Parallelizing Synthesis from Temporal Logic Specifications by Identifying Equicontrollable States

Sumanth Dathathri, Ioannis Filippidis and Richard M. Murray

Keywords: Synthesis, Temporal logic, Generalized Reactivity, Binary-Decision Diagrams, Formal Methods, Discrete Event Systems

Abstract For the synthesis of correct-by-construction control policies from temporal logic specifications the scalability of the synthesis algorithms is often a bottleneck. In this paper, we parallelize synthesis from specifications in the GR(1) fragment of linear temporal logic by introducing a hierarchical procedure that allows decoupling of the fixpoint computations. The state space is partitioned into equicontrollable sets using solutions to parametrized games that arise from decomposing the original GR(1) game into smaller reachability-persistence games. Following the partitioning, another synthesis problem is formulated for composing the strategies from the decomposed reachability games. The formulation guarantees that composing the synthesized controllers ensures satisfaction of the given GR(1) property. Experiments with robot planning problems demonstrate good performance of the approach.

1 Introduction

As robotic systems get more complex, logic specifications assist in precisely specifying desired behavior for a system and constructing controllers that provably guarantee satisfaction of the specification. In our work, we focus on reactive synthesis from temporal logic specifications. This involves reasoning about all admissible behaviors for the environment and synthesizing a policy for the controlled agent.

Sumanth Dathathri
CMS, California Institute of Technology, USA, e-mail: sdathath@caltech.edu

Ioannis Filippidis
CDS, California Institute of Technology, USA, e-mail: ifilippi@caltech.edu

Richard M. Murray
CDS, California Institute of Technology, USA, e-mail: murray@cds.caltech.edu

This makes the algorithms for synthesis difficult to scale and synthesis can be prohibitively expensive when applied to problems with large state spaces.

In this paper, we focus on reactive planning for specifications in linear temporal logic (LTL) and, in particular, the *Generalized Reactivity (1)* (GR(1)) fragment. The complexity for synthesis for general LTL specifications is doubly exponential in the length of the formula [25]. For LTL formulas in the GR(1) fragment, synthesis can be performed in time polynomial in the size of the state space, which is typically exponential in the number of variables that describe the problem. The complexity of synthesis for the GR(1) fragment scales as cubic or quadratic depending on the algorithms used for synthesis [2] . Tractability has enabled use of the GR(1) fragment for robotics applications [7, 11, 18, 22].

Scalability of synthesis aids in application of formal techniques to more challenging rotics applications [21]. Symbolic model checking based on local invariants has been parallelized with substantial speed-ups [9, 8]. Performing computations directly on the original system can accelerate policy synthesis for a fragment of LTL more restricted than GR(1) [28]. In [1], a compositional approach is taken where first a parametrized controller is synthesized, and then a controller is synthesized over the parameters to compose the parametric controllers for reachability and safety specifications. In [10], an approach to parallelize synthesis is presented for the case when the liveness guarantees correspond to singleton sets. The work here presents an approach for parallelizing synthesis in a more general setting.

In our work, we adopt a compositional approach where the objective is to allow for the synthesis procedure to be parallelized. Our approach applies to recurrence properties, in addition to safety and reachability. The approach relies on identifying equivalence classes of states that "look the same" from a control perspective, in that we can controllably steer the system from any state to any other state within the same class (thus the name *equicontrollable*). The parametric controllers for individual reachability games are synthesized in parallel to abstract the states corresponding to the liveness guarantees into sets of equicontrollable states. Following the decomposition into such sets, we abstract away the local transitions to construct a composite synthesis problem. The performance gain comes from solving the parametrized reachability games in parallel during the identification of the equicontrollable sets. Each process has its own BDD manager, thus a different kind of parallelization than a single BDD manager with parallelized operations [27].

The main contribution of this paper is an approach to decompose and parallelize synthesis for the GR(1) fragment. The approach is sound but as a limitation we do lose completeness as a result of the decomposition. The paper is organized as follows. Section 2 provides background and introduces notation and Section 3 develops an example used to illustrate the ideas presented in the paper. Section 4 describes the approach for partitioning a set into subsets of equicontrollable states. Section 5 describes a procedure for setting up the synthesis problem for the composite controller and employs the synthesized controller to compose the reachability games. This ensures that the liveness properties are satisfied (soundness). Section 6 summarizes the results from benchmarking experiments on planning problems for robots with mutually exclusive access to critical areas of the workspace.

2 Preliminaries

We use syntax from the raw temporal logic of actions (raw TLA$^+$) [19, 20], with semantics restricted to only declared variables, and variable values restricted to a finite set of values $Values$ (for example $\{\text{FALSE}, \text{TRUE}, -5, -4, \ldots, 35\}$). Temporal semantics resembles LTL [24], from where we borrow one past temporal operator. Let $Vars$ denote a finite set of variable identifiers (for example $\{\text{“}x\text{”}, \text{“}y\text{”}\}$). A state is a function with domain $Vars$ and values in $Values$, i.e., an assignment of values to variables. For example, the function $state \triangleq [var \in Vars \mapsto 3]$ assigns to every variable var the same value, 3. So if “x” $\in Vars$, then $state[var] = 3$. The set of states $\Sigma \triangleq [Vars \to Values]$ contains all assignments from $Values$. Let $X|_Y \triangleq X \cap Y$. Let Nat denote the set of natural numbers $\{0, 1, \ldots\}$.

