Decoupled formal synthesis for almost separable systems with temporal logic specifications

Scott C. Livingston and Pavithra Prabhakar

Abstract We consider the problem of synthesizing controllers automatically for distributed robots that are loosely coupled using a formal synthesis approach. Formal synthesis entails construction of game strategies for a discrete transition system such that the system under the strategy satisfies a specification, given for instance in linear temporal logic (LTL). The general problem of automated synthesis for distributed discrete transition systems suffers from state-space explosion because the combined state-space has size exponential in the number of subsystems. Motivated by multi-robot motion planning problems, we focus on distributed systems whose interaction is nearly decoupled, allowing the overall specification to be decomposed into specifications for individual subsystems and a specification about the joint system. We treat specifically reactive synthesis for the GR(1) fragment of LTL. Each robot is subject to a GR(1) formula, and a safety formula describes constraints on their interaction. We propose an approach wherein we synthesize strategies independently for each subsystem; then we patch the separate controllers around interaction regions such that the specification about the joint system is satisfied.

Key words: reactive synthesis, LTL, collision avoidance, motion planning

1 Introduction

Formal synthesis has gained prominence over the last few years as a promising method to design correct-by-construction controllers [3]. Formal synthesis consists of a formal model of the object that is to be controlled and a formal specification of the property this object along with the synthesized controller need to satisfy. The

S.C. Livingston (corresponding author)
California Institute of Technology, Pasadena, USA. e-mail: slivingston@cds.caltech.edu,

P. Prabhakar
IMDEA Software Institute, Madrid, Spain, e-mail: pavithra.prabhakar@imdea.org

problem has been studied for several classes of models including discrete systems, timed systems and hybrid systems and several classes of specification languages including linear-time temporal logic (LTL), computation tree logic (CTL), μ-calculus as well as timed logics such as Metric Temporal Logic; e.g., [17], [10], [9].

In this paper, we focus on models that are distributed systems. Our motivation is the problem of synthesizing controllers for multi-robot systems wherein each robot has a task to achieve. For simplicity we consider the setting with two robots, but extension to a multi-robot setting is straightforward. The two robots can operate fairly independently except that they perform their tasks on a common workspace and hence might need to jointly satisfy certain constraints such as collision avoidance. We formalize this problem as a synthesis problem involving three specifications: one for each robot and one on the joint workspace.

A naive approach to deal with controller synthesis for distributed systems wherein the system is flattened to a single system using a product construction is inefficient because the size of the flattened system grows exponentially in the number of components. Thus a variety of robot architectures and control methods have been proposed that avoid this by various assumptions about communication among the agents, types of tasks, and allocation of responsibility [16], [15], [8], [5]. Relatively little prior work considers distributed robotics where tasks are formally specified using LTL. However, we remark that distributed computer systems (no continuous dynamics) is a well-known context for formal synthesis, though there are undecidability results for the general case [18]. Chen and collaborators address the problem of synchronization and task allocation by exploiting previous results for trace-closed languages [6]. Unlike the present work, they do not consider uncertainty in the environment. Ozay and collaborators present a methodology for decomposition of a specification, thus exploiting some symmetry present in the setting of multiple-target tracking in a network of actuated cameras [14]. Like the present work, they consider GR(1) formulae that can handle nondeterministic (uncertain) environments. However, unlike the present work, their decomposition is top-down in that a global specification is initially given, from which separate component specifications are manually constructed. While here we assume a discrete abstraction is given, there is prior work using sampling-based motion planning [20].

Our systems are not completely decoupled, but interact in some minimal way. Hence, we propose the following approach to deal with the state-space explosion. We synthesize the controllers independently for each robot so as to satisfy their corresponding specifications. The simultaneous execution of the strategies however might violate the specification on the joint workspace. We identify the states which violate the joint specification and patch the individual controllers in such a way that they satisfy their original specification and also satisfy the joint specification. More precisely, we identify a neighborhood around the joint property violation point and resynthesize a joint strategy for the two robots in this neighborhood satisfying the individual specification as well as the joint specification. We then decompose this joint strategy to obtain the patching strategy for individual robots. Note that patching involves solving a synthesis problem on the joint state-space of the two robots restricted to a small neighborhood of this space.

2 Preliminaries

We are concerned with the control of robots that satisfy a specification expressed in a fragment of linear temporal logic (LTL) known as GR(1). These robots operate in a shared workspace; their interaction is important and expressed formally as part of a joint specification. While most of our treatment concerns dynamics of discrete systems, such systems are practically obtained from continuous systems by a process of abstraction [1] (or [19] for a textbook introduction). The basic idea is that a finite transition system is bisimilar—in a precise sense—to a hybrid system if transitions between cells in a partition of the state space are achievable in one system if and only if they are achievable in the other. We omit further detail here because it suffices for our purposes to know that at least some of the discrete variables were obtained from a continuous dynamical system and thus admit a notion of distance.

A specification is an LTL formula that formally describes how the system should behave. It is written in terms of finitely-valued variables, some of which may be uncontrolled, like a disturbance. In this paper we use the GR(1) fragment because it has useful structure that we can exploit [4]. Our treatment of the syntax and semantics is brief and informal; an introduction can be found, e.g., in [2].

