Compositional Synthesis with Parametric Reactive Controllers

Rajeev Alur
University of Pennsylvania
alur@seas.upenn.edu

Salar Moarref
University of Pennsylvania
moarref@seas.upenn.edu

Ufuk Topcu
University of Texas at Austin
utopcu@utexas.edu

ABSTRACT

Reactive synthesis with the ambitious goal of automatically synthesizing correct-by-construction controllers from high-level specifications, has recently attracted significant attention in system design and control. In practice, complex systems are often not constructed from scratch but from a set of existing building blocks. For example in robot motion planning, a robot usually has a number of predefined motion primitives that can be selected and composed to enforce a high-level objective. In this paper, we propose a novel framework for synthesis from a library of parametric and reactive controllers. Parameters allow us to take advantage of the symmetry in many synthesis problems. Reactivity of the controllers takes into account that the environment may be dynamic and potentially adversarial. We first show how these controllers can be automatically constructed from parametric objectives specified by the user to form a library of parametric and reactive controllers. We then give a synthesis algorithm that selects and instantiates controllers from the library in order to satisfy a given linear temporal logic objective. We implement our algorithms symbolically and illustrate the potential of our method by applying it to an autonomous vehicle case study.

1. INTRODUCTION

Reactive synthesis with the ambitious goal of automatically synthesizing correct-by-construction controllers from high-level specifications, has recently attracted significant attention in system design and control. Recent advances in this growing research area have enabled automatic synthesis of interesting real-world systems [4], indicating the potential of the synthesis algorithms for solving realistic problems.

Although automatic synthesis of realistic systems with large state spaces seems to be unattainable for now, in practice, complex systems are often not constructed from scratch (an implicit assumption in many of the related works,) but from a set of existing building blocks. For example in robot motion planning, a robot usually has a number of predefined motion primitives that can be selected and composed to enforce a high-level objective [7]. Intuitively, a compositional approach that solves smaller and more manageable subproblems, and hierarchically composes the solutions to implement more complicated behaviors seems to be a more plausible way to synthesize complex systems.

To this end, we propose a compositional and hierarchical framework for synthesis from a library of *parametric* and *reactive* controllers. Parameters allow us to take advantage of the symmetry in many synthesis problems, e.g., in motion planning for autonomous robots and vehicles. Reactivity of the controllers takes into account that the environment may be dynamic and potentially adversarial. We first show how these controllers can be synthesized from parametric objectives specified by the user to form a library of parametric and reactive controllers. We then give a synthesis algorithm that selects and instantiates controllers from the library in order to satisfy a given safety and reachability objective.

Consider an autonomous vehicle V_1 that starting from an initial location s_0 needs to navigate safely through streets and intersections to reach a final destination d, as shown in Figure 1. Safe navigation means that the vehicle must follow the traffic rules (e.g., moving in specific directions of streets), and besides avoid collision with other vehicles. In this example, V_1 can cross both intersections I_1 and I_2 on its way toward the location d. One can observe that although intersections I_1 and I_2 are located in different positions, V_1 can safely cross them in a similar way. In other words, V_1 can employ a controller to cross the intersection I_1 and employ the same controller to cross I_2. To take advantage of such *symmetry* in synthesis problems, we introduce *parametric* controllers. Let (x, y) be the location of V_1 at any time step. Assume a, b are two parameters. We would like to synthesize a controller that starting from a parametric location $(x, y) = (a, b)$, guarantees to eventually move two steps forward horizontally, i.e., eventually $(x, y) = (a+2, b)$, while avoiding collision with other vehicles. To this end, the parametric controller must also be reactive, i.e., it must react to other vehicles' movements to avoid collision. Once such parametric reactive controller is obtained, it can be instantiated by assigning values to parameters. For example, the same parametric controller can be instantiated based on the current location of the vehicle and be used to advance the vehicle in different locations. Note that in many application domains, systems may have task-specific controllers that are designed and verified a priori, e.g., an autonomous vehicle can have specialized controllers for different scenarios such as crossing intersections, making U-turns, switching lanes, etc.

HSCC'16, April 12-14, 2016, Vienna, Austria

 ISBN 978-1-4503-3955-1/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2883817.2883842

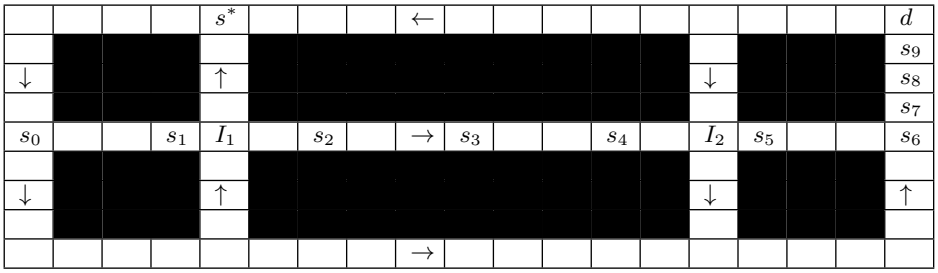

Figure 1: One-way streets connected by intersections.

Such controllers can be defined parametrically and instantiated and composed to perform more complicated tasks.

The proposed framework has two layers, parametric controller synthesis (bottom layer) and synthesis from a library of parametric controllers (top layer). In the bottom layer, a set of parametric controllers are synthesized from parametric objectives specified by the user. Here, unlike other related works [7, 12, 14], we do not assume that the controllers are a priori given, but we let the user specify them and synthesis is done automatically. This facilitates the design process and makes it more flexible, allowing the user to utilize her insight into the system being designed to construct different libraries. Furthermore, the user may not know the range of the parameter values that guarantees correct behavior of the controller. We allow the user to provide a parametric specification and the set of acceptable parameter values are discovered automatically. On the other hand, the high-level composer does not necessarily need to know *how* controllers enforce their objectives. Thus a *controller interface* that hides the controller's specific implementation while providing information on possible outcomes of the controller is synthesized for each parametric controller. A library of parametric controllers can be reused to realize more complex behaviors. In the top layer of the framework, given a library of parametric controllers and a high-level objective for the system, a *control strategy* that selects and instantiates parametric controllers from the library such that their composition enforces the objective is synthesized.

Note that adding parameters increases the size of the state space and can add to the complexity of the problem. Therefore, how parameters are handled is crucial. We provide *symbolic* algorithms that efficiently explore the parametric space. Besides, we show that the upper bound on the number of symbolic steps, i.e., pre-image or post-image computations, performed by the symbolic algorithm is independent from the parameters. Nevertheless, this does not mean that adding parameters has no cost as it increases the complexity of the symbolic steps. The main advantages of the introduced framework are *i*) *reusability of controllers* (parametric controllers are computed once and can be reused in different compositions to achieve higher level objectives,) *ii*) *separation of concerns* (design of controllers is separated from their composition which can also lead to strategies that are defined hierarchically and are easier to understand).

One of the main motivations for our work is the growing interest in controller synthesis for autonomous robots and vehicles from high-level temporal logic specifications (e.g., [8, 9, 11, 18]). A common theme is based on first computing a discrete controller satisfying the LTL specification over a discrete abstraction of the system, which is then used to synthesize continuous or hybrid controllers guaranteed to fulfill the high-level specification. In this paper, we assume that a finite-state abstraction of the system is given and we present a compositional algorithm for synthesizing a discrete controller. The computed controller can then be refined to a controller enforcing the specification over the original system using the techniques in the literature [16].

The concept of *motion primitives* is popular and widely used in robotics and control literature, since they can be designed by one group, e.g., the robot designer, and then be used by other groups of people such as the end-users to implement higher level objectives. The end-user only needs to have an understanding of what a specific motion primitive does through a provided interface, and the actual implementation is encapsulated and hidden from the end-user. A compositional motion planning framework for multi-robot systems is presented in [14] where given a library of motion primitives, the motion planning problem is reduced to solving a satisfiability modulo theories problem. A similar approach to ours is considered in [7] for solving motion-planning problems for time-invariant dynamical control systems with symmetries, such as mobile robots and autonomous vehicles, where motion plans are described as concatenation of a number of motion-primitives chosen from a finite library. The main difference of our work with [7, 14] is that our motion primitives are *reactive*, i.e., the controllers also takes the ongoing interaction between system and environment into account. To the best of our knowledge, we are the first to study the problem of synthesizing controllers from a library of parametric and reactive controllers.

The problem of LTL synthesis from a library of reusable components is considered in [12]. Sequential composition of controllers considered in this paper is similar to control-flow composition in [12] and is inspired by software systems. In the software context, when a function is called, the function gains the control over the machine and the computation proceeds according to the function until it calls another function or returns. Similarly, the controllers in our framework gain and relinquish control over computations of the system. The controllers have a designated set of final states. Intuitively, a reactive controller receives the control by entering an initial state and returns the control when reaching a final state. The goal of the composer is to decide which controller will gain control when the control is returned from the controller currently in charge. Although by enumerating the parameter values and instantiating parametric controllers to obtain a library of non-parametric controllers our problem can be reduced to the one considered in [12], such naive enumeration may lead to an exponentially larger number of controllers in the library, making the method infeasible in

practice. Our algorithms *symbolically* explore the parametric space, thus avoiding the excessive explicit enumeration. To the best of our knowledge, there is no implementation of the methods proposed in [12]. Compositional reactive synthesis from LTL specifications is also considered in some recent works [1, 2, 6, 10, 15] where a strategy is synthesized compositionally by treating parts of the given LTL specification separately and combining the solutions. The setting considered in this paper is different as we are interested in synthesizing from a library of controllers that can be reused.

Contributions. The main contributions of this paper are as follows. We give an algorithm for synthesizing a control strategy that reactively chooses and instantiates controllers from a given library of controllers to enforce a high-level safety and reachability objective for the system. We show how a designer can simply specify parametric controllers and then a controller and its interface along with admissible parameter values are synthesized automatically. We implement our algorithms symbolically using binary decision diagrams and apply them to an autonomous vehicle case study to show the potential of our approach.