We write $s[\![\varphi]\!]$ if formula φ is true at state $s \in \Sigma$ (also denoted by $s \models \varphi$). For example, the formula $(x < 2) \wedge (y = \text{FALSE})$ is true at the state $[x \mapsto 1, y \mapsto \text{FALSE}]$. We use the Boolean operators for conjunction $\wedge$, disjunction $\vee$, implication $\Rightarrow$, and equivalence $\equiv$, and the temporal operators eventually $\Diamond$ and always $\Box$.

Let $[\![\varphi]\!] \triangleq \{state \in \Sigma : state[\![\varphi]\!]\}$ denote the states in Σ that satisfy formula φ. A *behavior* is an infinite sequence of states $\sigma \in Behaviors \triangleq [Nat \to \Sigma]$, and a *step* is a pair of consecutive states within a behavior. Temporal formulas are interpreted over behaviors, as follows. A primed variable identifier denotes the value of a variable in the second state of a step (for example, x'), whereas an unprimed identifier denotes the variable value at the first state of a step. A formula that does not contain temporal operators (but may contain primes) is called an *action*. For an action A and σ a behavior, $\sigma, n \models A$ means $\langle \sigma[n], \sigma[n+1] \rangle [\![A]\!]$, and $\sigma, 0 \models \Box A$ means $\forall n \in Nat : \sigma, n \models A$, i.e., every step of σ satisfies A. For example, $\langle [x \mapsto 3], [x \mapsto 4] \rangle [\![x' = x + 1]\!]$. For a formula φ that contains temporal operators, $\sigma, 0 \models \Box\varphi$ means that every suffix of σ satisfies formula φ. Dually, $\sigma \models \Diamond P$ means $\neg(\sigma \models \Box\neg P)$.

Reactive synthesis can be modeled as a “game” between two players [2]. The component we design is required to behave as prescribed only as long as its environment does not violate certain assumptions. Different variables represent the component and the environment. Partition the variables into the sets

$$EnvVars, SysVars \in \text{SUBSET } Vars$$

that represent the environment and component, with $EnvVars \cap SysVars = \{\}$ and $Vars = EnvVars \cup SysVars$ (SUBSET S denotes the powerset of S, i.e., all its subsets). Let the restrictions of assignments be $\Sigma_e \triangleq [EnvVars \to Values]$ and $\Sigma_a \triangleq [SysVars \to Values]$.

Synthesis of a Mealy implementation is the problem of finding a function

$$f \in [(\Sigma_e \times \Sigma \times M) \to (\Sigma_a \times M)]$$

such that the sequences of states generated by this strategy satisfy a given specification φ. $M \subseteq Nat$ is a finite set of memory values with a unique initial memory

value m_0. For a finite-memory strategy f, the infinite sequences that occur when using f are referred to as *plays*:

$$\begin{aligned} &Plays(f) \triangleq \{\sigma \in Behaviors : \exists m \in [Nat \to M] : \\ &\quad \wedge\, m[0] = m_0 \\ &\quad \wedge\, \forall k \in Nat : \text{LET } r \triangleq k+1 \quad args \triangleq \langle \sigma[r]|_{EnvVars}, \sigma[k], m[k] \rangle \\ &\qquad\qquad\qquad \text{IN } \langle \sigma[r]|_{SysVars},\ m[r] \rangle = f[args]\} \end{aligned}$$

where $f|_S$ is the function $[x \in S \mapsto f[x]]$.

A strategy is *winning* for a formula φ if and only if all plays of f satisfy the formula, A state $s \in \Sigma$ is winning if there exists a strategy that is winning if we replace the initial condition of φ with the requirement to start at s.

2.1 Assume-guarantee specifications with GR(1) liveness

We consider temporal specifications of an assume-guarantee form, with one generalized Streett pair as liveness (i.e., an implication of recurrence properties). For Mealy implementations, assume-guarantee specifications are written using the operator [17]

$$\begin{aligned} &StrictImpl(EnvInit,\ EnvNext,\ EnvRecur, \\ &\qquad SysInit,\ SysNext,\ SysRecur) \triangleq \\ &\quad EnvInit \Rightarrow \wedge\, SysInit \\ &\qquad\qquad \wedge\, \Box(\, UpToNow(EnvNext) \Rightarrow SysNext) \\ &\qquad\qquad \wedge\, (\Box EnvNext) \Rightarrow (EnvRecur \Rightarrow SysRecur) \end{aligned}$$

The operators $EnvInit, EnvNext, EnvRecur$ describe the assumption we make about how the environment behaves, and $SysInit, SysNext, SysRecur$ the requirement from the component under design. The operator $UpToNow$ is borrowed from LTL, where it is known as *historically* [23]. This form of specification ensures that the component does not violate its safety constraint by forcing the environment to violate its assumption in the future. The operators $EnvInit, SysInit$ are state predicates (initial conditions), and $EnvNext, SysNext$ are actions. For Mealy specifications, primed $SysVars$ should not occur in $EnvNext$.

Let $Q(j), R(i)$ denote state predicates for all indices j and i. If

$$\begin{aligned} EnvRecur &\triangleq \forall j \in 1\,..\,m :\ \Box\Diamond Q(j) \\ SysRecur &\triangleq \forall i \in 1\,..\,k\ :\ \Box\Diamond R(i) \end{aligned}$$

then the liveness formula $EnvRecur \Rightarrow SysRecur$ is a GR(1) formula. The liveness formula in an assume-guarantee specification of the form $StrictImpl$ determines the complexity of synthesis [2].

2.2 Complexity of GR(1) Synthesis

Polynomial-time algorithms are known for the synthesis of strategies that implement assume-guarantee GR(1) specifications. This favorable complexity is why GR(1) synthesis is often used for high-level reasoning in engineering applications. Symbolic algorithms allow for reasoning about problems with very large state spaces, because they construct strategies by manipulating sets of states, as opposed to enumerative approaches. Symbolic algorithms for GR(1) synthesis represent sets of states as (ordered) binary decision diagrams (BDDs) [2].