LTL is an extension to Boolean logic for describing sequences of events. Syntactically, a Boolean logic formula can contain operators $\vee$ "or" or $\neg$ "negation", along with derivative operators $\wedge$ "and", $\implies$ "implies", and $\iff$ "if and only if". These operators can be combined with the Boolean constants True and False and finitely-valued variables. Depending on its domain, a variable may appear in a subformula with inequality, e.g., $x < 5$. A Boolean formula evaluates to True or False for a particular assignment of the variables that appear in it.

A variety of operators concerning both future and past events have been introduced in LTL [7]. In this paper, we only make use of three: $\Box$ "always", $\Diamond$ "eventually", and $\bigcirc$ "next". An LTL formula is evaluated with respect to an infinite sequence of assignments to variables. Let f be a Boolean formula. Then $\Box f$ is True if and only if f is True at every time step. $\Diamond f$ is True if and only if f is True at some future time. $\bigcirc f$ is True if and only if f is True at the next time step.

Let $\mathscr{X}$ be a set of environment (or "uncontrolled input") variables, and let $\mathscr{Y}$ be a set of system (or "controlled output") variables. The sets of states, i.e., assignments from the domains, of these variables are denoted $\Sigma_{\mathscr{X}}$ and $\Sigma_{\mathscr{Y}}$, respectively. A GR(1) formula is of the form

$$\theta_{\text{env}} \wedge \Box\rho_{\text{env}} \wedge \left(\bigwedge_{j=0}^{m-1} \Box\Diamond \psi_j^{\text{env}}\right) \implies \theta_{\text{sys}} \wedge \Box\rho_{\text{sys}} \wedge \left(\bigwedge_{i=0}^{n-1} \Box\Diamond \psi_i^{\text{sys}}\right) \qquad (1)$$

where the various subformulae are as follows. First notice the analogous form of both sides of the implication in (1), $\varphi^a \implies \varphi^g$; the left-side φ^a is commonly called the "assumption," and the right-side φ^g the "guarantee." θ_{env} and $\psi_0^{\text{env}}, \ldots, \psi_{m-1}^{\text{env}}$ are Boolean formulae in terms of $\mathscr{X} \cup \mathscr{Y}$. θ_{env} is a condition that any initial state is assumed to satisfy. $\psi_0^{\text{env}}, \ldots, \psi_{m-1}^{\text{env}}$ are liveness conditions; the environment must set the variables in $\mathscr{X}$ infinitely often so as to satisfy these. ρ_{env} is a Boolean formula

in terms of $\mathscr{X} \cup \mathscr{Y} \cup \mathscr{X}'$ that constrains from any particular state how the environment may move, i.e., set variables in $\mathscr{X}$ at the next time step (hence the primed notation $\mathscr{X}'$). The right-side is defined analogously, with the noticeable difference that ρ_{sys} is in terms of $\mathscr{X} \cup \mathscr{Y} \cup \mathscr{X}' \cup \mathscr{Y}'$, thus governing how the system may move from a particular state and given an anticipated environment move. The conditions $\psi_0^{\text{sys}}, \ldots, \psi_{n-1}^{\text{sys}}$ are also called "system goals."

There exist algorithms for the synthesis of finite-memory strategies realizing a given GR(1) formula [4]. In previous work, we proposed an annotation for strategies that facilitates online patching for coping with uncertainty [13, 12]. More precisely, suppose we are given a new specification φ' from modification of the original φ, e.g., due to online sensing necessitating updating an environmental model. Under certain conditions, the algorithm in [13] provides a way to locally change an original strategy to recover correctness with respect to φ'. The present paper builds on that method, and thus we summarize here the relevant results and notation. Unless stated otherwise, all GR(1) formulae in this paper are assumed to be realizable. Let φ be a GR(1) formula. A finite-memory strategy can be represented as an automaton $A = (V, \delta, L)$, where V is a finite set of nodes, $\delta \subset V \times \Sigma_{\mathscr{X}} \times V$ is a transition relation, and $L : V \to \Sigma_{\mathscr{X}} \times \Sigma_{\mathscr{Y}}$ labels nodes with states. $(u, e, v) \in \delta$ is also denoted $\delta(u, e) = v$. An automaton is said to be a *strategy automaton* for φ if its transition relation selects a valid next state from any state reachable in a play under φ. A is said to be *winning* if all plays resulting from its application satisfy φ.

Without loss of generality, winning strategy automata in this paper are assumed to be equipped with reach annotations, which are defined as follows. Denote the set of nonnegative integers by $\mathbb{Z}_+$. Given φ of the form (1), a state s is said to be a i-system goal if s satisfies ψ_i^{sys}. π_i is the mapping providing the i-th component of elements of a Cartesian product. E.g., if $x = (x_1, x_2) \in \mathbb{R}^2$, then $\pi_1(x) = x_1$.

