcal{X}_1 \uplus \mathcal{X}_2 \uplus \mathcal{X}_3$. Similarly, action variables in the game structure $\mathcal{G}^d$ can be partitioned into three sets Λ_1, Λ_2, and Λ_3 where Λ_1 is the action variables of those controlled agents that are involved in $\mathcal{G}_1^d$ and not in $\mathcal{G}_2^d$, Λ_2 is the action variables of controlled agents that are involved in $\mathcal{G}_2^d$ and not in $\mathcal{G}_1^d$, and Λ_3 is the action variables of controlled agents that are involved in both game structures $\mathcal{G}_1^d$ and $\mathcal{G}_2^d$. Finally, let $\mathcal{Z}_i = \mathcal{X}_i \cup \Lambda_i$ for $i = 1..3$.

Note that $\mathtt{S}$ can be viewed as a predicate over $\mathcal{V}^\otimes \cup \Lambda = \mathcal{Z}_1 \cup \mathcal{Z}_2 \cup \mathcal{Z}_3$. We have

$$\begin{aligned}
\mathcal{C}_1 &= \mathbf{Project}(\mathtt{S}, \mathcal{Y}_1) \\
&= \exists \mathcal{Z}_2 . \mathtt{S} = \left\{ (z_1, z_3) \in \Sigma_{\mathcal{Z}_1} \times \Sigma_{\mathcal{Z}_3} \mid \exists z_2 \in \Sigma_{\mathcal{Z}_2} . (z_1, z_2, z_3) \models \mathtt{S} \right\}, \text{ and} \\
\mathcal{C}_2 &= \mathbf{Project}(\mathtt{S}, \mathcal{Y}_2) \\
&= \exists \mathcal{Z}_1 . \mathtt{S} = \left\{ (z_2, z_3) \in \Sigma_{\mathcal{Z}_2} \times \Sigma_{\mathcal{Z}_3} \mid \exists z_1 \in \Sigma_{\mathcal{Z}_1} . (z_1, z_2, z_3) \models \mathtt{S} \right\}.
\end{aligned}$$

We show that for any $(z_1, z_3) \in \Sigma_{\mathcal{Z}_1} \times \Sigma_{\mathcal{Z}_3}$ such that $(z_1, z_3) \models \mathcal{C}_1$, $(z_1, z_3) \models \mathbf{Project}(\mathcal{C}_1 \otimes \mathcal{C}_2, \mathcal{Y}_1)$ and vice versa. That is, $\mathcal{C}_1 = \mathbf{Project}(\mathcal{C}_1 \otimes \mathcal{C}_2, \mathcal{Y}_1)$. Let $(z_1, z_3) \models \mathcal{C}_1$. Then by definition $\exists z_2.(z_1, z_2, z_3) \models \mathbb{S}$. Besides $(z_2, z_3) \models \mathcal{C}_2$. It follows that $(z_1, z_2, z_3) \models \mathcal{C}_1 \otimes \mathcal{C}_2$. Thus, $(z_1, z_3) \models \mathbf{Project}(\mathcal{C}_1 \otimes \mathcal{C}_2, \mathcal{Y}_1) = \exists \mathcal{Z}_2.\mathcal{C}_1 \otimes \mathcal{C}_2$. For the opposite direction, let $(z_1, z_3) \models \mathbf{Project}(\mathcal{C}_1 \otimes \mathcal{C}_2, \mathcal{Y}_1)$. Then there must exist $z_2 \in \Sigma_{\mathcal{Z}_2}$ such that $(z_1, z_2, z_3) \models \mathcal{C}_1 \otimes \mathcal{C}_2$. It follows from definition of the composition that $(z_1, z_3) \models \mathcal{C}_1$. The proof for $\mathcal{C}_2 = \mathbf{Project}(\mathcal{C}_1 \otimes \mathcal{C}_2, \mathcal{Y}_2)$ is symmetric. □

From Lemma 1 and 2 it follows that there is at least a set of maximally permissive strategies $\mathcal{C}_i$ for $1 \le i \le k$ such that the projection of their composition $\bigotimes_{i=1}^{k} \mathcal{C}_i$ to each sub-games is equivalent to the computed permissive strategies for that sub-game. As a result, if there exists a winning strategy for the centralized game, a winning strategy is computed by Algorithm 2. □