2. PRELIMINARIES

In this section we present the notation and terminology used in the rest of the paper. Let $\mathbb{Z}$ be the set of integers. For $a, b \in \mathbb{Z}$, let $[a..b] = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$. Let $\mathcal{V} = \{v_1, \cdots, v_n\}$ be a set of variables where each variable $v \in \mathcal{V}$ is defined over a finite domain Σ_v. We define $\Sigma_{\mathcal{V}} = \Sigma_{v_1} \times \cdots \times \Sigma_{v_n}$ to be the collective domain of the variables. Let $\mathcal{P} = \{p_1, \cdots, p_k\}$ be a set of parameters where each parameter $p \in \mathcal{P}$ is defined over a finite domain Σ_p. Let $\Sigma_{\mathcal{P}} = \Sigma_{p_1} \times \cdots \times \Sigma_{p_k}$. A valuation s over variables $\mathcal{V}$ is a value assignment to the variables in $\mathcal{V}$, i.e., $s \in \Sigma_{\mathcal{V}}$. For a subset $\mathcal{X} \subseteq \mathcal{V}$ of variables and a valuation $s \in \Sigma_{\mathcal{V}}$, we denote by $s_{|\mathcal{X}}$ the projection of s to $\mathcal{X}$.

Without loss of generality and to simplify the specification language, in the rest of the paper, we assume that all variables and parameters are defined over bounded integer domains. Boolean variables are special case where the domain is $\{0, 1\}$. Note that since the domains of variables and parameters are finite, they can be encoded using Boolean variables. Ordered binary decision diagrams (OBDDs) can be used for obtaining concise representations of sets and relations over finite domain [5]. Let $\mathcal{X} \subseteq \mathcal{V} \cup \mathcal{P}$. A predicate ϕ over $\mathcal{X}$ is a Boolean expression generated by the grammar $\phi ::= e \sim 0 \mid \phi \wedge \phi \mid \neg\phi$, where e is generated by the grammar $e ::= k \mid k \times v \mid e + e \mid e - e$, for $k \in \mathbb{Z}$ and $v \in \mathcal{V}$, and $\sim \in \{<, \leq, =, >, \geq\}$. We will use $v \neq k$ as a shorthand for $\neg(v = k)$. Other logical operators are defined in their standard manner. Let $\mathcal{X}_1, \cdots, \mathcal{X}_n$ be disjoint sets of variables defined over finite domains $\Sigma_{\mathcal{X}_i}$ for $i = 1..n$, and let ϕ be a predicate over $\mathcal{X}_1 \cup \cdots \cup \mathcal{X}_n$. A valuation $s = (s_1, \cdots, s_n) \in \Sigma_{\mathcal{X}_1} \times \cdots \times \Sigma_{\mathcal{X}_n}$ satisfies ϕ, denoted by $s \models \phi$, if replacing the variables in ϕ by their corresponding value in s makes ϕ true. For a predicate ϕ, let $\mathcal{VP}(\phi)$ be the set of variables and parameters that appear in the predicate's formula. We say ϕ is a *parametric* predicate, if $\mathcal{VP}(\phi) \cap \mathcal{P} \neq \emptyset$, i.e., there is at least one parameter in the predicate's formula. Otherwise we say ϕ is non-parametric. For a predicate ϕ over $\mathcal{X}$, we let $[\![\phi]\!]$ be the set of valuations over $\mathcal{X}$ that make ϕ true, that is, $[\![\phi]\!] = \{s \in \Sigma_{\mathcal{X}} \mid s \models \phi\}$. Given a parametric predicate ϕ over $\mathcal{V} \cup \mathcal{P}$ and a valuation $p \in \Sigma_{\mathcal{P}}$ over parameters, restriction of ϕ by p is a non-parametric predicate $\phi_{\downarrow p}$ obtained by replacing each parameter with its corresponding value. Given a parametric set $\Pi = \Sigma_{\mathcal{V}} \times \Sigma_{\mathcal{P}}$ and a parameter value $p \in \Sigma_{\mathcal{P}}$, projection of Π by p, denoted by $\Pi_{\downarrow p}$, is the set $\{s \in \Sigma_{\mathcal{V}} \mid (s, p) \in \Pi\}$.

2.1 Linear Temporal Logic (LTL)

We use LTL to specify system objectives. LTL is a formal specification language with two types of operators: logical connectives ($\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), and $\rightarrow$ (implication)) and temporal operators (e.g., $\bigcirc$ (next), $\mathcal{U}$ (until), $\Diamond$ (eventually), and $\Box$ (always)). The set of *atomic formulas AP* consists of any predicate over the variables $\mathcal{V}$. An LTL formula over variables $\mathcal{V}$ is interpreted over infinite words $w \in (\Sigma_{\mathcal{V}})^{\omega}$. The language of an LTL formula Φ, denoted by $\mathcal{L}(\Phi)$, is the set of infinite words that satisfy Φ, i.e., $\mathcal{L}(\Phi) = \{w \in (\Sigma_{\mathcal{V}})^{\omega} \mid w \models \Phi\}$. We assume some familiarity of the reader with LTL. An LTL specification over the set of variables $\mathcal{V} \cup \mathcal{P}$ is called a *parametric* LTL specification. Parametric LTL formulas are similar to non-parametric ones, except that their formulas are interpreted over infinite words $w \in (\Sigma_{\mathcal{V}} \times \Sigma_{\mathcal{P}})^{\omega}$.

EXAMPLE 1. *Let $x, y \in [0..3]$ be two integer variables. Let $a \in [0..3]$ be an integer parameter. LTL formula $\Phi = \Box(x > y) \wedge \Diamond(x = y+1)$ requires that x must always be greater than y and that eventually x is equivalent to $y+1$. The infinite word $w = (x = 2, y = 0), (x = 2, y = 1)^{\omega}$ satisfies Φ, i.e., $w \models \Phi$. The parametric formula $\Phi_{\mathcal{P}} = \Box(x > y) \wedge \Diamond(x = y+a)$ is an example of parametric LTL specification. Satisfaction of a parametric LTL formula also depends on parameter values. For example, by setting $a = 1$ in $\Phi_{\mathcal{P}}$, we have $w \models \Phi_{\mathcal{P}_{\downarrow a=1}}$.*

2.2 Symbolic Turn-Based Game Structures

Game structures provide a formalism for modeling possible executions of a system interacting with its environment. Let $\mathcal{V}$ be a set of variables defined over a finite domain $\Sigma_{\mathcal{V}}$, and $\mathcal{V}'$ be a primed copy of the variables in $\mathcal{V}$, used to represent the next values of variables after a transition. Assume there exists a special variable $t \in \mathcal{V}$ with domain $\Sigma_t = \{1, 2\}$ representing which player's turn it is during a game. Let Λ be a finite set of actions. A symbolic turn-based game structure $\mathcal{G}$ defined over the set of variables $\mathcal{V}$ and the set of actions Λ is a tuple $\mathcal{G} = (\mathcal{V}, \Lambda, \tau)$ where τ is a transition relation given as a predicate over $\mathcal{V} \cup \Lambda \cup \mathcal{V}'$. We denote by $\Sigma^i_{\mathcal{V}} = \{s \in \Sigma_{\mathcal{V}} \mid s_{|t} = i\}$ the set of player-i states for $i = 1, 2$. At any state $s \in \Sigma^i_{\mathcal{V}}$, the player-$i$ chooses an action $\ell \in \Lambda$ such that there exists a successor state $s' \in \Sigma_{\mathcal{V}'}$ where $(s, \ell, s') \models \tau$. Intuitively, at a player-i state, she chooses an available action according to the transition relation τ and the next state of the system is chosen from the possible successor states. For every state $s \in \Sigma_{\mathcal{V}}$, we define $\Gamma(s) = \{\ell \in \Lambda \mid \exists s' \in \Sigma_{\mathcal{V}'}.\ (s, \ell, s') \models \tau\}$ to be the set of available actions at that state. A *run* $s_0 s_1 s_2 \cdots$ of a game structure is a sequence of states $s_i \in \Sigma_{\mathcal{V}}$ such that for all $i > 0$ there is an action $\ell \in \Lambda$ with $(s_{i-1}, \ell, s'_i) \models \tau$, where s'_i is obtained by replacing the variables in s_i by their primed copies. A run π is maximal if either it is infinite or it ends in a state $s \in \Sigma_{\mathcal{V}}$ where $\Gamma(s) = \emptyset$.

Strategies. A strategy of player-i is a partial function $f_i : (\Sigma_{\mathcal{V}})^*.\Sigma^i_{\mathcal{V}} \rightarrow \Lambda$ such that for every sequence of states $r.s \in (\Sigma_{\mathcal{V}})^*$ ending in a player-i's state $s \in \Sigma^i_{\mathcal{V}}$, if s has a successor, then $f_i(r.s)$ is defined, and $(s, f_i(r.s), u) \models \tau$ for some $u \in \Sigma_{\mathcal{V}'}$. Given two strategies f_1 and f_2 for players 1

and 2, the *possible outcomes* $\Omega_{f_1,f_2}(s)$ from a state $s \in \Sigma_{\mathcal{V}}$ are runs: a run $s_0 s_1 s_2 \cdots$ belongs to $\Omega_{f_1,f_2}(s)$ iff $s_0 = s$ and for all $j \geq 0$, either s_j has no successor, or $s_j \in \Sigma^i_{\mathcal{V}}$ and $(s_j, f_i(s_0 \cdots s_j), s'_{j+1}) \models \tau$. Strategies may need memory to remember the history of a game. Let M be a finite set called memory. A finite-memory strategy $\mathcal{S} = (m_0, f_M, f_\Lambda)$ for player-i is defined as an initial memory $m_0 \in M$ along with a pair of functions: a memory-update function $f_M : M \times \Sigma_{\mathcal{V}} \to M$, which given the current state of the game and the memory, updates the memory, and a next-action function $f_\Lambda : M \times \Sigma^i_{\mathcal{V}} \to \Lambda$, which given the current player-i state and the memory, suggests the next action for the player. A strategy $\mathcal{S}$ is memory-less (a.k.a. positional) if the memory M is a singleton, i.e., $|M| = 1$. A memory-less strategy is independent of the history of the game and only depends on the current state. Thus, a memory-less strategy for player-i can be represented as a function $\mathcal{S} : \Sigma^i_{\mathcal{V}} \to \Lambda$.