Definition 1 (adapted from [12]) *A* reach annotation *on a strategy automaton $A = (V, \delta, L)$ for a GR(1) formula φ is a function* $\text{RA} : V \to \{0, \ldots, n-1\} \times \mathbb{Z}_+$ *that satisfies the following conditions. Given $p < q$, the numbers between p and q are $p+1, \ldots, q-1$, and if $q \leq p$, then the numbers between p and q are $p+1, \ldots, n-1, 0, \ldots, q-1$.*

1. *For each $v \in V$, $\pi_2 \circ \text{RA}(v) = 0$ if and only if $L(v)$ is a $\pi_1 \circ \text{RA}(v)$-system goal.*
2. *For each $v \in V$ and $u \in \text{Post}(v)$, if $\pi_2 \circ \text{RA}(v) \neq 0$, then $\pi_1 \circ \text{RA}(v) = \pi_1 \circ \text{RA}(u)$ and $\pi_2 \circ \text{RA}(v) \geq \pi_2 \circ \text{RA}(u)$.*
3. *For any path $\langle v_1, v_2, \ldots, v_K \rangle$ such that $\pi_2 \circ \text{RA}(v_1) = \cdots = \pi_2 \circ \text{RA}(v_K) > 0$, there exists an environment goal ψ_j^{env} such that for all $k \in \{1, \ldots, K\}$, $L(v_k)$ does not satisfy ψ_j^{env}.*
4. *For each $v \in V$ and $u \in \text{Post}(v)$, if $\pi_2 \circ \text{RA}(v) = 0$, then there exists a p such that for all r between $\pi_1 \circ \text{RA}(v)$ and p, $L(v)$ is a r-system goal, and $\pi_1 \circ \text{RA}(u) = p$.*

In addition to other uses, reach annotation plays a crucial role in the patching algorithm of [13]. If a new reach annotation can be constructed after modifying a strategy automaton, then we immediately have that the new automaton is winning.

3 Problem formulation

We are now ready to present the problem of formal synthesis for multiple robots addressed in this paper. Let φ_1 be a GR(1) formula (recall (1)) defined in terms of the environment variables $\mathscr{X}_1$ and the system variables $\mathscr{Y}_1$. Similarly for φ_2, $\mathscr{X}_2$, and $\mathscr{Y}_2$, where we require that the sets of system variables are disjoint, i.e, $\mathscr{Y}_1 \cap \mathscr{Y}_2 = \emptyset$. (It may be that $\mathscr{X}_1 \cap \mathscr{X}_2 \neq \emptyset$.) As for the single robot case of the previous section, a state is an assignment of values to variables. Unless indicated otherwise, a state assigns values to all of $\mathscr{X}_1 \cup \mathscr{X}_2 \cup \mathscr{Y}_1 \cup \mathscr{Y}_2$. Notice there is some redundancy in referring to states $\Sigma_{\mathscr{X}_1} \times \Sigma_{\mathscr{X}_2}$ when $\mathscr{X}_1$ and $\mathscr{X}_2$ share variables. All variables are finitely valued.

We will be concerned with a joint specification. To that end, we need a way to compose φ_1 and φ_2. This is achieved by introducing an operator $\otimes$ over the set of GR(1) formulae. Making the "assumption" and "guarantee" components of the separate robot formulae explicit, write $\varphi_1 = \left(\varphi_1^a \implies \varphi_1^g\right)$ and $\varphi_2 = \left(\varphi_2^a \implies \varphi_2^g\right)$, and then define $\varphi_1 \otimes \varphi_2$ as

$$\left(\varphi_1^a \wedge \varphi_2^a\right) \implies \left(\varphi_1^g \wedge \varphi_2^g\right).$$

This is clearly a GR(1) formula, hence the operation $\otimes$ is closed over the set of GR(1) formulae.

Though the sets of discrete variables $\mathscr{Y}_1$ and $\mathscr{Y}_2$ are disjoint—and thus may be assigned independently by each respective robot—the robots perform their tasks in a shared workspace and thus may also need to meet a specification written in terms of both. This is achieved by introducing a Boolean formula $\varphi_{1,2}$ in terms of $\mathscr{Y}_1 \cup \mathscr{Y}_2$ and requiring that it is always satisfied. Finally, the target specification is

$$\varphi_1 \otimes \varphi_2 \otimes (\square\, \varphi_{1,2}). \tag{2}$$

in which we have omitted the "assumption" portion of GR(1) formula $\square\, \varphi_{1,2}$.

This problem could be solved by synthesizing in a product space to obtain a joint strategy that simultaneously selects actions for both agents. To avoid exponential increase in problem size entailed by such an approach, we propose to exploit the availability of an indicator function on states that provides a sufficient condition for satisfaction of $\varphi_{1,2}$. Concretely, suppose that $\mathscr{Y}_1$ and $\mathscr{Y}_2$ describe agents with identical dynamics that are operating in a shared workspace. Suppose further that $\varphi_{1,2}$ describes states where the agents are dangerously close to each other. Because some discrete variables are abstractions of physical positions of the agents, we can construct from Euclidean distances a function

$$d : \Sigma_{\mathscr{Y}_1} \times \Sigma_{\mathscr{Y}_2} \to \{0, 1\} \tag{3}$$

such that

$$d(s^1, s^2) > 0 \implies (\varphi_{1,2} = \text{True}) \tag{4}$$

where s^1 and s^2 are states of the variables in $\mathscr{Y}_1$ and $\mathscr{Y}_2$, respectively. Intuitively, the safety condition $\varphi_{1,2}$ can only be violated when d is not positive. If the occurrence of d positive is uncommon, then we may be able to solve φ_1 and φ_2 independently and then correct parts of the strategies where interaction occurs, i.e., where extra effort is required to ensure $\square\,\varphi_{1,2}$.

