2017 Conference on Decision and Control
http://www.cds.caltech.edu/~murray/papers/dm17-cdc.pdf

Decomposing GR(1) Games with Singleton Liveness Guarantees for Efficient Synthesis

Sumanth Dathathri Richard M. Murray

***Abstract*— Temporal logic based synthesis approaches are often used to find trajectories that are correct-by-construction for systems with complex behavior. However, the scalability of such approaches is of concern and at times a bottleneck when transitioning from theory to practice. In this paper, we identify a class of problems in the GR(1) fragment of linear-time temporal logic (LTL) where the synthesis problem allows for a decomposition that enables easy parallelization. This decomposition also reduces the alternation depth, resulting in more efficient synthesis. A multi-agent robot gridworld example with coordination tasks is presented to demonstrate the application of the developed ideas and also to perform empirical analysis for benchmarking the decomposition-based synthesis approach.**

I. Introduction

Robot motion planning has traditionally focused on generating trajectories from a given initial state to a final goal position. Recently, there has been increased attention towards generating trajectories with correct behavior in terms of synchronization of processes, safety and scheduling. The required behavior is often expressed in terms of a logic specification. The case of reactive robot motion planning from logic specifications involves considering complex behaviors for an adversarial environment and reasoning about all admissible behaviors for the environment to generate a plan for the robot. Temporal logic is often employed in this setting for reactive motion planning when we wish to generate motion plans that may exhibit complex behavior but are provably correct.

In particular, we focus on LTL [18], [20], a modal temporal logic often used as the mathematical language to formally specify the desired behavior for the robot [4], [11]. Synthesizing finite-memory strategies from LTL specifications for the general case is doubly exponential in the length of the formula [21], but for *Generalized Reactivity (1)* (GR (1))–a rich, expressive fragment of LTL, the synthesis can be done in polynomial time in the number of states and the number of liveness guarantees for the system and the number of liveness assumptions for the adversary [14]. GR(1) specifications model a game where the system and its adversary infinitely often satisfy a set of liveness constraints while making moves that satisfy certain safety constraints. This fragment in particular has received considerable attention since its conception because of the computational tractability associated with it. The GR(1) fragment is also particularly attractive because of the symbolic nature of the synthesis algorithm, that enables scaling to large finite-transition systems.

The complexity of synthesis for this class of temporal logic scales as cubic or quadratic [5] depending on the algorithm used for computing the fixed point. In our work, we identify a special subclass of GR(1) where we can provably decompose the synthesis problem into several smaller independent synthesis problems. During the decomposition procedure, we also eliminate one of the nested fixed points to reduce the *alternation depth*, thereby improving performance. In the context of motion planning, this class corresponds to finding paths that visit a set of nodes in a graph infinitely often, where each of the nodes in the graph satisfies a separate liveness condition. When an adversary is present, the relevant reactive motion planning problem is that of performing coordinated tasks for multi-agent systems [7]. Informally, the motion planning problem is to find a path for the robot such that when the environment is in pose $\mathcal{X}$, the robot has to be in pose $\mathcal{Y}$ where $(\mathcal{X}, \mathcal{Y})$ completes the coordination task. Figure 1 depicts such a problem instance where the controlled robot has to find a plan for its motion such that when the uncontrolled agent is at pose E, the robot has to attain pose C. When the environment is at pose D (pose B), the robot must be at pose A (pose F). For all assumed behaviors of the uncontrolled agent, the robot must (if feasible) find a reactive trajectory such that the agent-robot system visits the poses (E, C), (D, A) and (B, F) infinitely often. For example, in [13], the authors propose

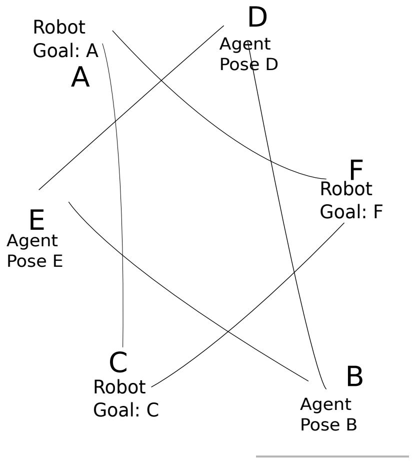

Fig. 1: Robot motion planning

an intermittent communication framework for mobile robot networks where the intermittent communication requirement is captured through an LTL formula that enforces the robots to meet infinitely often at certain rendezvous points.

The issue of scalability for synthesis algorithms has received attention in the past. In [2], the specification is compositionally decomposed to allow for scalable synthesis, with refinements being made to the decomposed specifications based on counter-strategies and strategies for the system and the environment. In [17], the GR(1) synthesis problem is decomposed into checking a feasibility problem and an online short-horizon strategy generation problem to improve scalability. In [22], the authors improve the scalability of controller synthesis by restricting themselves to linear systems and safety constraints, thereby enabling a reduction of the problem to the computation of control invariant sets. The main contribution of our work is to identify a subclass of GR(1) synthesis problems where the problem can be decomposed into smaller *reachability games*. The decomposed subgames are independent, allowing for the parallelization of the synthesis process.