Winning condition. A game $(\mathcal{G}, \phi_{init}, \Phi)$ consists of a game structure $\mathcal{G}$, a predicate ϕ_{init} over $\mathcal{V}$ specifying a set of initial states, and an objective Φ for player-2 given as an LTL formula. A run $\pi = s_0 s_1 \cdots$ is winning for player-2 if it is infinite and $\pi \in \mathcal{L}(\Phi)$. Let Π_1 be the set of runs that are winning for player-2. A strategy f_2 is winning for player-2 if for all strategies f_1 of player-1 and all states $s \models \phi_{init}$, we have $\Omega_{f_1,f_2}(s) \subseteq \Pi_1$, i.e., all possible outcomes are winning for player-2. The set of winning states $\mathcal{W}^\Phi$ for player-2 and for objective Φ in the game structure $\mathcal{G}$ is the set of states from which player-2 has a winning strategy. We say an objective Φ is enforceable over a game structure $\mathcal{G}$ from any initial state $s \models \phi_{init}$ if and only if player-2 has a winning strategy in the game $(\mathcal{G}, \phi_{init}, \Phi)$, in which case we also say $(\mathcal{G}, \phi_{init}, \Phi)$ is realizable. A game structure $\mathcal{G}$ defined over variables $\mathcal{V}$ and action Λ is deterministic iff for any state $s_1 \in \Sigma_{\mathcal{V}}$ and any actions $\ell \in \Lambda$, the set $Succ(s_1, \ell) = \{s_2 \in \Sigma_{\mathcal{V}'} \mid (s_1, \ell, s_2) \models \tau\}$ of successor states has at most one element, i.e., $\forall s_1 \in \Sigma_{\mathcal{V}}\ \forall \ell \in \Lambda.\ |Succ(s_1, \ell)| \leq 1$. Note that we allow non-determinism in our definition of game structures and we assume that the nondeterminism is always on player-1's side.

All the definitions can be extended to parametric versions in a straightforward manner by replacing $\mathcal{V}$ by $\mathcal{V} \cup \mathcal{P}$ and $\Sigma_{\mathcal{V}}$ by $\Sigma_{\mathcal{V}} \times \Sigma_{\mathcal{P}}$. For example, a parametric finite memory strategy $\mathcal{S}^{\mathcal{P}} = (m_0, f^{\mathcal{P}}_M, f^{\mathcal{P}}_\Lambda)$ for player-2 is defined by a memory-update function $f^{\mathcal{P}}_M : M \times \Sigma_{\mathcal{V}} \times \Sigma_{\mathcal{P}} \to M$ and next-action function $f^{\mathcal{P}}_\Lambda : M \times \Sigma^2_{\mathcal{V}} \times \Sigma_{\mathcal{P}} \to \Lambda$. For a parametric strategy $\mathcal{S}^{\mathcal{P}}$ and a parameters valuation $p \in \Sigma_{\mathcal{P}}$, *instantiation* of $\mathcal{S}^{\mathcal{P}}$ by p, denoted by $\mathcal{S}^{\mathcal{P}}_{\downarrow p}$, is a non-parametric finite-memory strategy $\mathcal{S}^{\mathcal{P}}_{\downarrow p} = (m_0, f_M, f_\Lambda)$ where for all $m \in M$ and $s \in \Sigma_{\mathcal{V}}$, $f_M(m, s) = f^{\mathcal{P}}_M(m, s, p)$ and $f_\Lambda(m, s) = f^{\mathcal{P}}_\Lambda(m, s, p)$.

Solving games. Symbolic algorithms for solving the realizability and synthesis problems are based on the *controllable predecessor* operator [13]. The (player-2) controllable predecessor operator $CPre : 2^{\Sigma_{\mathcal{V}}} \to 2^{\Sigma_{\mathcal{V}}}$ maps a set $Z \subseteq \Sigma_{\mathcal{V}}$ of states to the states from which player-2 can force the game into Z in one step. Player-2 can force the game into Z from a state $s \in \Sigma^1_{\mathcal{V}}$ iff for all available moves ℓ, all ℓ-successors of s are in Z, and she can force the game into Z from a state $s \in \Sigma^2_{\mathcal{V}}$ iff there is *some* available action ℓ such that all ℓ-successors of v are in Z. For example, the set of states from which player-2 can avoid a set $[\![\Phi_{err}]\!] \subseteq \Sigma_{\mathcal{V}}$ of states is the greatest fixed point $\nu Z.[\![\neg\Phi_{err}]\!] \cap CPre(Z)$ (safety objective,) and the set of states from which player-2 can reach a set of states $[\![\Phi_{reach}]\!] \subseteq \Sigma_{\mathcal{V}}$ is the least fixed point $\mu Z.[\![\Phi_{reach}]\!] \cup CPre(Z)$ (reachability objective).

2.3 Controllers and Controller Interfaces

Controller. We refer to memory-less strategies for player-2 with a designated set of final states as *finite-horizon reactive controllers* (or *controllers* for short). In our setting, controllers become active for a finite number of steps and interact with environment until reaching a desirable target state while avoiding some specified error states. Formally, a controller $\mathcal{C}$ is a pair $(\mathcal{S}, \mathcal{F})$ where $\mathcal{S} : \Sigma_{\mathcal{V}} \to \Lambda$ is a memory-less strategy and $\mathcal{F} \subseteq \Sigma_{\mathcal{V}}$ is a designated set of final states. At any time-step, if current state $s \in \Sigma_{\mathcal{V}}$ is a final state, i.e., $s \in \mathcal{F}$, the controller has reached the end of its computation. Note that we only consider controllers with reachability and safety objectives for which memory-less strategies suffice. A parametric reactive controller is a controller whose strategy and set of final states are parametric. Given a parameter valuation $p \in \Sigma_p$ and a parametric controller $\mathcal{C} = (\mathcal{S}, \mathcal{F})$, instantiation of $\mathcal{C}$ with $p \in \mathcal{P}$ is the controller $\mathcal{C}_{\downarrow p} = (\mathcal{S}_{\downarrow p}, \mathcal{F}_{\downarrow p})$ obtained by instantiating the strategy and projecting the set of final states by p.

Controller interface. A controller interface abstracts a controller by providing high-level information about its behavior while hiding its actual implementation. Formally, a controller interface $\mathcal{I}_{\mathcal{C}} = (\phi_{init_{\mathcal{C}}}, \phi_{inv_{\mathcal{C}}}, \phi_{f_{\mathcal{C}}})$ for a controller $\mathcal{C}$ is a tuple where $\phi_{init_{\mathcal{C}}}$ is a set of initial valuations over variables (and parameters), $\phi_{inv_{\mathcal{C}}}$ is an invariant that holds over all possible runs of $\mathcal{C}$ while it has the control, $\phi_{f_{\mathcal{C}}}$ is a possible set of valuations over variables (and parameters) once $\mathcal{C}$ reaches a final state. A controller $\mathcal{C} = (\mathcal{S}, \mathcal{F})$ over a game structure $\mathcal{G}$ *realizes* a controller interface $\mathcal{I}_{\mathcal{C}}$ if $\mathcal{S}$ is a winning strategy for the game $(\mathcal{G}, \phi_{init_{\mathcal{C}}}, \Psi)$ where $\Psi = \phi_{inv_{\mathcal{C}}}\ \mathcal{U}\ (\phi_{inv_{\mathcal{C}}} \wedge \phi_{f_{\mathcal{C}}})$ and $\mathcal{F} \subseteq [\![\phi_{f_{\mathcal{C}}}]\!]$, i.e., starting from any initial state $s \models \phi_{init_{\mathcal{C}}}$, the controller $\mathcal{C}$ guarantees that eventually a final state $s_f \models \phi_{f_{\mathcal{C}}}$ is visited and besides all the visited states along any possible outcome satisfy $\phi_{inv_{\mathcal{C}}}$, i.e., only safe states are visited. Instantiation of a controller interafce $\mathcal{I}_{\mathcal{C}}$ by $p \in \Sigma_{\mathcal{P}}$ is the non-parametric controller interface $\mathcal{I}_{\mathcal{C}_{\downarrow p}} = (\phi_{init_{\mathcal{C}_{\downarrow p}}}, \phi_{inv_{\mathcal{C}_{\downarrow p}}}, \phi_{f_{\mathcal{C}_{\downarrow p}}})$. A parameter valuation $p \in \Sigma_{\mathcal{P}}$ is *admissible* for controller $\mathcal{C}$ with interface $\mathcal{I}_{\mathcal{C}}$ over a game structure $\mathcal{G}$ iff the instantiation of $\mathcal{C}$ by p, $\mathcal{C}_{\downarrow p}$, realizes the non-parametric interface $\mathcal{I}_{\mathcal{C}_{\downarrow p}}$. Intuitively, a parametric controller can be instantiated by any admissible parameter value, and enforce its safety and reachability objectives, provided that its execution starts from a valid initial state. A set $\Sigma^a_{\mathcal{P}} \subseteq \Sigma_{\mathcal{P}}$ of admissible parameter values is maximal, if for any parameter valuation $p \in \Sigma_{\mathcal{P}} \backslash \Sigma^a_{\mathcal{P}}$, $\mathcal{C}_{\downarrow p}$ does *not* realize $\mathcal{I}_{\mathcal{C}_{\downarrow p}}$. A controller interface $\mathcal{I}_1 = (\phi_{init_1}, \phi_{inv_1}, \phi_{f_1})$ *respects* a controller interface $\mathcal{I}_2 = (\phi_{init_2}, \phi_{inv_2}, \phi_{f_2})$ if $\phi_{init_1} \rightarrow \phi_{init_2}$, $\phi_{inv_1} \rightarrow \phi_{inv_2}$, and $\phi_{f_1} \rightarrow \phi_{f_2}$. Note that any controller that realizes $\mathcal{I}_1$, also realizes the restricted interface $\mathcal{I}'_2 = (\phi_{init_1}, \phi_{inv_2}, \phi_{f_2})$, where $\mathcal{I}'_2$ is obtained from the interface $\mathcal{I}_2$ by restricting its initial states to $[\![\phi_{init_1}]\!] \subseteq [\![\phi_{init_2}]\!]$. In our setting, the designer can specify a parametric interface for the controllers without knowing for what parameter valuations the controller can enforce its safety and reachability objectives. A parametric controller, a maximal set of admissible parameter values, and an interface that respects the user-specified interface are then synthesized automatically.