A BDD is a graph-based data structure with a specific order of Boolean-valued variables (bits) associated with it. The variable ordering can have a significant effect on the size of a BDD [5, 6]. Finding a variable order that minimizes the size of a BDD is an NP-complete problem [3, 26]. The cost for reordering of BDDs is often neglected in the complexity analysis of symbolic synthesis algorithms.

A cubic-time symbolic algorithm for GR(1) synthesis is known and commonly used [2]. Memoization can reduce the run time to quadratic, but at the cost of storing and reordering (in the worst case) $nm\,|\Sigma|^2$ BDDs (roots) [4] (for an implementation see [12]). Reordering these stored BDDs can negate the gains from memoization.

Problem Statement 1 *Synthesize a strategy that is winning for a GR(1) formula φ in a scalable manner, computing sub-strategies in parallel, and compose those sub-strategies in a way that implements φ.*

Besides the obvious gain from parallelization, the decomposition-based approach presented here also allows us to compute, reorder, and store the BDDs for the sub-strategies separately, leading to gains in performance.

3 Example

In this section we introduce a simple instance of planning for a domestic mobile robot that we use to illustrate and develop ideas presented in the paper. Consider the workspace shown in Figure Fig. 1. The door is controlled by the environment. Rooms with charging stations have a lighting sign to indicate the same.

$DoorOpen$ is the proposition that indicates whether the door is open. The robot's position is controlled by us and the robot can transition from its current position through an open slit to any of the adjacent rooms. The movement between the $Corridor$ and the $LivingRoom$ is controlled by the $DoorOpen$ guarding the slit:

$$
\begin{aligned}
&\wedge \{DoorOpen, Corridor, LivingRoom, Garage, Corridor\} \subseteq \text{BOOLEAN}\\
&\wedge (DoorOpen \wedge Corridor) \Rightarrow (LivingRoom' \vee Garage' \vee Corridor')\\
&\wedge (\neg DoorOpen \wedge Corridor) \Rightarrow (Garage' \vee Corridor)\\
&\wedge (DoorOpen \wedge LivingRoom) \Rightarrow (LivingRoom' \vee Corridor')\\
&\wedge (\neg DoorOpen \wedge LivingRoom) \Rightarrow (LivingRoom')
\end{aligned}
$$

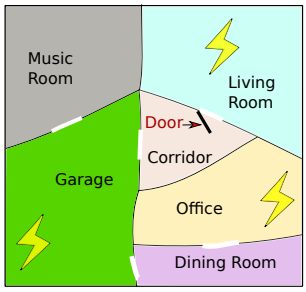


Fig. 1: Workspace for example in Section Section 3. Slits represent pathways for the robot. A door guards the path between the corridor and the living room. Lighting sign in a room indicates the presence of a charging station.

4 Partitioning into Sets of Equicontrollable States

The arguments $EnvRecur$ and $SysRecur$ of the operator $StrictImpl$ need not be GR(1) formulas. The $StrictImpl$ operator is useful for defining other kinds of games too, for example reachability games. When the formula $SysRecur$ is of the form $\diamond Goal$ where $Goal$ a state predicate, and $EnvRecur$ is TRUE, then the specification describes a reachability game. A reachability game takes $O(|\Sigma|)$ symbolic controllable steps ($\exists\forall$) to solve.

Let $G \triangleq \{s \in \Sigma \ : \ s[\![Goal]\!]\}$ be the set of states that satisfy the predicate S. Suppose that $EnvRecur$ is a conjunction of recurrence formulas ($\square\diamond$). Let $WinSet(G)$ be the set of states from where a strategy f exists that implements the specification $StrictImpl(EnvInit, EnvNext, EnvRecur, SysInit, SysNext, \diamond Goal)$. In other words, $WinSet(Goal)$ is the set of states from where the component under design can force the system to transition to $Goal$ for any admissible environment behavior. Computing the $WinSet$ takes $O(m\,|\Sigma|^2)$ symbolic steps [16].

The winning set can contain states where $EnvNext$ is false, because that case violates the assumption. In contrast to the winning set, the set of *reachable* states contains those states that the *assembled* system (component plus environment) can reach. Formally, those states that occur along behaviors that satisfy the formula $EnvInit \wedge SysInit \wedge \square(EnvNext \wedge SysNext)$ (if liveness is included, then this set can shrink). The reachable states can be computed in at most $O(|\Sigma|)$ symbolic successor steps ($\exists$). For the example, in Section 3, states where $Office = \text{TRUE} \wedge Garage = \text{TRUE}$ are unreachable, because the robot can be present in either the office or the garage, but not both.

4.1 Parametrized Games

By computing which states can be controllably reached from which other states, we can form equivalence classes of states that "look the same" from a control viewpoint. Let Prm denote a finite set of constant identifiers, distinct from variables [20, p. 25]. The value of a constant remains unchanged throughout a behavior [19, Note 16 on p. 920]. In BDD computations, bits that correspond to constants are represented

by variables that are constrained to remain unchanged. An alternative is to leave these bits unquantified. Each variable is mapped to a parameter by the function $f_{prm} \in [Vars \to Prm]$, which is a bijection. We use the parameters to define a specific state as goal, schematically

$$Goal \triangleq \bigwedge_{var \in SomeVars} (var = f_{prm}(var)) \tag{1}$$

This goal state is not fixed, but depends on the parameter values. For example, if $SomeVars \triangleq \{\text{“}x\text{”}, \text{“}y\text{”}\}$, and $Prm \triangleq \{\text{“}p_x\text{”}, \text{“}p_y\text{”}\}$, then $Goal \triangleq (x = p_x) \wedge (y = p_y)$. To map parameter assignments to states with same values for the corresponding variables, let $PrmToSt(p, V) \triangleq [var \in V \mapsto p[f_{prm}[var]]]$, and the inverse from state to parameter assignments $StToPrm(s, P) \triangleq [prm \in Prm \mapsto s[f^{-1}_{prm}[prm]]]$. We use these mappings also for assignments to subsets of variables and parameters, by passing $V \subset Vars$ and $P \subset Prm$. For simplicity, we discuss in terms of the Cartesian product of Σ with the set of parameter assignments $\Pi \triangleq [Prm \to Values]$.