II. Preliminaries

In this section, we introduce game structures and μ-calculus ([15]) over game structures as in [5] before presenting our contribution.

A. Game Structures and μ-calculus

A *game structure* $G = \langle \mathrm{AP}, \mathrm{AP}_{\mathrm{env}}, \mathrm{AP}_{\mathrm{sys}}\, \theta^{\mathrm{env}}, \theta^{\mathrm{sys}}, \rho^{\mathrm{sys}}, \rho^{\mathrm{env}}, \varphi\rangle$ represents a two player game where

- $\mathrm{AP} := \{v_1, \dots, v_n\}$: A finite set of atomic propositions,
- $\mathrm{AP}_{\mathrm{env}} \subseteq \mathrm{AP}$: Set of propositions controlled by the environment,
- $\mathrm{AP}_{\mathrm{sys}} \subseteq \mathrm{AP}$: Set of atomic propositions controlled by the system. Also, $\mathrm{AP}_{\mathrm{env}} \cap \mathrm{AP}_{\mathrm{sys}} = \emptyset$.
- $\theta^{\mathrm{sys}}, \theta^{\mathrm{env}}$: Boolean formulas in $\mathrm{AP}_{\mathrm{sys}}$ and $\mathrm{AP}_{\mathrm{env}}$ characterizing the system and environment initial states,
- $\Sigma := 2^{\mathrm{AP}}$,
- ρ^{env} is a formula in the propositions $\mathrm{AP}, \mathrm{AP}_{\mathrm{env}}$. An input $x \subseteq \mathrm{AP}_{\mathrm{env}}$ is a valid input at a state $s \in \Sigma$ if $(s, x) \models \rho^{\mathrm{env}}$,
- ρ^{sys} is a formula in the propositions $\mathrm{AP}, \mathrm{AP}_{\mathrm{env}}, \mathrm{AP}_{\mathrm{sys}}$ that specifies the system transition rules. A system output/action $y \subseteq \mathrm{AP}_{\mathrm{sys}}$ is valid if $(s, x, y) \models \rho^{\mathrm{sys}}$.

We will use the following Boolean operators: $\wedge$(conjunction), $\vee$(disjunction), $\rightarrow$(implication) and $\leftrightarrow$(bi-implication) to construct Boolean formulas. The temporal operators we use are *next* ($\bigcirc$), *eventually* ($\Diamond$) and *always* ($\Box$).

In general, the semantics of LTL are defined over infinite words in Σ^ω[20]. However, to simplify notation, we extend the semantics of LTL to reason over finite strings. For a finite string γ in Σ^*, we define:

$$\gamma \models \rho \Leftrightarrow \gamma\alpha \models \rho \text{ for any } \alpha \in \Sigma^\omega. \tag{1}$$

This allows for reasoning about states that might deadlock and playing games whose winning conditions allow for finite strings. For example, consider the formula $\Diamond\rho$. In accordance with the semantics of LTL, the language of this formula consists of strings in Σ^ω for which $\sigma_k \models \rho$ for some finite k. But, if we wanted to express the behavior for a motion planning problem where we are interested in only reaching a goal and not the behavior beyond, the extended semantics allow for that. According to the extended semantics of LTL for finite strings, the language of the above formula consists of strings in $\Sigma^\omega \cup \Sigma^*$ such that for σ in the language of the above formula, there exists a finite k such that $\sigma_k \models \rho$.

For a Boolean formula ξ, denote by $[[\xi]] \subseteq \Sigma$ the set of states that satisfy ξ. Additionally, given a game-structure G and a Boolean formula ψ, we define the relevant μ-calculus operator $\bigcirc\!\!\!\!\Diamond$ as:

$$[[\bigcirc\!\!\!\!\Diamond\psi]] = \{s \in \Sigma | \forall x \in \mathcal{P}(\mathrm{AP}_{\mathrm{env}}), (s, x) \models \rho^{\mathrm{env}} \rightarrow \\ \exists y \in \mathcal{P}(\mathrm{AP}_{\mathrm{sys}}).(s, x, y) \models \rho^{\mathrm{sys}} \wedge (x, y) \models \psi\}.$$

where $\mathcal{P}(\mathrm{AP}_{\mathrm{env}})$ denotes the power set of $\mathrm{AP}_{\mathrm{env}}$. $[[\bigcirc\!\!\!\!\Diamond\psi]]$ characterises the states from which the system can force the next state to satisfy ψ for any valid input. In this paper, we use the subscript notation, e.g., $\sigma_0\sigma_1\sigma_2 \cdots \sigma_n \in \Sigma^*$, noting that infinite strings can also be regarded as functions of the natural numbers $\mathbb{N}$ into Σ. For a finite string γ, by $\gamma_{:,r}$ we refer to the the string $\gamma_0\gamma_1 \dots \gamma_{r-1}$. By γ_{-1} we refer to the last element of γ. In other words, $\gamma_{-1} = \gamma_{|\gamma|-1}$.