2.4 Composing Controllers

Let $\mathcal{G} = (\mathcal{V}, \Lambda, \tau)$ be a game structure, and $\Gamma_{\mathcal{C}} = \{\mathcal{C}_1, \cdots \mathcal{C}_n\}$ be a set of parametric controllers. For a given set of initial states ϕ_{init} and objective Φ, the goal of the composer is to iteratively select a parametric controller and instantiate it with a parameter valuation, delegate the control to the instantiated controller until it enters a final state and relinquishes the control, upon which the composer selects the next controller and the next parameter valuation, and the process is repeated such that the objective Φ is enforced starting from any initial state $s_{init} \models \phi_{init}$. A *control strategy* $\mathcal{S}^{\mathcal{C}} : \Sigma_{\mathcal{V}} \rightarrow \Sigma_{\mathcal{P}} \times \Gamma_{\mathcal{C}}$ is a (partial) function that maps states of the game to a controller and a parameter valuation (note that we do not consider memory for the control strategy since it is not needed for safety and reachability objectives). A control strategy $\mathcal{S}^{\mathcal{C}}$ induces a finite-memory strategy $\mathcal{S} = (m_0, f_M, f_\Lambda)$ obtained by *sequentially* composing instantiated controllers according to $\mathcal{S}^{\mathcal{C}}$ as follows. Let $M \subseteq \Sigma_{\mathcal{P}} \times \Gamma_{\mathcal{C}} \cup \{\perp\}$ be the memory of the strategy where $m_0 = \perp$ and $\perp$ is a special symbol indicating the initial memory where a controller and a parameter valuation is yet to be selected. Intuitively, the memory of the strategy keeps track of the controller that currently has the control and the parameter valuation used to instantiate it. The memory-update function $f_M : M \times \Sigma_{\mathcal{V}} \rightarrow M$ and the next-action function $f_\Lambda : M \times \Sigma^2_{\mathcal{V}} \rightarrow \Lambda$ are defined as

$$f_M(m, s) = \begin{cases} m & \text{if } m \neq \perp \wedge s \notin \mathcal{F}_{\mathcal{C}_m} \\ \mathcal{S}^{\mathcal{C}}(s) & \text{otherwise} \end{cases}$$

$$f_\Lambda(m, s) = \begin{cases} \mathcal{S}_{\mathcal{C}_m}(s) & \text{if } m \neq \perp \wedge s \notin \mathcal{F}_{\mathcal{C}_m} \\ \mathcal{S}_{\mathcal{C}_{next}}(s) & \text{otherwise} \end{cases}$$

where $\mathcal{C}_m = (\mathcal{S}_{\mathcal{C}_m}, \mathcal{F}_{\mathcal{C}_m}) = \mathcal{C}_{i_\downarrow p}$ is the instantiated controller for the memory $m = (p, \mathcal{C}_i)$, and $\mathcal{C}_{next} = (\mathcal{S}_{\mathcal{C}_{next}}, \mathcal{F}_{\mathcal{C}_{next}}) = \mathcal{C}^{next}_{\downarrow p^{next}}$ with $\mathcal{S}^{\mathcal{C}}(s) = (p^{next}, \mathcal{C}^{next})$ is the next controller chosen by the control strategy. Intuitively, when a final state of the currently active controller is reached or initially when no controller is selected, the next controller and the next parameter valuation are chosen according to the control strategy and the memory is updated to reflect this selection. The selected and instantiated controller then becomes active and guides the actions of the system while the memory stays unchanged, until the active controller enters a final state, upon which the control strategy decides the next action and the process is repeated. Note that in practice, the induced strategy $\mathcal{S}$ from $\mathcal{S}^{\mathcal{C}}$ is not computed explicitly, and the controllers can be dynamically fetched, instantiated and executed according to the control strategy.

3. PROBLEM STATEMENTS AND OVERVIEW

In this section we formally define the problems we consider in this paper and give an overview of our solution approach. Let $\mathcal{V}$ and $\mathcal{P}$ be sets of variables and parameters defined over finite domains $\Sigma_{\mathcal{V}}$ and $\Sigma_{\mathcal{P}}$, respectively, Λ be a finite set of actions, and $\mathcal{G}$ be a game structure over $\mathcal{V}$ and Λ. We are interested in how a parametric controller can be synthesized from a given parametric controller interface. Formally,

PROBLEM STATEMENT 1. *(Synthesis of Parametric Reactive Controllers.) Given a game structure $\mathcal{G}$ and a parametric controller interface $\mathcal{I} = (\phi_{init}, \phi_{inv}, \phi_f)$, synthesize a parametric reactive controller $\mathcal{C}$, its corresponding interface $\mathcal{I}_{\mathcal{C}}$, and a maximal set $\Sigma^a_{\mathcal{P}} \subseteq \Sigma_{\mathcal{P}}$ of admissible parameter valuations such that $\mathcal{I}_{\mathcal{C}}$ respects $\mathcal{I}$, and for any admissible parameter valuation $p \in \Sigma^a_{\mathcal{P}}$ for $\mathcal{C}$, instantiation of $\mathcal{C}$ by p, $\mathcal{C}_{\downarrow p}$, realizes the instantiated controller interface $\mathcal{I}_{\mathcal{C}_{\downarrow p}}$.*

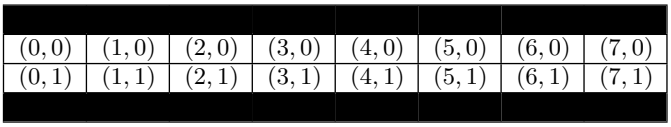

Figure 2: Part of a road divided into grids.

A designer can specify a set of parametric controller interfaces. The synthesis algorithm then automatically computes the set of controllers, their corresponding interfaces and admissible parameter values. Once the parametric controllers are computed, they can be reused in different compositions to synthesize control strategies for different objectives.

Once a library of parametric controllers and their corresponding interfaces are obtained, the next natural question is how they can be composed to enforce high-level objectives. Let ϕ_{init} be a non-parametric predicate specifying initial states of the game, Φ be a non-parametric LTL objective over $\mathcal{V}$, $\Gamma_{\mathcal{C}} = \{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$ be a set of parametric controllers, and $\Gamma_{\mathcal{I}_{\mathcal{C}}} = \{\mathcal{I}_{\mathcal{C}_1}, \cdots, \mathcal{I}_{\mathcal{C}_n}\}$ be the set of corresponding controller interfaces. Our goal is to synthesize a control strategy $\mathcal{S}^{\mathcal{C}}$ that instantiates and composes controllers from $\Gamma_{\mathcal{C}}$ using the information provided through interfaces $\Gamma_{\mathcal{I}_{\mathcal{C}}}$ such that its induced strategy enforces the global objective Φ in the game $(\mathcal{G}, \phi_{init}, \Phi)$. Formally,

PROBLEM STATEMENT 2. *(Synthesis with Parametric Reactive Controllers.) Given a game structure $\mathcal{G}$, a set of initial states specified by a non-parametric predicate ϕ_{init}, a non-parametric LTL objective Φ, and a set of parametric controllers $\Gamma_{\mathcal{C}}$ and their corresponding interfaces $\Gamma_{\mathcal{I}_{\mathcal{C}}}$, compute a control strategy $\mathcal{S}^{\mathcal{C}}$, if one exists, such that its induced strategy $\mathcal{S}$ is winning in the game $(\mathcal{G}, \phi_{init}, \Phi)$.*

We assume that Φ is given as a safety and/or reachability objective. We illustrate the methods with a simple example.

EXAMPLE 2. *Consider a block of a double-lane road divided into grids each identified by a tuple (x, y) as shown in Figure 2. Assume there is a controlled vehicle V_1 initially at $(x_1, y_1) = (0, 1)$ moving from left to right. Moreover, assume there is an uncontrolled vehicle V_2 initially at $(x_2, y_2) = (7, 1)$ moving from right to left while staying on the same lane at all times, i.e., always $y_2 = 1$. Formally, let $\phi_{init} = (x_1 = 0 \wedge y_1 = 1 \wedge x_2 = 7 \wedge y_2 = 1)$ be the predicate specifying the initial state of the system. Assume V_1 has two actions: move-forward action, ℓ_1, that moves the vehicle one step ahead by incrementing x_1 while keeping it on the same lane, and lane-switch action, ℓ_2, that moves the vehicle one step forward and changes the lane at the same time. Our goal is to synthesize a controller that guides V_1 from the starting point to the other end of the road without colliding with V_2. This objective can be specified with the formula $\Phi = \phi_1 \; \mathcal{U} \; (\phi_1 \wedge \phi_2)$ where $\phi_1 = (x_1 \neq x_2 \vee y_1 \neq y_2)$ (no collision) and $\phi_2 = (x_1 = 7)$ (reaching the other end.)*

Let a and b be two parameters. Assume the designer specifies a parametric controller interface $\mathcal{I} = (\phi_{init}, \phi_{inv}, \phi_f)$ where $\phi_{init} = (x_1 = a) \wedge (y_1 = b)$, $\phi_{inv} = (x_1 \neq x_2) \vee (y_1 \neq$

y_2), *and* $\phi_f = (x_1 = a + 1)$, *i.e., starting from initial parametric state* $(x_1, y_1) = (a, b)$, V_1 *must move one step forward (to satisfy* ϕ_f*) while avoiding collision with* V_2 *(thus satisfying* ϕ_{inv}*). A parametric controller* $\mathcal{C} = (\mathcal{S}, \mathcal{F})$ *is then synthesized with a memory-less strategy* $\mathcal{S}$ *defined as*

$$\mathcal{S}(x_1, y_1, x_2, y_2, a, b) = \begin{cases} \ell_2 & \text{if} \quad 0 \leq a \leq 6 \wedge x_1 = a \wedge y_1 = b \\ & \quad \wedge y_1 = y_2 \wedge x_2 = a + 1 \\ \ell_1 & \text{if} \quad 0 \leq a \leq 6 \wedge x_1 = a \wedge y_1 = b \wedge \\ & \quad (x_1 \neq x_2 \vee (y_2 \neq b \wedge y_2 \neq b + 1)) \end{cases}$$

Intuitively, the controller $\mathcal{C}$ *switches the current lane of the vehicle* V_1 *by taking lane-switch action* ℓ_2 *if the other vehicle* V_2 *is on the same lane and one cell ahead of* V_1*, and otherwise keeps moving forward by taking move-forward action* ℓ_1*. This way the controller* $\mathcal{C}$ *ensures that* V_1 *eventually makes progress by incrementing* x_1 *while avoiding collision with the other vehicle. For the set of final states of* $\mathcal{C}$ *we have* $\mathcal{F} = (0 \leq a \leq 6 \wedge x_1 = a + 1 \wedge ((x_1 \neq x_2) \vee (y_1 \neq y_2)))$*, i.e., once the controller reaches a final state,* V_1 *has moved one step forward and does not occupy the same grid with* V_2*. Besides, correct behavior of the controller is guaranteed for the parameter values* $0 \leq a \leq 6$*. A potential controller interface* $\mathcal{I}_\mathcal{C}$ *for* $\mathcal{C}$ *is* $(\phi'_{init}, \phi_{inv}, \phi_f)$ *where* $\phi'_{init} = \phi_{init} \wedge 0 \leq a \leq 6$*. Note that* $\mathcal{I}_\mathcal{C}$ *respects* $\mathcal{I}$ *and* $\mathcal{C}$ *realizes* $\mathcal{I}_\mathcal{C}$*.*