4 Solution approach

In this section the proposed solution is outlined, and detailed algorithms are presented in Section 5. Throughout this paper it is assumed that time is in discrete steps, and all agents are synchronized. More precisely, time evolves as it would in the product game, where at each step the environment first selects a move, and then, observing this, the system (i.e., the team of robots viewed as a single entity) selects a move corresponding to a simultaneous assignment of all variables in $\mathscr{Y}_1 \cup \mathscr{Y}_2$. Thus the next state is formed, and the process is repeated. This assumption allows us to avoid addressing what is otherwise a key issue in distributed robotics: synchronization. It is a reasonable approximation in the case of slowly moving robots, e.g., planar mobile robots, where such tools as the Network and Precision Time Protocols (IEEE standard 1588) are available. Furthermore, the specification (2) only requires the robots to be aware of each other to maintain the safety condition $\varphi_{1,2}$, which can be implemented passively using a range finder if this is collision avoidance. Finally, a shared environment (i.e., $\mathscr{X}_1 \cap \mathscr{X}_2 \neq \emptyset$) provides an external, event-triggered reference for coordination.

With the preceding assumption, synthesize winning strategies $A_1 = (V_1, \delta_1, L_1)$ and $A_2 = (V_2, \delta_2, L_2)$ for φ_1 and φ_2, respectively and independently, where φ_1 and φ_2 are as introduced in Section 3. Also compute reach annotations RA_1 and RA_2 for A_1 and A_2, respectively. Intuitively, we begin by treating the component specifications as being entirely separate and realize them using existing methods. To achieve (2), we must ensure that simultaneous execution of A_1 and A_2 does not lead to violation of $\varphi_{1,2}$. Since A_1 and A_2 were synthesized separately, it is possible that $\varphi_{1,2}$ is violated during their joint (simultaneous) execution. The proposed method addresses this in two parts:

1. During each time step, compute the set of nodes reachable by each robot's strategy A_i up to a horizon h. A dangerous configuration is one that may violate $\varphi_{1,2}$ and is checked for using the function (4) applied to the continuous positions of the robots. If no dangerous configurations are found within the designated horizon, then the robots' respective moves are performed. Else, proceed to the next part.
2. In order to guarantee avoidance of dangerous configurations, the robots must be aware of each other during motion planning. Accordingly, a local reachability game is constructed in terms of the joint specification (2). Its solution provides a local strategy for both robots to move from the current configuration, from which the danger was detected, to some configuration occurring after the danger. The

notion of "after" is made precise by the reach annotations of A_1 and A_2, which are used to ensure that the respective goals in (2) are met infinitely often for the new robot strategies A_1' and A_2' that result from patching-in the local joint strategy.

5 Algorithms

The proposed method consists of two major steps: identification of dangerous configurations, and joint patching around those configurations. Our method is run online, and the main loop is listed in Algorithm 1. When dangerous configurations are found there, Algorithms 2 and 3 are invoked to patch the robots' control strategies. Comments on particular lines in those algorithms follow.

Algorithm 1 Main loop, including online detection of dangerous node pairs

1: INPUT: multi-robot task specification $\varphi_1 \otimes \varphi_2 \otimes \square\varphi_{1,2}$, strategies A_1, A_2, reach annotations, $\mathrm{RA}_1, \mathrm{RA}_2$
2: Initialize with $(v_1, v_2) \in V_1 \times V_2$ depending on initial environment state.
3: **while True do**
4: **for** k in $1, 2, \ldots, h$ **do**
5: Compute $\mathrm{Post}_1^k(v_1)$ and $\mathrm{Post}_2^k(v_2)$.
6: **if** $d(s^1, s^2) = 0$ for some $(s^1, s^2) \in L_1(\mathrm{Post}_1^k(v_1)) \times L_2(\mathrm{Post}_2^k(v_2))$ **then**
7: $D_1 \times D_2 := \left\{(u^1, u^2) \in \mathrm{Post}_1^k(v_1) \times \mathrm{Post}_2^k(v_2) \mid d(L_1(u^1), L_2(u^2)) = 0\right\}$
8: Invoke Algorithm 2 with v_1, v_2 and the dangerous node pairs of A_1 and A_2.
9: **if** Algorithm 2 returned $(A_1', \mathrm{RA}_1', A_2', \mathrm{RA}_2')$ **then**
10: Replace A_1, RA_1 and A_2, RA_2 with the returned patched versions.
11: **else**
12: **abort** //Local reachability game unsolvable
13: **end if**
14: **end if**
15: **end for**
16: Each robot observes environment move: e_1, e_2
17: $v_1 := \delta_1(v_1, e_1); v_2 := \delta_2(v_2, e_2)$ //Take moves
18: **end while**

5.1 Comments on Algorithm 1

- Line 3 : because the specification describes infinite plays by the robots, the main loop should run forever to provide for online usage.
- Line 4 : to ensure that a dangerous configuration is not stepped over during the search, horizon lengths must be in increasing order, as listed in the for-loop of Algorithm 1.
- Line 5 : $\mathrm{Post}_1^k(v_1)$ is the set of nodes in A_1 reachable in k time steps beginning at v_1, for some sequence of environment moves. $\mathrm{Post}_2^k(v_2)$ is defined similarly for