B. Definitions

Let M be a finite set of memory values with $m^i \in M$ set to mark the initial memory value. A partial function $f : M \times \Sigma \times \mathcal{P}(\mathrm{AP}_{\mathrm{env}}) \rightarrow M \times \mathcal{P}(\mathrm{AP}_{\mathrm{sys}})$ is a finite-memory *strategy* for the game G if for all (w, s, x) for which $f(w, s, x)$ is defined, the condition $(s, x) \models \rho^{\mathrm{env}} \rightarrow (s, x, y) \models \rho^{\mathrm{sys}}$ holds.

A *play* $\sigma \in \Sigma^\omega \cup \Sigma^*$ for a strategy f is the maximimal sequence of states such that $\exists m \in M^\omega$ with $m_0 = m^i$ such that $(m_{k+1}, \sigma_{k+1} \cap \mathrm{AP}_{\mathrm{sys}}) = f(m_k, \sigma_k, \sigma_{k+1} \cap \mathrm{AP}_{\mathrm{env}})$ and $\sigma_k\sigma_{k+1} \models \rho^{\mathrm{env}}$ for $|\sigma| \geq k - 1 \geq 0$ if $\sigma \in \Sigma^*$ and $k \geq 0$ if $\sigma \in \Sigma^\omega$. By a sequence being maximal we imply that the play terminates when we reach a state where the strategy is not defined for a valid input or the environment deadlocks i.e. there is no valid input. We denote by $\mathrm{Plays}(f)$ the set of all plays generated by f. Also, define the set $\mathrm{Pref}(f) \in \Sigma^*$ as:

$$\mathrm{Pref}(f) := \{\sigma \in \Sigma^* | \exists \sigma\gamma \in \mathrm{Plays}(f).\gamma \in \Sigma^\omega \cup \Sigma^*\}.$$

Denote by $m^{\sigma,f}$ the sequence of memory values generated by f corresponding to $\sigma \in \mathrm{Plays}(f) \cup \mathrm{Pref}(f)$. That is, we start with the initial memory value and update the memory values as indicated by f. More formally,

$$(m^{\sigma,f}_{k+1}, \sigma_{k+1} \cap \mathrm{AP}_{\mathrm{sys}}) = f(m^{\sigma,f}_k, \sigma_k, \sigma_{k+1} \cap \mathrm{AP}_{\mathrm{env}})$$

Given a strategy $f : M \times \Sigma \times \mathcal{P}(\mathrm{AP}_{\mathrm{env}}) \rightarrow M \times \mathcal{P}(\mathrm{AP}_{\mathrm{sys}})$, we define the set of reachable state memory pairs from an initial condition $\theta = \theta^{\mathrm{sys}} \wedge \theta^{\mathrm{env}}$ and an initial memory value m^i:

$$\{(w, s) | \exists j : (w, s) = (m^{\sigma,f}_j, \sigma_j) : \ \sigma \in \mathrm{Plays}(f), \\ \sigma_0 \models \theta, m^{\sigma,f}_0 = m^i\}.$$

For a state s in Σ, a strategy f is *winning* if

$$\forall \sigma \in \text{ Plays}(f).(\sigma_0 = s \rightarrow \sigma \models \varphi), \quad (2)$$

$$f(m_{-1}^{\sigma,f}, \sigma_{-1}, x) \text{ is defined } \forall \sigma \in \text{Pref}(f) \text{ with } \sigma_0 = s, \quad \forall x \in \text{AP}_{\text{env}} \text{ with } \sigma_{-1}x \models \rho^{\text{env}}. \quad (3)$$

with φ being the winning condition. Note that from the above definition, starting from a winning state, the system has a well-defined output action for any given valid input at every state-memory pair visited by following f – as long as the environment has not violated the safety assumption in the past. Dually, we define a winning state based on the existence of a winning strategy. A state $s \in \Sigma$ is a winning state against a condition φ if there exists a strategy f that is winning from that state. The winning set is the largest set of winning states from which there exists a strategy f that is winning. We use W_φ to denote the set of winning states associated with a formula φ.

C. Generalized Reactivity(1)

A game structure G with a winning condition of the form

$$\varphi := \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \bigwedge_{j=1}^{n} \Box\Diamond\psi_j^{\text{sys}} \quad (4)$$

is an instance of a *Generalized Reactivity(1)* game where ψ_i^{env} and ψ_j^{sys} are Boolean formulae in AP.

Problem 1: For a given game structure G, the *GR(1) synthesis* problem is to find a finite memory strategy f that is winning for the set of states satisfying the initial condition θ, against the condition φ (as in equation (4)).

The worst case complexity for GR(1) synthesis in general scales as $\mathcal{O}\left((nm|\Sigma|)^3\right)$ [14]. Using the approach in [6], a GR(1) game can be solved with $\mathcal{O}(nm|\Sigma|^2)$ *next step* computations but this memoization scheme stores many binary decision diagrams (BDDs) simultaneously which makes reordering the BDDs expensive. We refer the reader to [3] for an introduction to BDDs, and to [1], [19] for a discussion of variable reordering algorithms.