Solving a game with the parametric liveness objective $\diamond Goal$ yields a parametric set of winning states, $WinSet(Goal) \subseteq \Sigma \times \Pi$. For each valuation of parameters $p \in [Prm \to Values]$, for each state $s \in \Sigma$, it is $\langle s, p \rangle \in WinSet(Goal)$ if, and only if, we can start from s and controllably reach the state $t \triangleq PrmToSt(p, Vars)$.

The parametric set $WinSet(Goal)$ can be computed in $O(|\Sigma|^2)$ symbolic steps, as follows from the corresponding μ-calculus formula [16, Lemma 9]. For each assignment of values to the parameters in Prm, the parametrized game corresponds to solving a non-parametric game with at most Σ reachable states. Thus, the symbolic set operations can be viewed as operating on copies of the same transition system, for different valuations of the parameters. So the number of symbolic steps needed for solving each non-parametric game is limited by the same upper bound. However, the symbolic steps themselves are more expensive due to the added parameters, which increase the size of BDDs. In other words, the parametric game corresponds to solving multiple non-parametric games side-by-side [1]. The parametrization we use is similar to previous work on reachability games [1], with the difference that we include liveness for the environment, as recurrence properties in $EnvRecur$.

4.2 Partitioning the states into Equicontrollable Sets

We use the parametric winning set $WinSet(Goal)$ from Section 4.1 to group states into sets that are controllably reachable from each other. Let $SomeVars \subseteq Vars$. A *partial state* is a function $s \in [SomeVars \to Values]$. Let $SomePrms \triangleq \{f_{prm}[var] : var \in SomeVars\}$, and

$$CanGoTo(a, b) \triangleq \forall s \in \Sigma : \vee (s|_{SomeVars} \neq a) \\ \vee \langle s, StToPrm(b, SomePrms) \rangle \in WinSet(Goal).$$

We call a pair of partial states $\langle s_1, s_2 \rangle$ *equicontrollable* if and only if, from any state s that agrees with s_1 on $SomeVars$, the agent can force the execution to some state that agrees with s_2 on $SomeVars$, and vice versa, i.e.,

$$AreEquicontrollable(s_1, s_2) \triangleq CanGoTo(s_1, s_2) \wedge CanGoTo(s_2, s_1).$$

Problem Statement 2 *Given a state predicate ξ, partition the set of states $\{s \in \Sigma : s[\![\xi]\!]\}$ into equivalence classes of equicontrollable states over the variables in $SomeVars \subseteq Vars$.*

In this paper, we let $SomeVars$ be the set of variables that the formula ξ depends on, also known as the *support* variables (in BDD terminology). We restrict attention to only the subset of variables $SomeVars$ for two reasons. Firstly, formulas of recurrence goals in $SysRecur$ typically depend on only a few variables. So the satisfaction of those goals is insensitive to the values of other variables. For example, if the liveness guarantee is $\square\diamond(LivingRoom \vee Garage)$ in our running example, and we declare a variable $Weather$, the value of this variable is irrelevant to this goal. The second reason is that too fine-grained distinction between states leads to unnecessary fragmentation of equivalence classes. By omitting $Weather$ from the set $SomeVars$, states with the robot in the living room belong to the same equivalence class, even though they may differ in what the wheather conditions are outside the house. The robot has no control over the weather conditions, but this fact can be ignored when considering whether the goal $LivingRoom \vee Garage$ has been reached. So viewed over $Vars$ (instead of $SomeVars$), not every pair of states need be equicontrollable. For a specific application, domain knowledge could guide the selection of the set $SomeVars$ to be different from the support variables of ξ.

The algorithm of Algo. 2 describes how the partitioning problem is solved. As liveness guarantee, we use the formula $NewGoal \triangleq \xi \wedge Goal$. This goal is the objective of the parametric game defined by $StrictImpl(\ldots, \diamond NewGoal)$. The algorithm splits the set of states into equicontrollability classes by iterating through assignments of values to parameters in the set $SomePrms \triangleq \{f_{prm}[var] : SomeVars\}$, i.e., only those parameters that correspond to variables in $SomeVars$. For simplicity, we use $Values$ for all variables and parameters, though the implementation has different ranges of values for each variable. This partitioning requires $O(k\,|\Sigma_{SomeVars}|)$ evaluations, where k is the number of classes. Furthermore, while iterating over sets of states, we can eliminate spurious classes by ignoring those sets that are disjoint from the set of reachable states.

Example 1. For the workspace in Section 3, we want to partition the set of states where the robot is in a room with a charging station into equivalence classes. ξ has the form $\xi = Office \vee LivingRoom \vee Garage$. This formula specifies whether the robot is in a room with a charging station. Denote the set of variables ξ depends on by $\mathcal{X} \triangleq \{Office, LivingRoom, Garage\}$.