Algorithm 2 Jointly patch strategies

INPUT: joint GR(1) formula $\varphi_1 \otimes \varphi_2 \otimes \square\varphi_{1,2}$, strategies A_1, A_2, reach annotations $\mathrm{RA}_1, \mathrm{RA}_2$, current node-pair (v_1, v_2), and respective danger nodes D_1, D_2
OUTPUT: new component strategy automata A_1', A_2', and new reach annotations $\mathrm{RA}_1', \mathrm{RA}_2'$
$i := \pi_1 \circ \mathrm{RA}_1(v_1)$ //Relevant goal mode for first robot
$j := \pi_1 \circ \mathrm{RA}_2(v_2)$ //Relevant goal mode for second robot
$m_i := \min_{d \in D_1} \pi_2 \circ \mathrm{RA}_1(d)$ //Min. reach annotation among dangerous nodes for first robot
$m_j := \min_{d \in D_2} \pi_2 \circ \mathrm{RA}_2(d)$
$B_1 := \{v \in V_1 \mid \pi_1 \circ \mathrm{RA}_1(v) = i \wedge \pi_2 \circ \mathrm{RA}_1(v) < m_i\}$
$B_2 := \{v \in V_2 \mid \pi_1 \circ \mathrm{RA}_2(v) = j \wedge \pi_2 \circ \mathrm{RA}_2(v) < m_j\}$
Entry $:= \{(v_1, v_2)\}$
Exit $:= B_1 \times B_2$
$A_{(i,j)} := \mathrm{Reach}_\varphi(L(\mathrm{Entry}), L(\mathrm{Exit}))$
if $A_{(i,j)} =$ `nil` **then**
    abort //Local reachability game unsolvable
end if
$(A^i, A^j) := \mathrm{Decompose}(A_{(i,j)})$
Patch A_1 with A^i and A_2 with A^j as in [13].
return $A_1', \mathrm{RA}_1', A_2', \mathrm{RA}_2'$

Algorithm 3 Decompose local strategy

INPUT: $A_{(i,j)} = \left(V_{(i,j)}, \delta_{(i,j)}, L_{(i,j)}\right)$ with state labels in $\Sigma_{\mathscr{X}_1} \times \Sigma_{\mathscr{Y}_1} \times \Sigma_{\mathscr{X}_2} \times \Sigma_{\mathscr{Y}_2}$
OUTPUT: A^i with state labels in $\Sigma_{\mathscr{X}_1} \times \Sigma_{\mathscr{Y}_1}$, and A^j with state labels in $\Sigma_{\mathscr{X}_2} \times \Sigma_{\mathscr{Y}_2}$
$V^i := \emptyset$; $V^j := \emptyset$
for $(v_1, v_2) \in V_{(i,j)}$ **do**
    $V^i := V^i \cup \{v_1\}$; $V^j := V^j \cup \{v_2\}$
    for $(u_1, u_2) \in \mathrm{Pre}((v_1, v_2))$ **do**
        $\delta^i(u_1, e_1) := v_1$, where $e_1 = L_1(v_1) \downarrow \Sigma_{\mathscr{X}_1}$
        $\delta^j(u_2, e_2) := v_2$, where $e_2 = L_2(v_2) \downarrow \Sigma_{\mathscr{X}_2}$
        if $L_1(u_1) = L_1(v_1)$ **then**
            $L^i(v_1) := L_1(v_1) \oplus \mathrm{Hash}(\mathrm{v}_1)$
        else
            $L^i(v_1) := L_1(v_1)$
        end if
        if $L_2(u_2) = L_2(v_2)$ **then**
            $L^j(v_2) := L_2(v_2) \oplus \mathrm{Hash}(\mathrm{v}_2)$
        else
            $L^j(v_2) := L_2(v_2)$
        end if
    end for
end for
return $A^i = \left(V^i, \delta^i, L^i\right), A^j = \left(V^j, \delta^j, L^j\right)$

the second robot. For clarity, Algorithm 1 does not include the obvious improvement of incrementally computing $\mathrm{Post}_2^k(v_2)$ using the value from the previous iteration of the for-loop.

- Line 6 : the predicate is in terms of continuous robot states; recall (4).
- Line 7 : compute all "dangerous" nodes that satisfy the predicate of line 6. Those of D_1 belong to the first robot; those of D_2 to the second.

- Line 16 : recall that there is only one environment, though each robot may see different parts of it. Their respective perspectives are indicated by using e_1 and e_2. On the next line, these are used to move to the appropriate next strategy automaton nodes.