Once the parametric controllers are synthesized and a library is formed, the next step is to instantiate right parametric controllers and compose them to enforce a given system objective. In the above example, the controller $\mathcal{C}$ *can be instantiated and composed sequentially in order to enforce the objective* Φ *according to the memory-less control strategy* $\mathcal{S}^\mathcal{C}(x_1, y_1, x_2, y_2) = ((x_1, y_1), \mathcal{C})$ *if* $0 \leq x_1 \leq 6$*. Intuitively, while* V_1 *has not reached the end of the road (i.e.,* $x_1 \neq 7$*), the control strategy selects* $\mathcal{C}$ *and instantiates it with* $(a = x_1, b = y_1)$*, i.e.,* V_1*'s current location. To enforce the objective* Φ*, the parametric controller* $\mathcal{C}$ *is instantiated and composed 7 times, where each controller moves the vehicle one step forward without colliding with the other vehicle.*

4. SYNTHESIZING PARAMETRIC REACTIVE CONTROLLERS

In this section we describe our solution for Problem 1 stated in Section 3. Let $\mathcal{G} = (\mathcal{V}, \Lambda, \tau)$ be a game structure, and $\mathcal{I} = (\phi_{init}, \phi_{inv}, \phi_f)$ be the user-specified controller interface. Our goal is to synthesize a controller $\mathcal{C}$ and its corresponding controller interface $\mathcal{I}_\mathcal{C} = (\phi_{init_\mathcal{C}}, \phi_{inv_\mathcal{C}}, \phi_{f_\mathcal{C}})$ and a set $\Sigma^a_\mathcal{P}$ of admissible parameter values such that for any $p \in \Sigma^a_\mathcal{P}$, $\mathcal{C}_{\downarrow p}$ realizes $\mathcal{I}_{\mathcal{C}_{\downarrow p}}$ and $\mathcal{I}_\mathcal{C}$ respects $\mathcal{I}$.

To this end, we first obtain a *parametric* game structure $\mathcal{G}^\mathcal{P}$ from $\mathcal{G}$. The idea is to treat parameters as special variables that have unknown initial value in a bounded set, but their value stays constant over the transitions of the game structure. Formally, let $\mathcal{P}'$ be a primed copy of parameters, and assume $same(\mathcal{P}, \mathcal{P}')$ is a predicate stating that the value of parameters stay unchanged. The parametric game structure $\mathcal{G}^\mathcal{P}$ is defined as $(\mathcal{V} \cup \mathcal{P}, \Lambda, \tau^\mathcal{P})$ where $\tau^\mathcal{P} = \tau \wedge same(\mathcal{P}, \mathcal{P}')$.

For example, Figure 3a shows a game structure where player-1 (player-2) states are depicted by ovals (boxes, respectively.) Each state is labeled by a state name q_i and a valuation over a variable x with domain $\Sigma_x = [0..3]$. At each player-i state for $i = 1, 2$, the player can choose one of the actions *inc* or *dec* (if available,) to increment or decrement x, respectively. Assume p is a parameter with domain $\Sigma_p = [0..2]$. Figure 3b shows the parametric game structure obtained from the game structure in Figure 3a. Each state is labeled with a state name q_i^j and a valuation over x and p. Each state q_i^j in the parametric game structure $\mathcal{G}^\mathcal{P}$ correspond to the state q_i in the game structure $\mathcal{G}$. Intuitively, the parametric game structure has parallel copies of the non-parametric game structure for different values of the parameters and moreover, there is no transition between different copies. Note that explicit-state representations of (parametric) game structures are *not* constructed in practice, and they are represented and manipulated symbolically, thus avoiding the explicit enumeration of the parameters.

Algorithm 1: Parametric controller synthesis

Input: Game structure $\mathcal{G} = (\mathcal{V}, \Lambda, \tau)$, controller interface $\mathcal{I} = (\phi_{init}, \phi_{inv}, \phi_f)$ and parameters $\mathcal{P}$
Output: Parametric controller $\mathcal{C}$, controller interface $\mathcal{I}_\mathcal{C}$, and admissible parameter values $\Sigma^a_\mathcal{P}$ s.t. $\mathcal{I}_\mathcal{C}$ respects $\mathcal{I}$ and $\forall p \in \Sigma^a_\mathcal{P}$. $\mathcal{C}_{\downarrow p}$ realizes $\mathcal{I}_{\mathcal{C}_{\downarrow p}}$.

1 $\tau^\mathcal{P} := \tau \wedge same(\mathcal{P}, \mathcal{P}')$;
2 $\mathcal{G}^\mathcal{P} := (\mathcal{V} \cup \mathcal{P}, \Lambda, \tau^\mathcal{P})$;
3 $\Phi_\mathcal{I} := \phi_{inv}\ \mathcal{U}\ (\phi_{inv} \wedge \phi_f)$;
4 Let $[\![\phi_\mathcal{W}]\!]$ be the set of winning states in $\mathcal{G}^\mathcal{P}$ with respect to $\Phi_\mathcal{I}$;
5 $\phi_{init_\mathcal{C}} := \phi_{init} \wedge \phi_\mathcal{W}$;
6 $\phi^a_\mathcal{P} := \exists \mathcal{V}.\ \phi_{init_\mathcal{C}}$;
7 $\Sigma^a_\mathcal{P} := [\![\phi^a_\mathcal{P}]\!]$;
8 Let $\mathcal{S}$ be a parametric winning strategy in the game $(\mathcal{G}^\mathcal{P}, \phi_{init_\mathcal{C}}, \Phi_\mathcal{I})$;
9 $\phi_\mathcal{R} := \mathbf{Reachable}(\mathcal{G}^\mathcal{P}, \phi_{init_\mathcal{C}}, \mathcal{S})$;
10 $\mathcal{F} := [\![\phi_f \wedge \phi_\mathcal{R}]\!]$;
11 $\mathcal{C} := (\mathcal{S}, \mathcal{F})$;
12 $\phi_{inv_\mathcal{C}} := \phi_\mathcal{R}$;
13 $\phi_{f_\mathcal{C}} := \phi_f \wedge \phi_\mathcal{R}$;
14 $\mathcal{I}_\mathcal{C} = (\phi_{init_\mathcal{C}}, \phi_{inv_\mathcal{C}}, \phi_{f_\mathcal{C}})$;
15 return $(\mathcal{C}, \mathcal{I}_\mathcal{C}, \Sigma^a_\mathcal{P})$;

Algorithm 1 shows how a parametric controller is synthesized for a given game structure $\mathcal{G}$ and specified interface $\mathcal{I}$. Once the parametric game structure $\mathcal{G}^\mathcal{P}$ is obtained, the game $(\mathcal{G}^\mathcal{P}, \phi_{init}, \Phi_\mathcal{I})$ where $\Phi_\mathcal{I} = \phi_{inv}\ \mathcal{U}\ (\phi_{inv} \wedge \phi_f)$ can be solved by standard realizability and synthesis algorithms and a set of winning states can be computed [13]. Let $\mathcal{W} \subseteq \Sigma_\mathcal{V} \times \Sigma_\mathcal{P}$ be the set of winning states in $\mathcal{G}^\mathcal{P}$ with respect to objective $\Phi_\mathcal{I}$, and let $\phi_\mathcal{W}$ be a predicate specifying $\mathcal{W}$, i.e., $[\![\phi_\mathcal{W}]\!] = \mathcal{W}$. We define $\phi_{init_\mathcal{C}} = \phi_{init} \wedge \phi_\mathcal{W}$ as the intersection of set of parametric initial states specified by the user and set of winning states where player-2 can enforce the objective $\Phi_\mathcal{I}$. The set $[\![\phi_{init_\mathcal{C}}]\!]$ includes all the parametric initial states from which player-2 can win the game $(\mathcal{G}^\mathcal{P}, \phi_{init_\mathcal{C}}, \Phi_\mathcal{I})$ and hence, it contains all the admissible parameter valuations. The set $\Sigma^a_\mathcal{P}$ of admissible parameter values can be computed by existentially quantifying the variables from $\phi_{init_\mathcal{C}}$, i.e., $\Sigma^a_\mathcal{P} = \exists \mathcal{V}.\ \phi_{init_\mathcal{C}}$. Algorithm 1 then computes a parametric winning strategy over the game $(\mathcal{G}^\mathcal{P}, \phi_{init_\mathcal{C}}, \Phi_\mathcal{I})$ using a game solver. Let $\phi_\mathcal{R} = \mathbf{Reachable}(\mathcal{G}^\mathcal{P}, \phi_{init_\mathcal{C}}, \mathcal{S})$ be a predicate specifying the set $[\![\phi_\mathcal{R}]\!] \subseteq \Sigma_\mathcal{V} \times \Sigma_\mathcal{P}$ of reachable states in the parametric game structure $\mathcal{G}^\mathcal{P}$ starting from any initial state $s \models \phi_{init_\mathcal{C}}$

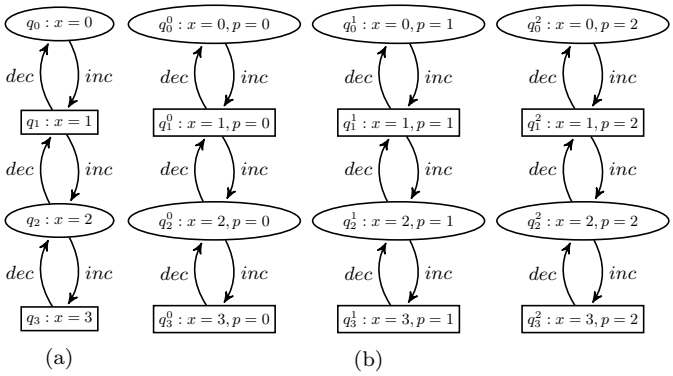

Figure 3: (a) A game structure $\mathcal{G}$ defined over a variable $x \in [0..3]$, and (b) A parametric game structure $\mathcal{G}^{\mathcal{P}}$ obtained from $\mathcal{G}$ with parameter $p \in [0..2]$.

when player-2 actions are chosen according to the strategy $\mathcal{S}$. We define $\mathcal{I}_{\mathcal{C}} = (\phi_{init_{\mathcal{C}}}, \phi_{inv_{\mathcal{C}}}, \phi_{f_{\mathcal{C}}})$ as the controller interface for $\mathcal{C}$ where $\phi_{inv_{\mathcal{C}}} = \phi_{\mathcal{R}}$ and $\phi_{f_{\mathcal{C}}} = \phi_f \wedge \phi_{\mathcal{R}}$. Intuitively, $\phi_{inv_{\mathcal{C}}}$ specifies the set of states that may be visited during the game when player-2 behaves according to the controller $\mathcal{C}$, and serves as the invariant of the computed controller interface. Similarly, $\phi_{f_{\mathcal{C}}}$ specifies a set of reachable final states when $\mathcal{C}$ is active. The following theorem states that Algorithm 1 can correctly synthesize a parametric controller, if one exists, and that it computes the controller with effort $O(|\Sigma_{\mathcal{V}}|)$, where effort is measured in symbolic steps, i.e., in the number of pre-image or post-image computation [3]. Due to the lack of space, the proofs of theorems are omitted and can be found in the technical report.