Suppose that the behavior of the door is unconstrained. This yields that the garage and office are in the same equicontrollability class, whereas the living room is in a different class. If we assume that the door opens infinitely often $EnvRecur \triangleq$

$\square\diamond DoorOpen$), then the living room is in the same equicontrollability class as the office and garage.

Fig. 2: Separating into equicontrollable classes.

Input:

- Environment behavior $EnvInit, EnvNext, EnvRecur$, and system safety formulas $SysInit, SysNext$, and f_{prm}.
- State predicate ξ that represents the set of states to be partitioned.
- Set of states $Reach$ reachable by the assembled system (as BDD).
- Set of variables $SomeVars \in$ SUBSET $Vars$ that define equicontrollability.

Output:

- Partition $EquiCls$ of Σ into classes of equicontrollable states.

$Phi := StrictImpl(EnvInit,\ EnvNext, EnvRecur,$
$\qquad SysInit,\ SysNext,\ \xi \wedge Goal)$
$WinSet := ComputeWinSet(Phi);\qquad EquiCls := \{\}$
for $p_1 \in [SomePrms \to Values]$:
$\quad x_1 := PrmToSt(p_1, SomeVars);\qquad IsInClass := \text{FALSE}$
$\quad$ **for** $p_2 \in EquiCls$:
$\qquad x_2 := PrmToSt(p_2, SomeVars)$
$\qquad IsInClass := IsInClass \vee AreEquicontrollable(x_1, x_2)$
$\quad$ **if** $IsInClass \wedge \exists s \in \Sigma : (x_1 = s|_{SomeVars}) \wedge s[\![Reach]\!]$:
$\qquad EquiCls := EquiCls \cup \{p_1\}$
return $EquiCls$

5 Synthesizing a Composite Controller

This section describes an approach to build a transition system and a specification such that the winning strategy for this system can be used to compose the sub-strategies that are synthesized and stored in parallel to find a strategy winning against a GR(1) specification.

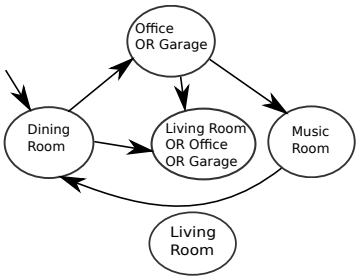
Example 2. For the workspace from Section 3, consider a synthesis problem where the robot has to patrol the dining room and the music room infinitely often, while making sure to visit a room with a charging station infinitely often and the robot is initially in the dining room. The liveness guarantees to be satisfied are:

$$\square\diamond DiningRoom, \quad \square\diamond(Office \vee Garage \vee LivingRoom), \quad \square\diamond MusicRoom.$$

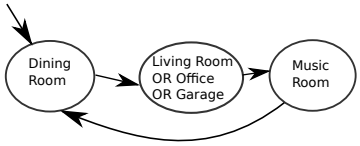
We fix the ordering of liveness guarantees ($DiningRoom, Office \vee Garage \vee LivingRoom, MusicRoom$) and build a new transition system, shown in Fig. 3. For each liveness guarantee, there is a state in the transition system that corresponds to

a subset of equicontrollable classes arising from decomposition of the set of states that satisfy the liveness guarantee. We add transitions between these states if all states in the predecessor can reach some state in the successor.

If the environment behavior is modeled as $Env \triangleq$ TRUE, we get the abstracted supervisory transition system shown in Fig. 3a. For the liveness guarantee $Office \vee Garage \vee LivingRoom$, the classes are $\{Office \vee Garage, Living\}$. The transition system in Fig. 3a has states corresponding to all non-empty subsets of these classes. If we assume that the door infinitely often opens, the supervisory transition system is that shown in Fig. 3b. Both transition systems have a cycle that can be used to compose the sub-strategies. The construction of the transition system and the composition of the sub-strategies is described below.



(a) Door need not open infinitely often.



(b) Door does open infinitely often.

Fig. 3: Supervisory transition system for different assumptions about the environment.

5.1 Synthesizing a Supervisory Controller

Consider a GR(1) formula $StrictImpl$. The states corresponding to the liveness guarantees ${\psi^{\mathrm{a}}}_i$ for the agent (within $SysRecur$) are partitioned into equicontrollable classes as described in Section 4. For each $i \in 1..n$, let k_i be the number of classes ${\psi^{\mathrm{a}}}_i$ is partitioned into. Let $\Lambda_i \triangleq \{\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,k_i}\}$ be the set of classes associated with ${\psi^{\mathrm{a}}}_i$. Let $\Omega_i \triangleq (\text{SUBSET } \Lambda_i) \setminus \{\}$. Note that $|\Omega_i| = 2^{k_i} - 1$. Without loss of generality, we fix some ordering of the elements in Ω_i such that $\Omega_i = \{\Omega_{i,1}, \Omega_{i,2}, \ldots, \Omega_{i,2^{k_j}-1}\}$. Denote by $A_{\Omega_{i,j}}$ the $WinSet$ for $\Omega_{i,j}$ i.e $A_{\Omega_{i,j}} = WinSet(\Omega_{i,j})$ For the case in Example 2 where the door is not assumed to open infinitely often, the classes that correspond to the specification where the robot has to infinitely often visit a room with a charging station are $\Lambda_2 = \{Office \vee Garage, LivingRoom\}$ and $\Omega_2 = \Big\{\{Office \vee Garage\}, \{LivingRoom\}, \{Office \vee Garage, LivingRoom\}\Big\}$.