5.2 Comments on Algorithm 2

- Lines 5–6 : for each robot strategy, compute the minimum reach annotation value for all nodes that are part of a dangerous configuration.
- Lines 7–8 : for each robot strategy, find all existing automaton nodes with the current goal mode and that have reach annotation strictly less than all dangerous nodes.
- Line 11 : L is the product labeling constructed from L_1 and L_2. Using the transition rules and safety conditions of the joint specification $\varphi := \varphi_1 \otimes \varphi_2 \otimes \square\varphi_{1,2}$, solve a reachability game that drives any play initially from a state in $L(\text{Entry})$ to some state in $L(\text{Exit})$, or else block one of the environment liveness conditions (recall (1)) if this is not possible. For brevity we omit a code segment for constructing a strategy automaton realizing a solution. Many algorithms to do this are known, for instance a μ-calculus fixed point is used in [13]. A reachability game is defined as an LTL formula, called $\text{Reach}_\varphi(L(\text{Entry}), L(\text{Exit}))$,

$$\chi_{L(\text{Entry})} \wedge \square\rho_{\text{env}} \wedge \left(\bigwedge_{j=0}^{m-1} \square\diamondsuit \psi_j^{\text{env}} \right) \implies \square\rho_{\text{sys}} \wedge \diamondsuit \chi_{L(\text{Exit})}$$

  for which a strategy must be synthesized. The solution may be found by restricting attention to a set of states within a distance of the dangerous configuration, as described in [13]. While we omit details here, the basic idea is to form a smaller synthesis problem by modifying ρ_{env} and ρ_{sys} given the subset of states over which patching is performed.
- Line 13 : local reachability games can be unsolvable even when the multi-robot specification (2) has solutions. In the present context, there are two possible causes. First, if one of the Entry will inevitably lead to violation of the inter-robot safety requirement $\varphi_{1,2}$. Second, we only treat the case of patching within a particular goal mode. Accordingly, it is not necessary to consider intermediate satisfaction robots' liveness goals as part of solving local reachability games. However, this may result in a trivially unsolvable reachability game, e.g., if B_1 is empty. Consult discussion in Section 6 concerning extension to the general case.
- Line 15 : invoke subroutine Decompose(), as provided in Algorithm 3, for decomposing local strategies into corresponding local strategies for separate agents, on respective discrete states $\Sigma_{\mathscr{X}_1} \times \Sigma_{\mathscr{Y}_1}$ and $\Sigma_{\mathscr{X}_2} \times \Sigma_{\mathscr{Y}_2}$.
- Line 16 : having obtained local strategies A^i and A^j on the previous line, which may be used to patch A_1 and A_2 for goal modes i and j, respectively, the single-agent algorithm of [13] can now be applied directly. We omit it for brevity.

5.3 Comments on Algorithm 3

- Lines 7–8: the notation $e_1 = L_1(v_1) \downarrow \Sigma_{\mathscr{X}_1}$ indicates projection onto the set of environment states from the perspective of agent 1, i.e., $\Sigma_{\mathscr{X}_1}$. Thus, the transition $\delta^i(u_1, e_1) := v_1$ means that upon reaching node u_1 in any play, if the environment takes the move e_1, then the strategy automaton will select a system state in $\Sigma_{\mathscr{Y}_1}$ such that the resulting discrete state in $\Sigma_{\mathscr{X}_1} \times \Sigma_{\mathscr{Y}_1}$ is $L(v_1)$.
- Lines 9–18 : avoid stuttering in component local strategies. We assume a unique number generator Hash() is available and append it to the state labelings on lines 10 and 15.

6 Analysis

As shown below, the proposed method is sound in the sense that infinite executions by a team of robots using it will realize the specification.

Theorem 2 *Any infinite play resulting from the combined operation of Algorithms 1–3 is correct with respect to specification $\varphi_1 \otimes \varphi_2 \otimes \Box \varphi_{1,2}$, provided the initial state does not violate $\varphi_{1,2}$.*

Proof. Let σ be an infinite play, i.e., a mapping

$$\sigma : \mathbb{N} \to \Sigma_{\mathscr{X}_1} \times \Sigma_{\mathscr{Y}_1} \times \Sigma_{\mathscr{X}_2} \times \Sigma_{\mathscr{Y}_2}$$

that assigns to each discrete time step a product state. Denote the projection onto the variables used by the first and second robots respectively by

$$\sigma_1 : \mathbb{N} \to \Sigma_{\mathscr{X}_1} \times \Sigma_{\mathscr{Y}_1}, \tag{5}$$
$$\sigma_2 : \mathbb{N} \to \Sigma_{\mathscr{X}_2} \times \Sigma_{\mathscr{Y}_2}, \tag{6}$$

so that $\sigma(t) = (\sigma_1(t), \sigma_2(t))$ for each time t. The proof proceeds by induction. By hypothesis, the first state $\sigma(1)$ must not violate $\varphi_{1,2}$. Since A_1 and A_2 were synthesized for φ_1 and φ_2, respectively, the initial states of both robots $\sigma_1(1)$ and $\sigma_2(1)$ satisfy the initial conditions of φ_1 and φ_2, respectively. Therefore the joint initial state $\sigma(1)$ satisfies the initial conditions of the specification $\varphi_1 \otimes \varphi_2 \otimes \Box \varphi_{1,2}$.