For the case when all the liveness formulae ψ_i^{sys} are such that the sets $[[\psi_i^{\text{sys}}]]$ are singletons i.e there exists exactly one $s \in \Sigma$ for each ψ_i^{sys} such that $s \models \psi_i^{\text{sys}}$, we propose an approach to decompose the original GR(1) games into $n+1$ independent smaller subgames. Solving each smaller subgame involves solving a μ-calculus formula with a smaller alternation depth of 2. The *alternation depth* of a formula is the number of alternations in the nesting of least and greatest fixpoints.

III. Decomposition for Singleton Liveness Guarantees

A. Reachability Games

A *reachability* game is an instance of a game structure G with a winning condition of the form

$$\varphi_{\text{rg}} := \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \Diamond\psi^{\text{sys}}. \quad (5)$$

$L(\varphi_{rg}) = \{\sigma : \sigma \in \Sigma^* \cup \Sigma^\omega, \exists$ finite k such that $\sigma_k \models \psi^{\text{sys}}$ OR $\sigma \in \Sigma^\omega$ such that $\sigma \models \bigvee_{i=1}^{m} \Diamond\Box\neg\psi_i^{\text{env}}\}$.

Remark 1: For a game structure G, the winning states for a reachability game can be computed by solving a μ-calculus formula with an alternation depth of 2.

This holds as a direct consequence of Lemma 9 from [14]. Consider the μ-calculus formula μ_{rg} defined as:

$$\mu_{\text{rg}} := \mu Y \left(\bigvee_{j=1}^{m} \nu X \left(\left((\psi^{\text{sys}} \vee \bigcirc\!\!\!\!\!\Diamond Y) \vee \neg\psi_j^{\text{env}} \right) \wedge \bigcirc\!\!\!\!\!\Diamond X \right) \right). \quad (6)$$

Here, ν is the greatest fixpoint operator and μ is the least fixpoint operator (See [23] for detailed definitions of these operators). The alternation depth for this formula is 2. Intuitively, the fixed point in X characterizes the set of states from which the system can force the play to stay indefinitely in $[[\neg\psi_j^{\text{env}}]]$ for some j or in a finite number of steps reach a state satisfying $\psi^{\text{sys}} \vee \bigcirc\!\!\!\!\!\Diamond Y$. Staying in $[[\neg\psi_j^{\text{env}}]]$ for some j indefinitely implies blocking the environment from satisfying one of its liveness assumptions. The outer least fixed point in Y makes sure that the phase of play represented by $\bigcirc\!\!\!\!\!\Diamond Y$ eventually ends in $[[\psi^{\text{sys}}]]$. This way either $\Diamond\psi^{\text{sys}}$ is satisfied or $\bigvee_{i=1}^{m} \Diamond\Box\neg\psi_i^{\text{env}}$ is satisfied. These fixed points can be computed with complexity $\mathcal{O}\left((m|\Sigma|)^2\right)$[10]. The approach in [6] results in $\mathcal{O}(m|\Sigma|^2)$ *next step* computations to solve for the fixed points.

B. Generalized Reactivity (1) Games

In this section, we identify a special class of $GR(1)$ synthesis problems where the winning strategy can be computed by solving $n+1$ reachability games instead of solving the cyclic μ-calculus formula with an alternation depth of 3.

For the case with two liveness guarantees, the μ-calculus formula in [14] can be written using the *vector notation* as:

$$\mu_\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{j=1}^{m} \nu X \left(\left(\left(\psi_1^{\text{sys}} \wedge \bigcirc\!\!\!\!\!\Diamond Z_2 \right) \vee \bigcirc\!\!\!\!\!\Diamond Y \vee \neg\psi_j^{\text{env}} \right) \wedge \bigcirc\!\!\!\!\!\Diamond X \right) \right) \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X \left(\left(\left(\psi_2^{\text{sys}} \wedge \bigcirc\!\!\!\!\!\Diamond Z_1 \right) \vee \bigcirc\!\!\!\!\!\Diamond Y \vee \neg\psi_j^{\text{env}} \right) \wedge \bigcirc\!\!\!\!\!\Diamond X \right) \right) \end{bmatrix} \quad (7)$$

The μ-calculus formula in equation (7) has an alternating depth of 3. We propose and prove an algorithm to solve for the winning states and extract a strategy by solving independent reachability-games, each involving solving a μ-calculus formula with an alternation depth of 2.