References

[1] D. Halperin, J.-C. Latombe, R.H. Wilson, A general framework for assembly planning: the motion space approach, Algorithmica 26 (3–4) (2000) 577–601.
[2] S. Rodriguez, N.M. Amato, Behavior-based evacuation planning, in: IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2010, pp. 350–355.
[3] J.S. Jennings, G. Whelan, W.F. Evans, Cooperative search and rescue with a team of mobile robots, in: 8th International Conference on Advanced Robotics (ICAR), IEEE, 1997, pp. 193–200.
[4] D. Fox, W. Burgard, H. Kruppa, S. Thrun, A probabilistic approach to collaborative multi-robot localization, Auton. Robots 8 (3) (2000) 325–344.
[5] D. Rus, B. Donald, J. Jennings, Moving furniture with teams of autonomous robots, in: Proceedings of 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 1, IEEE, 1995, pp. 235–242.
[6] T. Balch, R.C. Arkin, Behavior-based formation control for multirobot teams, IEEE Trans. Robot. Autom. 14 (6) (1998) 926–939.
[7] A. Pnueli, R. Rosner, On the synthesis of a reactive module, in: Proceedings of the 16th ACM Symposium on Principles of Programming Languages, ACM, 1989, pp. 179–190.
[8] A. Pnueli, R. Rosner, Distributed reactive systems are hard to synthesize, in: Proceedings of 31st Annual Symposium on Foundations of Computer Science, IEEE, 1990, pp. 746–757.
[9] G. Peterson, J. Reif, S. Azhar, Lower bounds for multiplayer noncooperative games of incomplete information, Comput. Math. Appl. 41 (7) (2001) 957–992.
[10] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, Y. Sa'ar, Synthesis of reactive (1) designs, J. Comput. Syst. Sci. 78 (3) (2012) 911–938.
[11] H. Kress-Gazit, G.E. Fainekos, G.J. Pappas, Temporal-logic-based reactive mission and motion planning, IEEE Trans. Robot. 25 (6) (2009) 1370–1381.
[12] I. Saha, R. Ramaithitima, V. Kumar, G.J. Pappas, S.A. Seshia, Automated composition of motion primitives for multi-robot systems from safe LTL specifications, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2014, pp. 1525–1532.
[13] N. Ayanian, V. Kallem, V. Kumar, Synthesis of feedback controllers for multiple aerial robots with geometric constraints, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2011, pp. 3126–3131.
[14] T. Keviczky, F. Borrelli, G.J. Balas, Decentralized receding horizon control for large scale dynamically decoupled systems, Automatica 42 (12) (2006) 2105–2115.
[15] W.B. Dunbar, R.M. Murray, Distributed receding horizon control for multi-vehicle formation stabilization, Automatica 42 (4) (2006) 549–558.
[16] E. Şahin, S. Girgin, L. Bayindir, A.E. Turgut, Swarm robotics, in: Swarm Intelligence, Springer, 2008, pp. 87–100.
[17] Z. Shi, J. Tu, Q. Zhang, L. Liu, J. Wei, A survey of swarm robotics system, in: Advances in Swarm Intelligence, Springer, 2012, pp. 564–572.
[18] E. Filiot, N. Jin, J.-F. Raskin, Antichains and compositional algorithms for LTL synthesis, Form. Methods Syst. Des. 39 (3) (2011) 261–296.
[19] A. Church, Logic, arithmetic and automata, in: Proceedings of the International Congress of Mathematicians, 1962, pp. 23–35.
[20] S. Safra, On the complexity of ω-automata, in: 29th Annual Symposium on Foundations of Computer Science, IEEE, 1988, pp. 319–327.
[21] R. Rosner, Modular Synthesis of Reactive Systems, Ph.D. thesis, Weizmann Institute of Science, 1992.
[22] R. Alur, S. La Torre, Deterministic generators and games for Ltl fragments, ACM Trans. Comput. Log. 5 (1) (2004) 1–25.
[23] E. Asarin, O. Maler, A. Pnueli, J. Sifakis, Controller synthesis for timed automata, IFAC Proc. Vol. 31 (18) (1998) 447–452.
[24] O. Kupferman, M.Y. Vardi, Safraless decision procedures, in: 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, 2005, pp. 531–540.
[25] O. Kupferman, N. Piterman, M. Vardi, Safraless compositional synthesis, in: Computer Aided Verification (CAV), Springer, 2006, pp. 31–44.
[26] C. Baier, J. Klein, S. Klüppelholz, A compositional framework for controller synthesis, in: Concurrency Theory (CONCUR), Springer, 2011, pp. 512–527.
[27] S. Sohail, F. Somenzi, Safety first: a two-stage algorithm for LTL games, in: Formal Methods in Computer-Aided Design, IEEE, 2009, pp. 77–84.
[28] R. Alur, S. Moarref, U. Topcu, Pattern-based refinement of assume-guarantee specifications in reactive synthesis, in: Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2015, pp. 501–516.
[29] Y. Lustig, M.Y. Vardi, Synthesis from component libraries, Int. J. Softw. Tools Technol. Transf. 15 (5–6) (2013) 603–618.
[30] R. Alur, S. Moarref, U. Topcu, Compositional synthesis with parametric reactive controllers, in: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, ACM, 2016, pp. 215–224.
[31] J.H. Reif, The complexity of two-player games of incomplete information, J. Comput. Syst. Sci. 29 (2) (1984) 274–301.
[32] K. Chatterjee, T.A. Henzinger, Semiperfect-information games, in: Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Springer, 2005, pp. 1–18.
[33] M. De Wulf, L. Doyen, J.-F. Raskin, A lattice theory for solving games of imperfect information, in: Hybrid Systems: Computation and Control, Springer, 2006, pp. 153–168.
[34] K. Chatterjee, L. Doyen, T.A. Henzinger, J.-F. Raskin, Algorithms for omega-regular games with imperfect information, in: Computer Science Logic, Springer, 2006, pp. 287–302.
[35] T. Wongpiromsarn, U. Topcu, R.M. Murray, Receding horizon temporal logic planning, IEEE Trans. Autom. Control 57 (11) (2012) 2817–2830.
[36] H. Kress-Gazit, T. Wongpiromsarn, U. Topcu, Correct, reactive, high-level robot control, IEEE Robot. Autom. Mag. 18 (3) (2011) 65–74.
[37] T. Wongpiromsarn, A. Ulusoy, C. Belta, E. Frazzoli, D. Rus, 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, pp. 5011–5018.
[38] M. Kloetzer, C. Belta, Automatic deployment of distributed teams of robots from temporal logic motion specifications, IEEE Trans. Robot. 26 (1) (2010) 48–61.
[39] N. Ozay, U. Topcu, R.M. Murray, Distributed power allocation for vehicle management systems, in: 50th IEEE Conference on Decision and Control and European Control Conference, IEEE, 2011, pp. 4841–4848.
[40] P. Tabuada, Verification and Control of Hybrid Systems: a Symbolic Approach, Springer Science & Business Media, 2009.