Suppose that for some time t, the finite play fragment $\sigma(1) \cdots \sigma(t)$ satisfies the safety requirements of the specification. In terms of the components of the specification, satisfaction of the safety requirements means that for each $\tau \leq t$, $\sigma(\tau)$ satisfies $\varphi_{1,2}$, and for each transition $(\sigma(\tau), \sigma(\tau+1))$, the transition requirements of φ_1 are met by $(\sigma_1(\tau), \sigma_1(\tau+1))$ and the transition requirements of φ_2 are met by $(\sigma_2(\tau), \sigma_2(\tau+1))$ for $\tau \leq t-1$. The robots' strategy automata A_1 and A_2 transition on line 17 of Algorithm 1, which from the labelings determines the next state, i.e.,

$$\sigma_1(t) = L_1(v_1) \tag{7}$$
$$\sigma_2(t) = L_2(v_2) \tag{8}$$
$$\sigma_1(t+1) = L_1(\delta_1(v_1, e_1)) \tag{9}$$
$$\sigma_2(t+1) = L_2(\delta_2(v_2, e_2)). \tag{10}$$

Observe that L_1 in (7) may differ from L_1 in (9), since A_1 may have changed as a result of patching (Algorithm 2). A similar statement applies to L_2 in (8) and (10). Thus we consider two cases. For the first case, if patching does not occur, i.e., if the condition of the if-statement on line 6 of Algorithm 1 is always false, then the robot strategy automata are not changed in the present iteration. In particular, the labelings L_1 and L_2 are not changed and it suffices to check that the transitions resulting from δ_1 and δ_2 are safe. If these transitions are as in the originally given A_1 and A_2, then by hypothesis they are safe with respect to φ_1 and φ_2, respectively. They are also safe with respect to $\varphi_{1,2}$ since the condition of the if-statement on line 6 was false in the present case. Summarizing the second case, the solution of the local reachability game in Alogrithm 2 implies that transitions in the modified A_1' and A_2' are safe with respect to the multi-robot specification. Therefore, by induction the safety requirements of the specification $\varphi_1 \otimes \varphi_2 \otimes \Box\varphi_{1,2}$ are always met.

It remains to show that the liveness requirements of $\varphi_1 \otimes \varphi_2 \otimes \Box\varphi_{1,2}$ are satisfied. By the definition of GR(1) formulae (1), this occurs if one of the environment liveness conditions is eventually never satisfied, or if all robots' goals are repeatedly achieved. Note that $\Box\varphi_{1,2}$ is not relevant here, and therefore, it suffices to check the first robot goals as in φ_1 and the second robot goals as in φ_2. By hypothesis, we have an infinite play and thus the abort-statement in Algorithm 2 must never occur, i.e., for every time that patching is attempted, a solution is found for the local reachability game. That means a product strategy automaton $A_{(i,j)}$ ("product" in the sense of concerning both robots' system variables $\mathscr{Y}_1$ and $\mathscr{Y}_2$) is found in which either an environment liveness condition is blocked or strict progress is made toward the respective robots' goals (modes i and j as appearing in φ_1 and φ_2, respectively). Because Algorithm 3 decomposes this joint solution of the local reachability game into strategy automata A^i and A^j for each robot so that simultaneous execution of A^i and A^j is identical to $A_{(i,j)}$, it follows that the results A_1' and A_2' of patching robot strategy automata ensure the robot goals are repeatedly reached, or else a liveness condition in φ_1 or φ_2 is blocked.

An important part of the hypothesis for the previous theorem is that it concerns infinite plays. However, during usage of the presented algorithms, a patching attempt may fail and result in a finite play. To simplify the presentation, the algorithms in this paper rely on patching to be completed within the same goal mode. While an extension to the method of [13] has been developed that can visit robot goals while connecting Entry and Exit sets, it has not yet been published and length constraints prevent inclusion of it here. Thus our present treatment is restrictive and may fail to find a joint strategy when one exists.

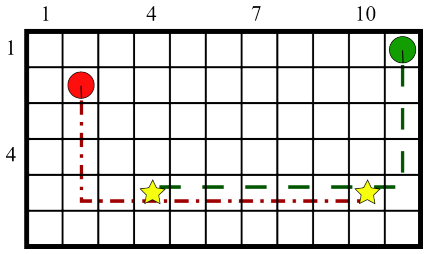

Fig. 1 Illustration of experiment setting: two-robot gridworlds. Cells that are to be visited repeatedly are indicated by stars. The robots are shown in their initial positions.

7 Experiments

In this section preliminary results concerning the complexity of the method presented in this paper are described. The experiment setting considered is illustrated in Figure 1. The underlying dynamics can be driven among cells using gradient methods [11], so to simplify the presentation we model the robots as entirely discrete transition systems.

Informally, the multi-robot specification in Figure 1 requires that both robots visit both stars repeatedly while avoiding collisions with each other. It can be shown that, in the absence of an adversarial environment, winning strategies are of the form of "lassos," with a prefix and suffix loop; example paths are illustrated by dashed lines in the figure. In this example, a collision state occurs at row 5, column 7, and it would first be found h-steps away, where h is the maximum horizon parameter used in Algorithm 1.