A GR(1) game with n liveness guarantees is decomposed into $n+1$ reachability games when $|[[\psi_i^{\text{sys}}]]| = 1$ for each i. The reachability games are independent, unlike the cyclic dependency in equation (7) between the various liveness guarantees. The outermost fixed point computation in Z_i can be avoided here as the liveness guarantees correspond to singleton sets and this allows for the separation of the subgames (we prove this later). Here, for example, the GR(1) game can be split into three reachability games:

$$\bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \Diamond\psi_1^{\text{sys}}, \tag{8}$$
$$\bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \Diamond\psi_2^{\text{sys}},$$
$$\bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \Diamond\,\texttt{False},$$

with the initial conditions $\theta_1 = \psi_2^{\text{sys}}$, $\theta_2 = \psi_1^{\text{sys}} \vee \theta$ and $\theta_0 = \theta$ from the reachability games respectively (recall θ is the initial condition for the original synthesis problem). In general, for a problem with n liveness constraints, the reachability games can be set-up as for $j \in \{1, 2, \ldots, n\}$:

$$\bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \Diamond\psi_{j\oplus 1}^{\text{sys}} \tag{9}$$

with the initial conditions being $\theta_j = \psi_j^{\text{sys}}$ for $j \neq n$ and $\theta_j = \psi_j^{\text{sys}} \vee \theta$ for $j = n$. Note that $\oplus$ is the `modulo` n operator i.e $j \oplus 1 = (j+1)$ modulo n. For example, $n \oplus 3 = 3$ when $3 < n$.

We shall refer to the winning condition

$$\bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \Diamond\psi_k^{\text{sys}}$$

as φ_k^{reach}. Additionally, define φ_0^{reach} as

$$\bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \Diamond\,\texttt{False} \tag{10}$$

and the initial condition for this game is θ. Note that for any given state, a strategy that is winning against this condition can only do so by forcing the play to block the environment from satisfying its assumptions.

IV. Strategy for GR(1) from reachability games

Here, we formalize the approach for combining the strategies for the reachability games. Suppose φ_0^{reach} is winnable, then from solving φ_0^{reach} we have a strategy for $\bar{\varphi}$ and this is also winning for φ, since the environment is blocked from satisfying its assumptions.

Suppose φ_0^{reach} is not winnable and the other n reachability games are winnable. We construct the strategy f_G^{φ} by combining the n reachability games that is winning against φ. To do this, we introduce a variable $\mathcal{Z}_n$ that can take values in $\{1, 2, \ldots, n\}$ to track which liveness guarantees have been satisfied in the current cycle, with $\mathcal{Z}_n$ initialized to n. Let $f^{\text{reach}_j} : M^j \times \Sigma \times \mathcal{P}(\text{AP}_{\text{env}}) \rightarrow M^j \times \mathcal{P}(\text{AP}_{\text{sys}})$ be the winning strategy for φ_j^{reach}, with m_0^j as the initial memory. The strategy f_G^{φ} is constructed such that starting with a state $s \models \theta$, the execution follows f^{reach_n} to reach a state satisfying ψ_1^{sys} or blocks the environment from satisfying one of the liveness assumptions. If the execution reaches ψ_1^{sys}, the strategy switches to f^{reach_1} and reaches ψ_2^{sys} or blocks the environment, and so on.

Formally, the strategy

$$f_G^{\varphi} : (M \times \{1, 2, \ldots, n\}) \times \Sigma \times \mathcal{P}(\text{AP}_{\text{env}}) \rightarrow (M \times \{1, 2, \ldots, n\}) \times \mathcal{P}(\text{AP}_{\text{sys}})$$

is constructed as

$$f_G^{\varphi}((w, \mathcal{Z}_n), s, s' \cap \text{AP}_{\text{env}}) = ((w', \mathcal{Z}_n'), s' \cap \text{AP}_{\text{sys}}),$$

where if $s \models \psi_{\mathcal{Z}_n \oplus 1}^{\text{sys}}$,

$$\mathcal{Z}_n' = \mathcal{Z}_n \oplus 1,$$
$$(w', s' \cap \text{AP}_{\text{sys}}) = f^{\text{reach}_{\mathcal{Z}_n'}}(m_0^{\mathcal{Z}_n'}, s, s' \cap \text{AP}_{\text{env}},),$$

and if $s \not\models \psi_{\mathcal{Z}_n \oplus 1}^{\text{sys}}$,

$$(w', s' \cap \text{AP}_{\text{sys}}) = f^{\text{reach}_{\mathcal{Z}_n}}(w, s, s' \cap \text{AP}_{\text{env}}),$$
$$\mathcal{Z}_n' = \mathcal{Z}_n.$$

Here Z_n' denotes the value of Z_n at the next step. Similarly, s' is the the next state with s being the current state. When $s \models \psi_{\mathcal{Z}_n \oplus 1}^{\text{sys}}$ for a given $\mathcal{Z}_n$, we increment $\mathcal{Z}_n$. Thereby switching to the strategy $f^{\text{reach}_{\mathcal{Z}_n \oplus 1}}$ till we reach $\psi_{\mathcal{Z}_n \oplus 2}^{\text{sys}}$.

If for the initial condition θ, φ_0^{reach} is not winnable and for some i such that $n \geq i > 0$, φ_i^{reach} is not winnable then φ is not winnable from θ.

V. Discussion

In general GR(1) games do not allow for such a decomposition, but the singleton nature of the sets corresponding to the liveness goals allows us to decompose the specifications here. To see why, we take a closer look at the μ-calculus formula correspoding to a GR(1) game with two liveness guarantees for the system from equation (7).