THEOREM 1. *Algorithm 1 is sound and complete. It performs $O(|\Sigma_{\mathcal{V}}|)$ symbolic steps in the worst case.*

The number of symbolic steps is not the only factor determining the time taken by the symbolic algorithm, however, it is an important measure of difficulty since image and pre-image computations are typically the most expensive operations [3]. Intuitively, the number of symbolic steps in Algorithm 1 is independent of the parameters because the symbolic algorithm for computing the set of winning states can manipulate the parallel copies in the parametric game structure simultaneously, and that each copy is of size $O(|\Sigma_{\mathcal{V}}|)$ (the parametric game structure can be viewed as $|\Sigma_{\mathcal{P}}|$ copies of the original game structure.) As an example, consider the parametric game structure in Figure 3b. Observe that each copy for each parameter valuation is of the same size of the non-parametric game structure shown in Figure 3a. Let $\Phi = (X = p)$ be a parametric predicate. It is easy to see that states q_0^0, q_1^1, and q_2^2 in the parametric game structure satisfy Φ. The pre-image of these states (states that can reach them in one step) is the set $\{q_0^1, q_1^0, q_2^1, q_1^2, q_3^2\}$ that is computed by the symbolic algorithm in one step. Note that although the number of symbolic steps in Algorithm 1 is independent of the parameters, it does not mean that adding parameters has no additional cost as they may increase the complexity of the symbolic steps. However, transitions over parameters have a special structure that may be utilized for efficient implementation of the symbolic steps, which is subject to our future research.

5. SYNTHESIS OF CONTROL STRATEGY WITH PARAMETRIC CONTROLLERS

In this section we describe our solution for Problem 2 stated in Section 3. Our goal is to synthesize a control strategy $\mathcal{S}^{\mathcal{C}}$ such that its induced strategy is winning in the game $(\mathcal{G}, \phi_{init}, \Phi)$. To this end, we first obtain a *control game structure* $\mathcal{G}^{\mathcal{C}}$ using the set of controller interfaces $\Gamma_{\mathcal{I}_{\mathcal{C}}}$. Intuitively, $\mathcal{G}^{\mathcal{C}}$ models what controllers and parameter valuations the system can choose at any state, possible states that may be visited while the selected controller is active, and potential final states that may be reached once the controller is done. From the standpoint of the composer, each instantiated controller that becomes active goes through three steps: *initialization* (the controller starts its execution from a valid initial state), *execution* (the state of the system evolves according to the controller), and *termination* (the controller enters a final state and returns the control).

Formally, let $\gamma_{\mathcal{C}} \notin \mathcal{V}$ defined over the domain $\Sigma_{\gamma_{\mathcal{C}}} = [1..n]$ be a variable representing the controllers, i.e., $\gamma_{\mathcal{C}} = i$ corresponds to the controller $\mathcal{C}_i \in \Gamma_{\mathcal{C}}$ for $i = 1, \cdots, n$. Let $t_c \notin \mathcal{V}$ defined over $\Sigma_{t_c} = \{1, 2\}$ be a variable indicating which player's turn it is in the control game structure. Moreover, let $t_e \notin \mathcal{V}$ defined over $\Sigma_{t_e} = \{1, 2\}$ be an additional variable that player-1 uses to distinguish a controller's possible initial states from *intermediate* states that may be visited during the execution of the controller. A control game structure $\mathcal{G}^{\mathcal{C}}$ is a tuple $(\mathcal{V}^{\mathcal{C}}, \Lambda^{\mathcal{C}}, \tau^{\mathcal{C}})$ where $\mathcal{V}^{\mathcal{C}} = \mathcal{V} \cup \{t_c, t_e, \gamma_{\mathcal{C}}\} \cup \mathcal{P}$ is a set of variables defined over the domain $\Sigma_{\mathcal{V}^{\mathcal{C}}} = \Sigma_{\mathcal{V}} \times \Sigma_{t_c} \times \Sigma_{t_e} \times \Sigma_{\gamma_{\mathcal{C}}} \times \Sigma_{\mathcal{P}}$, $\Lambda^{\mathcal{C}} = \Sigma_{\mathcal{P}'} \times \Sigma_{\gamma'_{\mathcal{C}}}$ is a set of actions, and $\tau^{\mathcal{C}}$ is symbolically defined as $\tau^{\mathcal{C}} = \bigvee_{i=1}^{n} (\tau_s^{\mathcal{C}_i} \vee \tau_{e1}^{\mathcal{C}_i} \vee \tau_{e2}^{\mathcal{C}_i})$ where

$$\tau_s^{\mathcal{C}_i} := t_c = 2 \wedge same(\mathcal{V}, \mathcal{V}') \wedge t'_c = 1 \wedge t'_e = 1 \wedge \gamma'_c = i \wedge \phi'_{init_{\mathcal{C}_i}},$$

$$\tau_{e1}^{\mathcal{C}_i} := t_c = 1 \wedge \gamma_c = i \wedge t_e = 1 \wedge t'_c = 1 \wedge t'_e = 2 \wedge \phi'_{inv_{\mathcal{C}_i}} \wedge same(\mathcal{P}, \mathcal{P}') \wedge same(\gamma_c, \gamma'_c), \text{ and}$$

$$\tau_{e2}^{\mathcal{C}_i} := t_c = 1 \wedge t_e = 2 \wedge \gamma_c = i \wedge t'_c = 2 \wedge \phi'_{f_{\mathcal{C}_i}} \wedge same(\mathcal{P}, \mathcal{P}') \wedge same(\gamma_c, \gamma'_c)$$

where $\gamma'_{\mathcal{C}}$ is a primed copy of $\gamma_{\mathcal{C}}$, and ϕ' is obtained by replacing variables in ϕ by their primed copies. Note that the primed copies of the parameters and $\gamma_{\mathcal{C}}$ encode the actions $\Lambda^{\mathcal{C}}$ of the control game structure to indicate that the composer (player-2 in $\mathcal{G}^{\mathcal{C}}$) selects the parameter valuation and the parametric controller when it is her turn, and also to avoid introducing additional variables. We denote by $\Sigma^i_{\mathcal{V}^{\mathcal{C}}} = \{v^{\mathcal{C}} \in \Sigma_{\mathcal{V}^{\mathcal{C}}} \mid v^{\mathcal{C}}_{|t_c} = i\}$ the set of player-i states in the control game structure for $i = 1, 2$.

At any player-2 state in the control game structure, the composer must choose a controller $\mathcal{C} \in \Gamma_{\mathcal{C}}$ and an admissible parameter valuation $p \in \Sigma_{\mathcal{P}}$, if one exists. Furthermore, the composer must ensure that the selected controller starts from a valid initial state, i.e., the state where the instantiated controller $\mathcal{C}_{\downarrow p}$ receives the control satisfies the predicate $\phi_{init_{\mathcal{C}_{\downarrow p}}}$. This is captured in the predicate $\tau_s^{\mathcal{C}_i}$ of $\tau^{\mathcal{C}}$ for each controller $\mathcal{C}_i$. According to $\tau_s^{\mathcal{C}_i}$ at any state $(v, t_c = 2, t_e, \gamma_{\mathcal{C}}, p) \in \Sigma^2_{\mathcal{V}^{\mathcal{C}}}$, the controller $\mathcal{C}_i$ can be chosen by selecting $\gamma'_{\mathcal{C}} = i$ if there exists a parameter valuation $p' \in \Sigma_{\mathcal{P}'}$ such that the initial condition of the controller is satisfied, i.e., $(v', p') \models \phi'_{init_{\mathcal{C}_i}}$ where v' is obtained by replacing variables in v by their primed copies. $t'_c = 1$ means that it is player-1 state in the next turn, and $t'_e = 1$ means

that player-1 states in the next turn satisfy the initial condition of the controller. Intuitively, each predicate $\tau_s^{\mathcal{C}_i}$ for $i = 1..n$ models initialization of a controller $\mathcal{C}_i$.

Once a controller $\mathcal{C}_i$ and a parameter valuation $p \in \Sigma_{\mathcal{P}}$ are selected by the composer, the control is transferred to the instantiated controller $\mathcal{C}_{i_{\downarrow p}}$, and the controller and parameter valuation are fixed until the control is returned to the composer. This is captured in $\tau_{e_1}^{\mathcal{C}_i}$ and $\tau_{e_2}^{\mathcal{C}_i}$ by $same(\mathcal{P}, \mathcal{P}') \wedge same(\gamma_{\mathcal{C}}, \gamma'_{\mathcal{C}})$. Player-1 states with $t_e = 1$ ($t_e = 2$) and $\gamma_{\mathcal{C}_i} = i$ in $\mathcal{G}^{\mathcal{C}}$ represent initial (intermediate) states where the predicate $\phi_{init_{\mathcal{C}_{\downarrow p}}}$ ($\phi_{inv_{\mathcal{C}_{\downarrow p}}}$, respectively) of the instantiated controller interface $\mathcal{I}_{\mathcal{C}_{i_{\downarrow p}}}$ is satisfied. Intuitively, each predicate $\tau_{e_1}^{\mathcal{C}_i}$ captures transitions from controller's initial state to its intermediate states (representing the execution of the controller), and $\tau_{e_2}^{\mathcal{C}_i}$ shows the transition from intermediate states to final states (modeling termination) where the controller has reached a final state and control is returned to the composer. We illustrate the ideas with a simple example.