7.1 Quantifying coupling

The usefulness of the proposed method depends largely on how "loose" is the coupling of the robots imposed by $\varphi_{1,2}$ (recall (2)), given that nominal strategy automata constructed for φ_1 and φ_2 assume independence. We studied this in the gridworld setting as follows. Generate a random gridworld with fixed static obstacle density 0.2 (i.e., 20% of cells are occupied), and create two robot specifications φ_1 and φ_2 in it by randomly placing, for each robot, one initial position and multiple goal positions. Synthesize strategy automata A_1 and A_2 independently for φ_1 and φ_2, respectively. Compute the synchronous product of A_1 and A_2 (viewed as finite transition systems), and find all nodes that are labeled with the same grid position, i.e., all nodes at which the robots would be in collision. These are referred to as "dangerous configurations", and histograms of occurrences for varying numbers of goals and world sizes is shown in Figure 2. It is clear that in all cases, most counts are zero, i.e., the independently synthesized robot strategies A_1 and A_2 never result in dangerous configurations. While this observation holds as well for both gridworld sizes

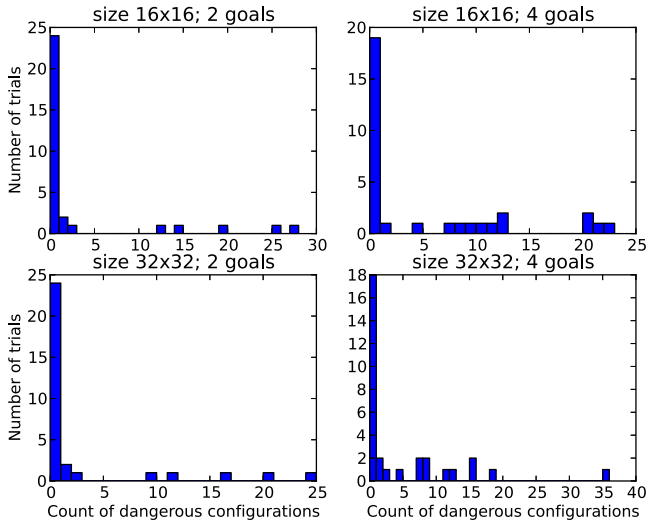

Fig. 2 Histograms for the number of trials in which a given count of dangerous configurations occurred, after generating multi-robot specifications for random gridworld instances as described in Section 7.1. In each case 32 trials were performed.

Table 1 Automaton synthesis times for multi-robot random gridworlds

Setting	Mean time (s) for specification $\varphi_1 \otimes \varphi_2$	Mean time (s) for distributed synthesis
size 16×16 , 2 goals	0.938	0.196
size 16×16, 4 goals	1.75	0.297

considered, the plots indicate that increasing the number of goal positions increases the occurrences of dangerous configurations. Intuitively we may expect this because the goal positions are randomly placed in the gridworld and thus may cause motion plans to cover more of the workspace, thereby increasing possibilities for collisions.

7.2 Simulation trials

Using the experiment setting described in the previous section, the times required for independently synthesizing strategy automata are compared with those required for using a combined specification. In terms of the previous section, strategy automata are synthesized for φ_1, φ_2, and $\varphi_1 \otimes \varphi_2$. Synthesis times are shown in Table 7.2.

References

1. R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, July 2000.
2. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
3. C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion: Finding the missing pieces of current methods and ideas. *IEEE Robotics & Automation Magazine*, pages 61–70, March 2007.
4. R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78:911–938, May 2012.
5. F. Bullo, J. Cortés, and S. Martínez. *Distributed Control of Robotic Networks*. Applied Mathematics Series. Princeton University Press, 2009. Electronically available at http://coordinationbook.info.
6. Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, February 2012.
7. E. A. Emerson. *Handbook of theoretical computer science (vol. B): formal models and semantics*, chapter Temporal and modal logic, pages 995–1072. MIT Press, 1990.
8. B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, September 2004.
9. S. Karaman and E. Frazzoli. Vehicle routing problem with metric temporal logic specifications. In *Proc. of the 47th IEEE Conference on Decision and Control (CDC)*, pages 3953–3958, Cancun, Mexico, December 2008.
10. S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *Proc. of the American Control Conference (ACC)*, pages 735–742, Montréal, Canada, June 2012.
11. S. R. Lindemann and S. M. LaValle. Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions. *The International Journal of Robotics Research*, 28(5):600–621, May 2009.
12. S. C. Livingston and R. M. Murray. Hot-swapping robot task goals in reactive formal synthesis. Technical report, California Institute of Technology, September 2014. http://resolver.caltech.edu/CaltechCDSTR:2014.001.
13. S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray. Patching task-level robot controllers based on a local μ-calculus formula. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4573–4580, Karlsruhe, Germany, May 2013.
14. N. Ozay, U. Topcu, T. Wongpiromsarn, and R. M. Murray. Distributed synthesis of control protocols for smart camera networks. In *International Conference on Cyber-physical Systems*, 2011.
15. L. E. Parker. Current state of the art in distributed autonomous mobile robotics. In *Proceedings of Distributed Autonomous Robotic Systems 4*, pages 3–12. Springer Japan, 2000.
16. L. E. Parker. Distributed intelligence: Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, 2(1):5–14, March 2008.
17. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM.
18. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, October 1990.
19. P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.
20. M. Zhu, M. Otte, P. Chaudhari, and E. Frazzoli. Game theoretic controller synthesis for multi-robot motion planning, Part I : Trajectory based algorithms. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1646–1651, Hong Kong, China, 2014.