To simplify things, suppose that the environment cannot be blocked from satisfying its liveness assumptions (i.e. φ_0^{reach} is not reazliable). The fixed points in Z_1 and Z_2 are such that after starting at a state in Z_1, the system forces the play into Z_2 while visiting a state satisfying ψ_1^{sys}. Similarly, from Z_2 the system forces the play into a state in Z_1 while visiting ψ_2^{sys}. This could take a few iterations for the fixed points in Z_1 and Z_2 to converge because a change in Z_1 may cause a change in Z_2, and this goes on till the greatest fixed point is met. Figure 2 illustrates this process, where initially Z_2 is the set of all states and Z_1 is the set of states from which the system can force the play into Z_2 from visiting `Goal 1`. Subsequently, in the next computation, Z_2 is the set of states from where the play can be forced into Z_1 after visitng `Goal 2`. The iterations are continued till convergence [1].

But, if the sets corresponding to the liveness guarantees are singletons, we can chain a sequence of strategies (synthesized independently) at the liveness guarantees to enforce the cyclic satisfaction of the liveness guarantees as opposed to going through the iterations in Z_i.

[1]In the case when the liveness guarantees correspond to singleton sets, it can be shown that no more than two iterations are needed to compute the outermost fixpoint.

[2]Z_2 after iteration i is a subset of Z_1 from iteration i, Z_1 from iteration $i+1$ is a subset of Z_1 from iteration i. The figure is not an accurate representation and is drawn so for better visualization.

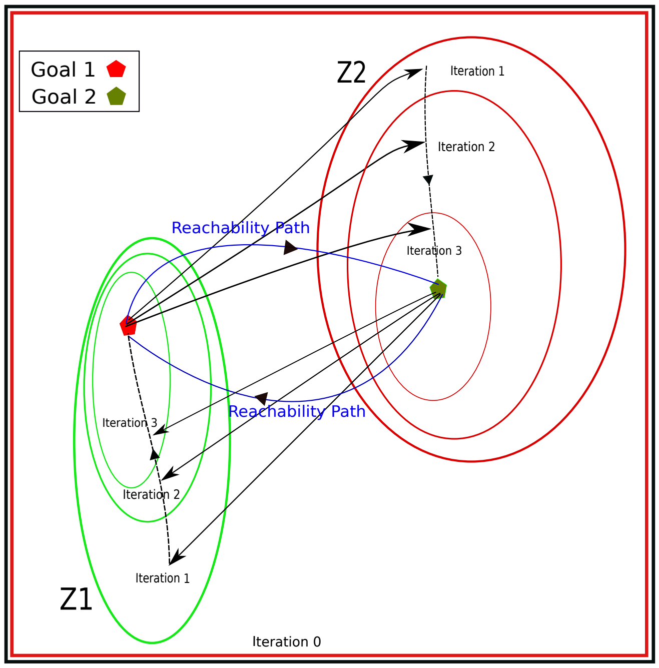

Fig. 2: Visualization of the different approaches [2]

VI. Results

Define $\bar{\varphi}$ as the following formula:

$$\bar{\varphi} := \bigwedge_{i=1}^{m} \psi_i^{\text{env}} \rightarrow \Diamond \psi_1^{\text{sys}} \wedge \left(\bigwedge_{i=1}^{n} \Diamond \left(\psi_i^{\text{sys}} \rightarrow \Diamond \psi_{i \oplus 1}^{\text{sys}} \right) \right). \tag{11}$$

Claim 2: $W_\varphi = W_{\bar{\varphi}}$ if $|[[\psi_i^{\text{sys}}]]| = 1 \, \forall i \in \{1, 2 \ldots n\}$.
The winning sets for the formulas $\bar{\varphi}$ and φ are the same. This implies that the set of states from which we can satisfy each ψ_i^{sys} once is the same as the set of states from which we can cycle through the ψ_i^{sys}-s infinitely often. A proof of the Claim is provided in [8].

Lemma 3: A game structure G with a GR(1) winning condition of the form $\varphi = \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{env}} \rightarrow \bigwedge_{i=1}^{n} \Box\Diamond\psi_i^{\text{sys}}$ can be solved by solving $n+1$ independent reachability games if $|[[\psi_i^{\text{sys}}]]| = 1 \; \forall i \in \{1, 2, \ldots, n\}$.
A proof of the Lemma is provided in [8].

Note that the construction of the strategy in Section IV combines constructions from the proofs of Claim 2 and Lemma 3.

VII. Experiments

For comparing the performance of the synthesis algorithm proposed here with standard GR(1) synthesis, we consider the problem of coordinated planar reactive robot motion planning on a gridworld. For a given set of cells $\{(a_1^r, a_1^c), (a_2^r, a_2^c), \ldots, (a_n^r, a_n^c)\}$ and $\{(b_1^r, b_1^c), (b_2^r, b_2^c), \ldots, (b_n^r, b_n^c)\}$, the controlled robot has to coordinate with a moving agent such that the robot is in cell (b_i^r, b_i^c) when the agent is in cell (a_i^r, a_i^c). The robot has to complete this coordination task infinitely often. To make sure the problem is feasible, the cells $\{\{(b_1^r, b_1^c), \ldots, (b_n^r, b_n^c)\}\}$ are added as liveness conditions for the agent. The robot's motion constraints allow movement to any of its non-diagonally adjacent cells.