EXAMPLE 3. *Let $x \in [0..2]$ be a variable, and $p \in [0..2]$ be a parameter. Consider two controllers $\mathcal{C}_1$ and $\mathcal{C}_2$ with controller interfaces $\mathcal{I}_{\mathcal{C}_1} = (\phi_{init_{\mathcal{C}_1}}, \phi_{inv_{\mathcal{C}_1}}, \phi_{f_{\mathcal{C}_1}})$ and $\mathcal{I}_{\mathcal{C}_2} = (\phi_{init_{\mathcal{C}_2}}, \phi_{inv_{\mathcal{C}_2}}, \phi_{f_{\mathcal{C}_2}})$ defined as follows: $\phi_{init_{\mathcal{C}_1}} = (\phi_{\mathcal{P}_1} \wedge x = p)$, $\phi_{f_{\mathcal{C}_1}} = (\phi_{\mathcal{P}_1} \wedge (x = p + 1))$, $\phi_{inv_{\mathcal{C}_1}} = \phi_{init_1} \vee \phi_{f_{\mathcal{C}_1}}$, $\phi_{init_{\mathcal{C}_2}} = (\phi_{\mathcal{P}_2} \wedge x = p)$, $\phi_{f_{\mathcal{C}_2}} = (\phi_{\mathcal{P}_2} \wedge (x = p - 1))$, and $\phi_{inv_{\mathcal{C}_2}} = \phi_{init_2} \vee \phi_{f_{\mathcal{C}_2}}$, where $\phi_{\mathcal{P}_1} = (0 \leq p \leq 1)$ and $\phi_{\mathcal{P}_2} = (1 \leq p \leq 2)$. Intuitively, $\mathcal{C}_1$ eventually increments the value of x by 1, while $\mathcal{C}_2$ eventually decrements it by 1. Furthermore, $\phi_{inv_{\mathcal{C}_i}} = \phi_{init_i} \vee \phi_{f_{\mathcal{C}_i}}$, for $i = 1, 2$, indicates that the set of states that are possibly visited during execution of controller $\mathcal{C}_i$ is the union of initial states and final states. Figure 4 shows the control game structure for this example where player-2 (player-1) states are depicted by boxes (ovals, respectively) and player-2 states are grouped together based on their valuations over x for a compact representation. Each node of the graph in Figure 4 is labeled with a name q_j and a set of predicates that hold in those states. For example, node q_0 represents all player-2 states in the control game structure for which $x = 1$. Nodes q_7 and q_{11} can be interpreted in a similar way. Outgoing edges from player-2 states are labeled by an instantiated controller that the composer can select at those states, e.g., at q_0, the composer can select either the instantiated controller $\mathcal{C}_{1_{\downarrow 1}}$ (corresponding to the action $(p' = 1, \gamma'_{\mathcal{C}} = 1)$,) or the instantiated controller $\mathcal{C}_{2_{\downarrow 1}}$. If the composer chooses $\mathcal{C}_{2_{\downarrow 1}}$, then the control of the system is transferred to $\mathcal{C}_{2_{\downarrow 1}}$, and q_4 is visited next in the control game structure. Note that the controller and parameter valuations are selected by the composer and they do not change in player-1 states. Once the controller is initialized, any intermediate state that satisfies the invariant of the instantiated controller can be visited in the control game structure (nodes q_5 and q_6 in Figure 4,) and at the next step, a final state of the instantiated controller is visited (represented by q_{11}), indicating the execution of the controller is over and the composer must decide the next action.*

Once the control game structure is obtained, we solve the *control game* $(\mathcal{G}^{\mathcal{C}}, \phi_{init}^{\mathcal{C}}, \Phi)$ where $\phi_{init}^{\mathcal{C}} = (t_c = 2 \wedge \phi_{init})$, i.e., the control game starts from a player-2 state so that the composer can initially select a controller and a parameter valuations. If player-2 has a winning strategy in the control game, we synthesize a winning strategy $\mathcal{S}^{\Phi}$ of special form that only depends on the valuation over variables $\mathcal{V}$ and then extract a control strategy $S^{\mathcal{C}}$ from it. Formally, let $\Upsilon = \mathcal{V}^{\mathcal{C}} \backslash \mathcal{V}$ be the set of parameters and additional variables introduced for the control game structure. We say a player-2 strategy $\mathcal{S}^{\Phi}$ in the control game structure is Υ-*independent* if there exists a partial function $f^{\mathcal{C}} : \Sigma_{\mathcal{V}} \to \Lambda^{\mathcal{C}}$ such that for any player-2 state $v^{\mathcal{C}} = (v, 2, i, j, p) \in \Sigma^2_{\mathcal{V}^{\mathcal{C}}}$, $\mathcal{S}^{\Phi}(v, 2, i, j, p) = f^{\mathcal{C}}(v)$. Intuitively, it means that $\mathcal{S}^{\Phi}$ only depends on the valuation over variables $\mathcal{V}$. The following theorem states that if player-2 can win the control game, then a Υ-independent winning strategy can be synthesized.

THEOREM 2. *If the control game is realizable, then there exists a Υ-independent winning strategy for player-2.*

The main reason is that the predicates $\tau_s^{\mathcal{C}_i}$ in the transition relation $\tau^{\mathcal{C}}$ of $\mathcal{G}^{\mathcal{C}}$ do not depend on the variables $t_e, \gamma_{\mathcal{C}}$ or parameters (though depending on their primed copies,) and the value of $t_c = 2$ is fixed. Intuitively, it means the composer can select the next controller and parameter valuations only based on the current valuation over $\mathcal{V}$ and regardless of current values of parameters, t_e and $\gamma_{\mathcal{C}}$. A control strategy $\mathcal{S}^{\mathcal{C}} : \Sigma_{\mathcal{V}} \to \Sigma_{\mathcal{P}} \times \Gamma_{\mathcal{C}}$ is extracted from $\mathcal{S}^{\Phi}$ using the function $f^{\mathcal{C}}$ as follows. For any $v \in \Sigma_{\mathcal{V}}$ such that $f^{\mathcal{C}}(v)$ is defined and $f^{\mathcal{C}}(v) = (p', i) \in \Sigma_{\mathcal{P}'} \times \Sigma_{\gamma'_{\mathcal{C}}}$, we let $\mathcal{S}^{\mathcal{C}}(v) = (p, \mathcal{C}_i)$ where p is obtained by replacing primed copies of parameters in p' by their unprimed versions. Algorithm 2 summarizes the steps for computing $\mathcal{S}^{\mathcal{C}}$. The following theorem establishes the correctness and the complexity of Algorithm 2.

THEOREM 3. *Algorithm 2 is sound. For reachability and safety objectives, it performs $O(|\Sigma_{\mathcal{V}}|)$ symbolic steps.*

Note that the upper-bound on the number of symbolic steps in Algorithm 2 is independent from the variables $\mathcal{V}^{\mathcal{C}} \backslash \mathcal{V}$. This is partly because transitions from player-2 states in the control game structure do not depend on the current valuation over variables $t_e, \gamma_{\mathcal{C}}$, and $\mathcal{P}$. Thus, if a player-2 state with the valuation $v \in \Sigma_{\mathcal{V}}$ over variables $\mathcal{V}$ is winning, then *all* player-2 states with the same valuation v over $\mathcal{V}$ are winning. Roughly speaking, the symbolic algorithm for computing the set of winning states manipulates the set of player-2 states based on their valuations over $\mathcal{V}$. Note that there are only $|\Sigma_{\mathcal{V}}|$ player-2 states with different valuations over $\mathcal{V}$ in $\mathcal{G}^{\mathcal{C}}$. Besides, any infinite run in $\mathcal{G}^{\mathcal{C}}$ starting from a player-2 state has a special form where every player-2 state is followed by two player-1 states and then a player-2 state is visited, and this pattern repeats infinitely. Due to this special form, with every three symbolic steps, either the symbolic algorithm terminates by reaching a fix point that characterize the set of winning states, or a new set of player-2 states with some valuation v over $\mathcal{V}$ are discovered to be winning (or losing) by the symbolic algorithm. Hence, the number of symbolic steps is bounded by $O(3|\Sigma_{\mathcal{V}}|) = O(|\Sigma_{\mathcal{V}}|)$.

EXAMPLE 4. *Consider the setting in Example 3. Let $\phi_{init} = (x = 0)$ be a set of initial states, and $\Phi = \Box(x \neq 2) \wedge \Diamond(x = 1)$ be the objective. A control strategy enforcing the objective Φ is shown in Figure 4 by solid edges at player-2 states. Initially, at q_{11}, the composer chooses controller $\mathcal{C}_1$ with parameter value $p = 0$. Once $\mathcal{C}_{1_{\downarrow 0}}$ reaches a final state, the control is returned to the composer, and based on the current valuation over x, $x = 1$ in this example, the next controller, $\mathcal{C}_2$, and the next parameter valuation, $p = 1$, are chosen by the composer. Intuitively, at states with $x = 0$, the composer increments x by selecting $\mathcal{C}_1$ and at states with $x = 1$, it decrements x by choosing $\mathcal{C}_2$.*

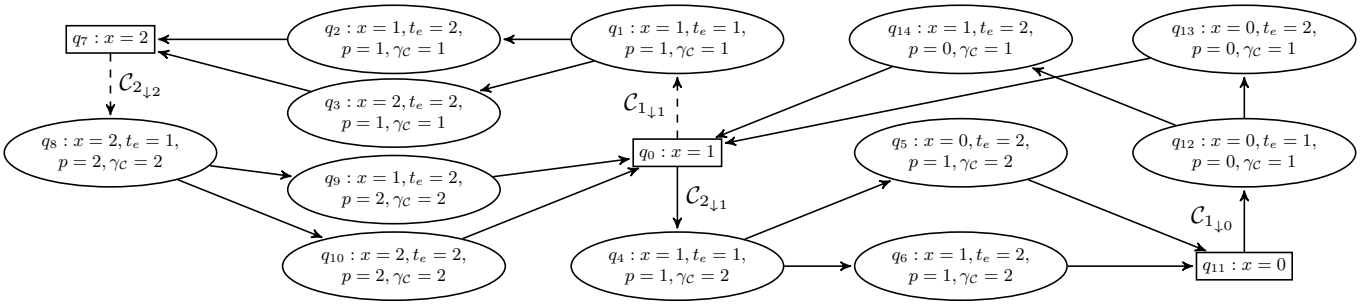

Figure 4: Control game structure for Example 3 where player-2 states are grouped together for a compact representation. Outgoing edges from player-2 states are labeled by an instantiated controller that the composer can choose at those states. A Control strategy for objective $\Phi = \Box(x \neq 2) \wedge \Diamond(x = 1)$ is to choose solid edges at player-2 states.