Note that if $|\Omega_{i,j}| = 1$ we can use the parametric $WinSet$ computed for decomposing $[\![\psi^{\mathrm{a}}{}_i]\!]$ into equicontrollable classes to obtain the $WinSet$ for $\Omega_{i,j}$, by setting values to the parametric parameters appropriately. For $\Omega_{i,j}$ with $|\Omega_{i,j}| > 1$, we compute ***WinSet***$(A_{\Omega_{i,j}})$ by solving a reachability game with $\Omega_{i,j}$ as the goal to be reached. As these computations are independent, they can be performed in parallel. The symbolic steps here are less difficult than that for finding the ***WinSet*** set of $[\![\psi^{\mathrm{a}}{}_i]\!]$, because we are only considering a subset of the set of states satisfying the liveness constraint $[\![\psi^{\mathrm{a}}{}_i]\!]$ and hence there are fewer transitions to be accounted for in each symbolic step.

Following this setup, the hierarchical game is constructed as follows. The set of additional Boolean-valued variables is $\overline{\mathrm{AP}} \triangleq \{\rho_{i,j_i} : i \in 1..n, j_i \in 1..2^{k_i} - 1\}$. The transition rule is specified as:

$$\rho^{\mathrm{cmp}} \triangleq \bigvee_{v \in \overline{\mathrm{AP}}} v \quad \wedge \bigwedge_{\substack{i \in 1..n\,, \\ j \in 1..2^{k_i}-1}} \Big(\rho_{i,j} \Rightarrow \bigvee_{\substack{\Omega_{i,j} \subseteq A_{\Omega_{k,l}}, k \in \{i, i \oplus 1\}, \\ l \in 1..2^{k_i}-1}} \rho'_{k,l}\Big). \tag{2}$$

Here $\bigvee$ is the exclusive-or operator. In Eq. (2), we allow for a transitions between the states $\{\rho_{i,j}\}$ and $\{\rho_{k,l}\}$ only if the current $\Omega_{i,j}$ is contained in the $WinSet$ for the $\Omega_{k,l}$ corresponding to the successor state. For the transition system in Fig. 3a, the $DiningRoom$ is in the $WinSet$ of $LivingRoom \vee Office \vee Garage$ but $LivingRoom \vee Office \vee Garage$ is not in the ***WinSet*** of $MusicRoom$. The transition relations reflect the same. Similarly, transition relations are constructed between the other states.

Note that we restrict $k \in \{i, i \oplus 1\}$, ensuring that only those transitions are chosen that either stay in the same liveness guarantee or lead to the next liveness guarantee. Thus, we avoid cycling back to a liveness guarantee that was visited earlier in the current cycle.

The liveness guarantees ensure that infinitely often for each $i \in 1..n$, $\rho_{i,j}$ for some j is satisfied, and can be written as ${\psi^{cmp}}_i \triangleq \bigvee_{j \in 1..2^{k_i}-1} \rho_{i,j}$. This ensures that at least one of the classes corresponding to a liveness guarantee is visited in each cycle through the liveness guarantees. For a particular i, satisfying ${\psi^{\mathrm{cmp}}}_i$ is equivalent to satisfying ${\psi^{\mathrm{a}}}_i$ in the original system.

Note that here there is no environment here and we only need to search for a cycle passing through all the liveness guarantees for a given set of initial states. Consider some $i \in 1..n$. The set of valid initial states are the nodes corresponding to the elements in Ω_i for which $EnvInit \wedge SysInit$ lies in their $WinSet$. The initial condition can be written as (for some i) $\theta^{cmp} \triangleq \bigvee_{\theta \in A_{\Omega_{i,j}}} \rho_{i,j}$. Conjoining the specifications above, we need to find a controller that implements the formula

$$\theta^{cmp} \wedge \Box\rho^{cmp} \wedge \bigwedge\nolimits_{i \in 1..n} \Box\Diamond\psi_i^{cmp}. \tag{3}$$

Finding a winning strategy $f^{cmp} \in [\Sigma_{\overline{\mathrm{AP}}} \times M^{sup} \to \Sigma_{\overline{\mathrm{AP}}} \times M^{sup}]$ for the above specification gives the supervisory controller for composing the strategies for the reachability games.

5.2 Assembling the Sub-strategies

Define $f^{k,l}_{i,l} \in [\Sigma_e \times \Sigma \times M^{k,l}_{i,l} \to \Sigma_a \times M^{k,l}_{i,l}]$ to be the strategy that takes the agent from a state in $\Omega_{i,l}$ to $\Omega_{k,l}$. Let $\underline{m}^{k,l}_{i,j}$ be the initial memory value for $M^{k,l}_{i,l}$. Without loss of generality, assume $M^{k_1,l_1}_{i_1,l_1} \cap M^{k_2,l_2}_{i_2,l_2} = \{\}$ when $\langle i_1, j_1, k_1, l_1\rangle \neq \langle i_2, j_2, k_2, l_2\rangle$. Let $k_{\max} \triangleq \max(\{k_i \ : \ i \in 1..n\})$, i.e., $k_{\max}$ is the size of the largest number of equicontrollable classes for any of the liveness classes. Let $\overline{M} \triangleq M^{\mathrm{sup}} \times \bigcap\limits_{i,j,k,l} M^{k,l}_{i,j}$ and $\xi_{comp} \triangleq 1..N \times 1..k_{\max} \times 1..N \times 1..k_{\max}$. We construct a strategy $f^{compose} \in [\Sigma_e \times \Sigma \times \xi_{comp} \times \overline{M} \to \Sigma_a \times \xi_{comp} \times \overline{M}]$ that uses f^{cmp} to compose the strategies for the reachability games ($f^{k,l}_{i,l}$)