Let Y_r denote the row (horizontal) position of the controlled robot Y_c the column (vertical) position. Similarly, let X_r and X_c denote the row and the column position of the uncontrolled agent. The transition rule for the robot at position $(Y_r, Y_c) = (i, j)$ can be written as:

$$\begin{aligned}(Y_r = i \wedge Y_c = j) \rightarrow \Big(&(Y_r' = i+1 \wedge Y_c = j) \\ &\vee (Y_r' = i \wedge Y_c' = j+1) \\ &\vee (Y_r' = i \wedge Y_c' = j) \\ &\vee (Y_r' = i-1 \wedge Y_c' = j) \\ &\vee (Y_r' = i \wedge Y_c' = j-1) \Big)\end{aligned}$$

with the additional constraint that Y_r and Y_c always stay in the bounds of the gridworld i.e

$$0 \leq Y_r \leq r_{\max}, 0 \leq Y_c \leq c_{\max}.$$

The agent's motion is constrained in a similar way. Furthermore, the controlled robot as a part of the safety specification has to avoid collision with the uncontrolled agent i.e

$$\neg(X_r = Y_r \wedge Y_c = X_c)$$

where $\neg$ is the negation operator. Both the controlled robot and the uncontrolled agent have to avoid collisions with the walls (shaded). For example, if the location (w_r, w_c) is shaded, then the safety specification corresponding to avoiding collision with this wall for the controlled robot is:

$$\neg(Y_r = w_r \wedge Y_c = w_c).$$

The liveness assumptions can be specified as:

$$\bigwedge_{i=1}^{n} \Box\Diamond(X_r = b_i^r \wedge X_c = b_i^c).$$

The liveness guarantees are written as:

$$\bigwedge_{i=1}^{n} \Box\Diamond(Y_r = a_i^r \wedge Y_c = a_i^c \wedge X_r = b_i^r \wedge X_c = b_i^c).$$

Figure 3 shows an example gridworld instance. The runtimes for the approach presented here using the decomposed reachability games is compared with those for the solvers `gr1c`[16] and `slugs`[9]. The solvers are accessed using the interfaces in the Temporal Logic Planning Toolbox (`TuLiP`)[12]. The reachability games are solved using the `rg` module from `gr1c`. The computations were performed on a 2.40GHz Quadcore machine with 16 GB of RAM [3]

Gridworld instances with varying number of liveness guarantees and gridsizes are used for benchmarking. Figure 4 depicts the mean runtimes from the benchmarking experiments on $t \times t-$ sized gridworld instances, with varying t. For each

[3]A link for the implemented examples is available in [8].

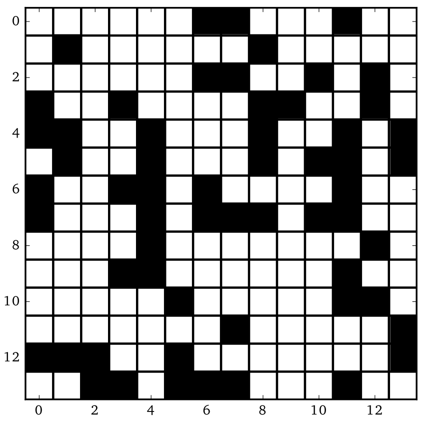

Fig. 3: Gridworld of size 14×14 with wall density of 0.3

grid size, 50 random problem instances (with a wall density of 10 percent and 6 liveness guarantees) are created. We see that the decomposition based approach outperforms GR(1) synthesis (using `slugs` and `gr1c`).

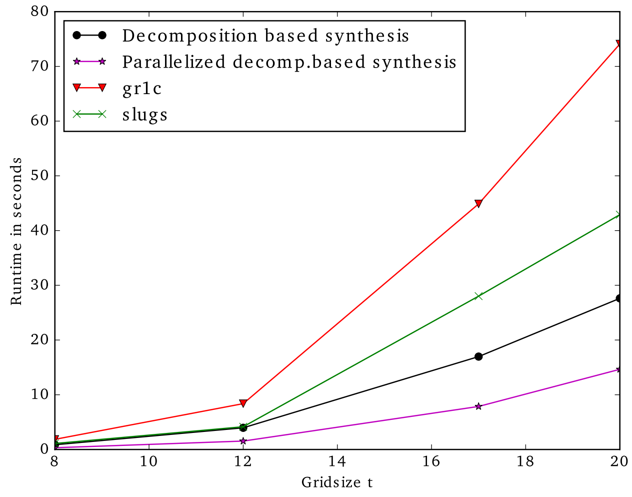

Fig. 4: Performance on gridworld problems with varying grid size ($t \times t$)

Figure 5 depicts the performance for gridworld instances of size 14×14 (wall density 0.3) with the number of liveness constraints changing. Here again we observe similar trends with the decomposition approach outperforming GR(1) synthesis using `slugs` and `gr1c`. When the reachability games are solved in parallel, we observe improved scaling for the decomposition based approach. The slope for the parallelized decompositioned-based synthesis is lesser than that of decomposition-based synthesis without parallelization. This is because the complexity of each of the reachability games is independent of n, where n is the number of liveness-guarantees. Since the $n+1$ reachability games are solved in parallel, the runtime approximately stays constant even with the varying number of liveness guarantees.