[41] R. Parikh, R. Ramanujam, Distributed processes and the logic of knowledge, in: Logics of Programs, 1985, pp. 256–268.
[42] R. Fagin, J.Y. Halpern, Y. Moses, M. Vardi, Reasoning About Knowledge, MIT Press, 2004.
[43] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd edition, Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.
[44] S.M. LaValle, Planning Algorithms, Cambridge University Press, 2006.
[45] R. Alur, S. Moarref, U. Topcu, Compositional synthesis of reactive controllers for multi-agent systems, in: Computer Aided Verification, Springer, 2016, pp. 251–270.
[46] E.M. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 1999.
[47] Y. Gurevich, L. Harrington, Trees, automata, and games, in: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, ACM, 1982, pp. 60–65.
[48] S. Schewe, B. Finkbeiner, Bounded synthesis, in: Automated Technology for Verification and Analysis, Springer, 2007, pp. 474–488.
[49] R. Ehlers, Symbolic bounded synthesis, Form. Methods Syst. Des. 40 (2) (2012) 232–262.
[50] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems, in: STACS 95, Springer, 1995, pp. 229–242.
[51] R. Bloem, H.N. Gabow, F. Somenzi, An algorithm for strongly connected component analysis in n log n symbolic steps, Form. Methods Syst. Des. 28 (1) (2006) 37–56.
[52] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in: Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, IEEE Computer Society Press, 1993, pp. 42–47.
[53] S. Moarref, Compositional Reactive Synthesis for Multi-Agent Systems, University of Pennsylvania, 2016.