$$f^{compose}[x, s, \langle i, j, k, l\rangle, \langle w, w_{sup}\rangle] = \langle y, \langle i', j', k', l'\rangle, \langle w', w'_{sup}\rangle\rangle$$

where if $s \in \Omega_{k,l}$, then

$$\begin{array}{ll} \wedge \langle y, w'\rangle = {f^{k,l}}_{i,j}[x, s, \underline{m}^{k,l}_{i,j}], & \text{else} \wedge \langle y, w'\rangle = f^{k,l}_{i,j}[x, s, w] \\ \wedge \langle \{\rho_{k',l'}\}, w'_{sup}\rangle = f^{cmp}[\{\rho_{k,l}\}, w_{sup}], & \quad \wedge \langle i', j', k', l'\rangle = \langle i, j, k, l\rangle \\ \wedge \langle i', j'\rangle = \langle k, l\rangle. & \quad \wedge w'_{sup} = w_{sup} \end{array} \tag{4}$$

For the case $s \notin \Omega_{k,l}$, while we are moving towards $\Omega_{k,l}$, the values are updated according to the strategy $f^{k,l}_{l,j}$. Once we reach $\Omega_{k,l}$ ($s \in \Omega_{k,l}$), the next goal is updated according to f^{cmp} and we continue towards the next goal, switching goals again once the next goal is reached.

Theorem 1. *Strategy* f^{compose} *is sound. Solving Eq.* (3) *takes* $O((2^{k_{max}})^2 n^3)$ *symbolic steps.*

PROOF SKETCH: By construction, a winning strategy in the original system was computed corresponding to every transition in the abstracted system i.e the agent can either force the execution to the next liveness guarantee or block the environment from satisfying the assumption on its behavior. A winning strategy for the abstracted system finds an execution that cycles through the liveness guarantees. Cycling through the liveness guarantees in the abstracted system correspond to cycling through liveness guarantees in the original system. Hence, composing the strategies from the reachability games in accordance with the composite controller ensures satisfaction of the original GR(1) formula.
The specification resulting in Eq. (3) is a GR(1) formula without an environment i.e it is not reactive, hence the innermost fixpoint associated with blocking the environment from satisfying its assumptions does not add to the number of symbolic steps

to be performed. The total number of states is $(n2^{k_{max}})$ and there are n liveness guarantees, resulting in $\mathcal{O}((2^{k_{max}})^2 n^3)$ symbolic steps [13].
For applications where the number of liveness guarantees and the number of equicontrollable classes are much smaller than the total number of states, i.e $n \ll |\Sigma|$ and $k \ll |\Sigma|$, the parallelized approach presented here is well-suited and should result in performance gains in term of computation time. We expect such behavior in multi-agent systems with large state spaces where the agents' dynamics are not closely coupled.

Limitations

The presented algorithm is not complete, because transitions can be lost during the abstraction into a supervisory transition system. Another limitation of the approach is that if we end up with a large number of equicontrollable classes, computing the compositional strategy can become intractable.

6 Experiments

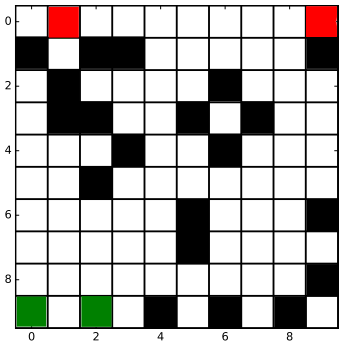


Fig. 4: Workspace with shaded obstacles (black) and critical sections (green and red).

We consider a multi-agent robot motion planning problem where the objective is to, for a set of robots, schedule access to critical sections of a given workspace in a safe manner. The environment consists of an uncontrolled adversarial mobile robot that is functioning in the same workspace as the controlled robots and requires access to certain critical sections of the workspace. An example instance is shown in Figure 4. Both the controlled robots and the uncontrolled robot have to visit cells shaded with each of the three colors (red and green) infinitely often. The problem instances are parametrized in terms of the size of the workspace, the number of critical sections and the number of controlled robots. The colored cells represent critical

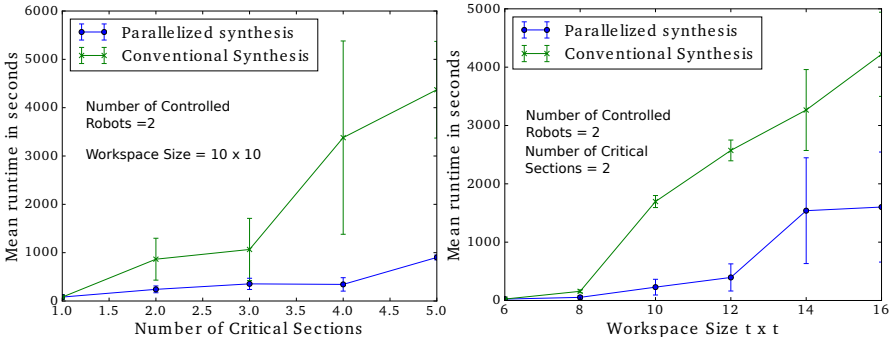
sections of the workspace that must be accessed in a mutually exclusive manner and each color represents a critical resource of a type. While the adversarial robot is accessing a critical section, the controlled robots must not access the same critical section. Similarly, no two controlled robots can access a critical section at the same time. The adversarial robot's access to the critical sections is prioritized over the controlled robots. When the adversarial robot attempts to access a critical space, the controlled robots as a part of their safety requirement must allow the adversarial robot to gain access by vacating the critical section. The regions shaded black (density=5%) represent static obstacles and both the uncontrolled and controlled robots must avoid the obstacles. The robots are allowed to transition to any of their non-diagonally adjacent cells in a single step. The uncontrolled robot is allowed to pursue a trajectory of its choice and the only assumption on its behavior, in addition to the constraints on its motion, is that the uncontrolled robot will access cells shaded with each of the colors infinitely often.