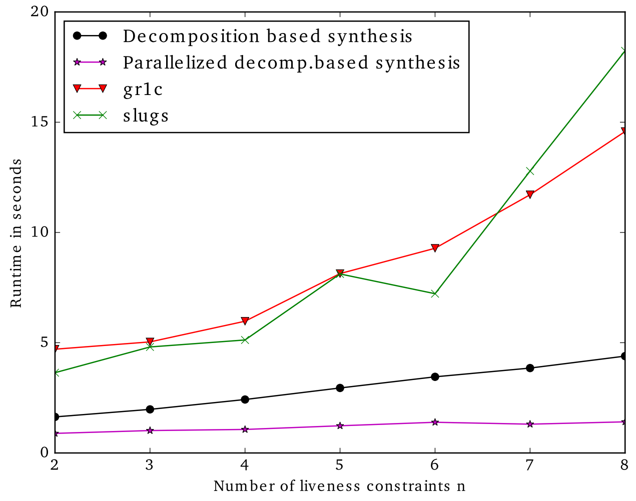

Fig. 5: Performance on gridworld problems with varying number of liveness constraints

VIII. Conclusion

We identified a class of synthesis problems where GR(1) games can be solved by solving a set of reachability games that arise from decomposing the original GR(1) formula. The decomposed reachability games are independent from each other allowing for parallelization. Synthesis from the decomposed set of games also results in better performance from eliminating the outer greatest fixed point in GR(1) synthesis. For benchmarking, the decomposition based approach is used to solve LTL motion planning problems that involve coordination between an external agent and a controlled robot on a gridworld. The experiments demonstrate the improved performance of the decomposition-based approach.

Directions for future work include extending the results here to the case when one of the liveness guarantees corresponds to a set of states that is a singleton. Even though in that case the problem cannot be fully decomposed for parallelization–the alternation depth can still be reduced by eliminating the outer greatest fixed point corresponding to GR(1) synthesis. Using the intuiton from here, we wish to further explore approaches for the decomposition of the GR(1) synthesis problems in more general settings to allow for parallelization–heuristic based if complete decomposition is not possible. Another direction of research is to explore approaches for abstraction where

1) Existing discrete transition systems can be abstracted into those with liveness conditions that correspond to singleton sets;
2) For abstraction based synthesis for systems with continous dynamics, the abstraction into a discrete transition system is done so that the sets corresponding to the liveness guarantees are singletons.

Acknowledgments: The authors would like to thank Scott C. Livingston and Tung M. Phan for helpful input. This work was supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA.

References

[1] *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[2] R. Alur, S. Moarref, and U. Topcu. *Pattern-Based Refinement of Assume-Guarantee Specifications in Reactive Synthesis*, pages 501–516. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[3] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[4] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *2010 IEEE International Conference on Robotics and Automation*, pages 2689–2696, May 2010.

[5] 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.

[6] A. Browne, E. Clarke, S. Jha, D. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178(1):237 – 255, 1997.

[7] 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.

[8] S. Dathathri and R. M. Murray. Decomposing gr(1) games with singleton liveness guarantees for efficient synthesis. http://resolver.caltech.edu/CaltechAUTHORS:20170920-152025463, 2016.

[9] R. Ehlers and V. Raman. *Slugs: Extensible GR(1) Synthesis*, pages 333–339. Springer International Publishing, Cham, 2016.

[10] E. A. Emerson and C. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proc. Symp. on Logic in Computer Science*, Cambridge, MA, 1986.

[11] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 2020–2025, Barcelona, Spain, 2005.

[12] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray. Control design for hybrid systems with TuLiP: The temporal logic planning toolbox. In *2016 IEEE Conference on Control Applications (CCA)*, pages 1030–1041, Sept 2016.

[13] Y. Kantaros and M. M. Zavlanos. Simultaneous intermittent communication control and path optimization in networks of mobile robots. *CoRR*, abs/1609.07038, 2016.

[14] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace inclusion. *Information and Computation*, 200:35–61, 2005.

[15] D. Kozen. Results on the propositional -calculus. *Theoretical Computer Science*, 27(3):333 – 354, 1983.

[16] S. C. Livingston. gr1c: a collection of tools for GR(1) synthesis and related activities. `http://scottman.net/2012/gr1c`. [Online; accessed 15-March 2016].

[17] S. C. Livingston and R. M. Murray. Just-in-time synthesis for reactive motion planning with temporal logic. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5033–5038, Karlsruhe, Germany, May 2013.

[18] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[19] C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1998.

[20] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[21] 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.

[22] M. Rungger, M. Mazo, and P. Tabuada. Scaling up controller synthesis for linear systems and safety specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 7638–7643, Dec 2012.

[23] K. Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms*. SpringerVerlag, 2004.