Figures 5a, 5b report performance over problem instances of varying size. The mean time over 50 problem instances is reported. For each of these instances the initial positions of the robots, the positions of the obstacles and the critical sections are randomized. The computations were performed on a 32 core AMD Opteron machine at 2.4GHz with 96 GB of RAM, and and we see considerable gains in computation time for the parallelized approach. The implementation is written in Python using the packages `dd`, `omega` [15], and `tulip` [14].[1] A possible explanation for the large variance is the cost of reordering the BDDs, which can show large variance depending on the structure of the formula for which the reordering is being done. For the case of the varying number of critical sections, the number of equivalence classes roughly increases linearly with the number of critical sections. Hence, for the parallelized approach, the dependence on the number of critical sections is smaller.

7 Conclusions and Future Work

A major challenge to the widespread adoption of formal methods is their scalability. As systems get larger and complex, scalable algorithms that can deal with the size and complexity of the systems are necessary. In this regard, we present an approach that allows us to decompose and parallelize the synthesis algorithm for the GR(1) fragment of linear temporal logic. The approach relies on the construction of a composite strategy that is used to compose local strategies to ensure satisfaction of the GR(1) specification. However, the approach comes with certain drawbacks as outlined earlier. Empirical evidence demonstrating the resulting gains in performance is presented for a robot motion planning problem, where in addition to path planning, safe access to certain critical sections of the workspace is scheduled.

[1] Available from PyPI: `https://pypi.org/project/dd/`, `https://pypi.org/project/omega/`, and `https://pypi.org/project/tulip/`.



(a) Varying Number of Critical Sections

(b) Varying size of the workspace

Fig. 5: Performance on benchmark experiments – mean runtimes over 50 randomized problem instances are reported, error bars indicate standard deviation.

Future work would be to explore if a similar approach can be used to synthesize policies that can handle uncertainty, because some local uncertainty can be tolerated without a resynthesis of the entire strategy by applying a local correction. Some unexpected disturbances can be handled locally, without the need for global synthesis. A hierarchical framework as presented here could be used in such settings. Another direction for future work includes exploring the possibility of a symbolic approach for decomposition of sets into equicontrollable classes as opposed to the enumerative approach considered here.

Acknowledgements This work was supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA.

References

1. R. Alur, S. Moarref, and U. Topcu. Compositional synthesis with parametric reactive controllers. In *HSCC*, pages 215–224, 2016.
2. 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, 2012.
3. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.
4. 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.
5. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *TC*, 35(8):677–691, 1986.
6. R. E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *TOC*, 40(2):205–213, 1991.
7. S. Chinchali, S. C. Livingston, U. Topcu, J. W. Burdick, and R. M. Murray. Towards formal synthesis of reactive controllers for dexterous robotic manipulation. In *ICRA*, pages 5183–5189, 2012.

8. A. Cohen, K. S. Namjoshi, and Y. Sa'ar. SPLIT: A compositional LTL verifier. In *CAV*, pages 558–561, 2010.
9. A. Cohen, K. S. Namjoshi, Y. Sa'ar, L. D. Zuck, and K. I. Kisyova. Parallelizing a symbolic compositional model-checking algorithm. In *HVC*, pages 46–59, 2010.
10. S. Dathathri and R. M. Murray. Decomposing GR(1) games with singleton liveness guarantees for efficient synthesis. In *CDC*, 2017.
11. J. A. DeCastro, J. Alonso-Mora, V. Raman, D. Rus, and H. Kress-Gazit. Collision-free reactive mission and motion planning for multi-robot systems. In *ISRR*, 2015.
12. R. Ehlers and V. Raman. `Slugs`: Extensible GR(1) synthesis. In *CAV*, pages 333–339, 2016.
13. E. A. Emerson and C. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *LICS*, pages 267–278. IEEE Computer Society Press, 1986.
14. 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 *CCA*, pages 1030–1041, 2016.
15. I. Filippidis, R. M. Murray, and G. J. Holzmann. A multi-paradigm language for reactive synthesis. In *4th Work. on Synthesis, SYNT*, pages 73–97, 2015.
16. Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace inclusion. *Information and Computation*, 200:35–61, 2005.
17. U. Klein and A. Pnueli. Revisiting synthesis of GR(1) specifications. In *HVC*, pages 161–181, 2010.
18. H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's Waldo? Sensor-based temporal logic motion planning. In *ICRA*, pages 3116–3121, 2007.
19. L. Lamport. The temporal logic of actions. *TOPLAS*, 16(3):872–923, 1994.
20. L. Lamport. *Specifying Systems*. Addison-Wesley, 2002.
21. R. Majumdar. Robots at the edge of the cloud. In *TACAS*, pages 3–13, 2016.
22. S. Maniatopoulos, P. Schillinger, V. Pong, D. C. Conner, and H. Kress-Gazit. Reactive high-level behavior synthesis for an Atlas humanoid robot. In *ICRA*, pages 4192–4199, 2016.
23. Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *PODC*, pages 377–408, 1990.
24. A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.
25. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.
26. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD*, pages 42–47, 1993.
27. T. van Dijk, A. Laarman, and J. van de Pol. Multi-core BDD operations for symbolic reachability. *ENTCS (PDMC'12)*, 296:127–143, 2013.
28. E. M. Wolff and R. M. Murray. Optimal control of nonlinear systems with temporal logic specifications. In *ISRR*, pages 21–37, 2016.