Algorithm 2: Control Strategy Synthesis

Input: A predicate ϕ_{init} specifying a set of initial states, a non-parametric safety and reachability objective Φ, a set $\Gamma_{\mathcal{I}_\mathcal{C}}$ of controller interfaces

Output: A control strategy $\mathcal{S}^\mathcal{C}$ s.t. its induced strategy is winning in the game $(\mathcal{G}, \phi_{init}, \Phi)$

1 Obtain the control game structure $\mathcal{G}^\mathcal{C}$ using $\Gamma_{\mathcal{I}_\mathcal{C}}$;

2 $\phi^\mathcal{C}_{init} := t_c = 2 \wedge \phi_{init}$;

3 Synthesize a $\mathcal{V}^\mathcal{C} \backslash \mathcal{V}$-independent winning strategy $\mathcal{S}^\Phi$ by solving the game $(\mathcal{G}^\mathcal{C}, \phi^\mathcal{C}_{init}, \Phi)$;

4 Extract and return a control strategy $\mathcal{S}^\mathcal{C}$ from $\mathcal{S}^\Phi$;

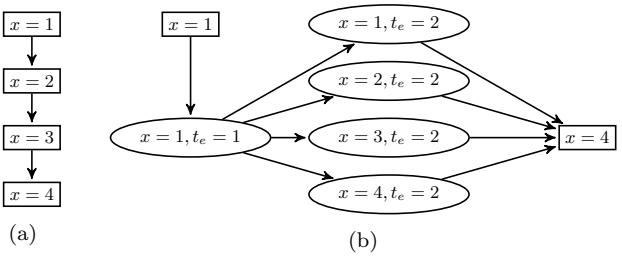

Figure 5: (a) Part of a run in a game structure where the controller takes the control at a state with $x = 1$ and increments x by 3. (b) Part of a control game structure capturing execution of the controller from a state with $x = 1$.

Completeness. Note that Algorithm 2 is not complete as the interfaces provide an abstraction of the controllers and they might lack some information on the *sequence* of states that will be visited during the execution of the controller. As an example consider a game structure $\mathcal{G}$ with an integer variable x. Consider a controller $\mathcal{C}$ that starting from a state $x = p$ where p is a parameter, increments x by three, i.e., eventually $x = p+3$. Assume that the controller does this by incrementing x one by one in three consecutive steps. Figure 5a shows a part of a run of $\mathcal{G}$ starting from a state where $x = 1$ and applying the controller $\mathcal{C}_{\downarrow p=1}$. For simplicity, we assume that all states in $\mathcal{G}$ are player-2 states (represented by boxes). The interface $\mathcal{I}_\mathcal{C}$ of the controller $\mathcal{C}$ is defined as $\mathcal{I}_\mathcal{C} = (\phi_{init_\mathcal{C}}, \phi_{inv_\mathcal{C}}, \phi_{f_\mathcal{C}})$ where $\phi_{init_\mathcal{C}} = (x = p)$, $\phi_{inv_\mathcal{C}} = \bigvee_{i=p}^{p+3} x = i$, and $\phi_{f_\mathcal{C}} = (x = p+3)$. That is, starting from a parametric state $x = p$ and using the controller $\mathcal{C}$, any state $x \in [p..p+3]$ can be visited, and eventually x is incremented by 3. Note that here we removed the constraints concerning the set of admissible parameter values from the interface to keep the example simple. Figure 5b shows part of a control game structure obtained from $\mathcal{I}_\mathcal{C}$ from any state with $x = 1$. To keep the figure simple, we removed the parameters and variables corresponding to the controllers, and similar to Example 3, player-2 states are grouped together based on their valuations over x. Let $\phi_{init} = (x = 1)$ and $\Phi = \Diamond(x = 3)$ specify the initial state and the objective of the system, respectively. It is easy to see that using controller $\mathcal{C}$ guarantees visiting the state $x = 3$ on its path to the final state $x = 4$. However, there is no control strategy over the control game structure that guarantees visiting $x = 3$ as player-1 can avoid the state with $x = 3$ in the control game structure. Intuitively, the sequence of states $(x = 1)(x = 2)(x = 3)(x = 4)$ is "lost" in the control game structure. This loss of information is the cost paid for having a simpler control game structure.

For a given objective Φ, completeness of the framework can be achieved by analyzing the controllers and enriching the interfaces, or in the extreme case by having interfaces that exactly capture the possible outcomes of applying corresponding controllers. However, our main emphasis is on simplicity and separating the two design layers, the parametric controller synthesis and synthesis from a library of parametric controllers. Controllers in our framework can be viewed as *black-boxes* where their input-output behavior is provided through their simple interfaces. An alternative view is to see the controllers as *white-boxes* and extract more information from them. The trade-off is that the former approach is simpler and more computationally efficient as the control game structure is simpler and requires less number of symbolic steps, while the latter guarantees completeness.

6. CASE STUDY

In this section we apply the methods developed in Sections 4 and 5 to an autonomous vehicle case study. Consider a network of one-way streets connected via intersections as shown in Figure 1. Assume there is a controlled autonomous vehicle V_1 initially positioned in the grid marked with s_0. Also assume there is an uncontrolled vehicle V_2 initially at the grid-cell s^*. Each vehicle has actions move-forward, back-up, turn-left, turn-right, and stop that moves it one step forward, backward, to the left, to the right, and leaves its position unchanged, respectively. The goal is to synthesize a controller for V_1 that can guide it from the initial position

s to the final destination d while obeying the traffic laws (e.g., moving in the specified directions of streets, no stopping inside an intersection, etc.) and avoiding collision with V_2 and static obstacles. We assume that the uncontrolled vehicle respects traffic laws by always moving in the specified directions for the streets. We implemented our algorithms symbolically in Java and using BDD package JDD [17]. We first specify and synthesize a set of parametric reactive controllers that guarantee advancing the vehicle in north, south, west, and east directions while avoiding collision with static obstacles and the other vehicle. We then synthesize a control strategy that instantiates and composes these controllers to navigate V_1 safely from initial position to the destination.

Synthesis of Parametric Controllers. We denote the location of the vehicle V_i at any time-step with (x_i, y_i) for $i = 1, 2$. We specify four parametric controllers that can move the car in different directions: Controller $\mathcal{C}_1$ ($\mathcal{C}_2$) with specified controller interface $\mathcal{I}_1$ ($\mathcal{I}_2$) that moves the car three steps toward east (west, respectively), and controller $\mathcal{C}_3$ ($\mathcal{C}_4$) with specified controller interface $\mathcal{I}_3$ ($\mathcal{I}_4$) that advance the car one step toward north (south, respectively.) More specifically, let a, b be two parameters. Let $\phi_{init} = (x_1 = a \wedge y_1 = b)$ be the parametric initial state. Let $\phi_{inv} = (x_1 \neq x_2 \vee y_1 \neq y_2)$, i.e., no collision between vehicles. Finally, let $\phi_{f_1} = (x = a + 3, y = b)$, $\phi_{f_2} = (x = a - 3, y = b)$, $\phi_{f_3} = (x = a, y = b + 1)$, and $\phi_{f_4} = (x = a, y = b - 1)$ specify moving in different directions. Controllers are specified by interfaces $\mathcal{I}_{\mathcal{C}_i} = (\phi_{init}, \phi_{inv}, \phi_{f_i})$. Algorithm 1 is then used to synthesize parametric controllers, their corresponding interfaces and the set of admissible parameter values.

Synthesis of Control Strategy. Once a library of parametric reactive controllers $\Gamma_{\mathcal{C}}$ and their corresponding interfaces $\Gamma_{\mathcal{I}_{\mathcal{C}}}$ are formed, Algorithm 2 is used to synthesize a control strategy for the controlled vehicle. A synthesized control strategy instantiates and applies the controller $\mathcal{C}_1$ consecutively to advance the vehicle toward east and finally bring it to the position marked by s_6, from which controller $\mathcal{C}_3$ is instantiated and employed consecutively to take the vehicle to its destination. More specifically, at any position marked by s_i for $i = 0, \cdots, 5$, controller $\mathcal{C}_1$ is instantiated and becomes active, and it guarantees to eventually advance the controlled vehicle to the next position s_{i+1}. Similarly, at any position marked by s_i for $i = 6, \cdots, 9$, controller $\mathcal{C}_3$ is instantiated and becomes active, and it eventually navigates the controlled vehicle to the next position s_{i+1} where $s_{10} = d$ is the final destination. The control strategy sets the parameters a and b according to the current state of the vehicle, i.e., if the current position of V_1 is $(x_1 = i, y_1 = j)$, the control strategy instantiates the controllers $\mathcal{C}_1$ and $\mathcal{C}_3$ by parameter valuation $(a, b) = (i, j)$.

7. CONCLUSION AND FUTURE WORK

We presented a framework for symbolic synthesis from a library of parametric and reactive controllers. We also showed how these controllers can be synthesized from parametric objectives specified by the user. In this paper, we assumed that the controllers have perfect information about the state of the system at any time-step. However, in practice, this assumption might be unrealistic, e.g., due to the imperfection and limitations of the sensors of the system. In future, we plan to investigate how our approach can be generalized to synthesize strategies for systems from a library of controllers with *partial* information.

Acknowledgement

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

8. REFERENCES

[1] R. Alur, S. Moarref, and U. Topcu. Pattern-based refinement of assume-guarantee specifications in reactive synthesis. *TACAS*, 2015.

[2] C. Baier, J. Klein, and S. Klüppelholz. A compositional framework for controller synthesis. In *Concurrency Theory*. 2011.

[3] R. Bloem, H. N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in n log n symbolic steps. *FMSD*, 2006.

[4] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 2012.

[5] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[6] E. Filiot, N. Jin, and J.-F. Raskin. Antichains and compositional algorithms for LTL synthesis. *FMSD*, 2011.

[7] E. Frazzoli, M. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics*, 2005.

[8] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 2009.

[9] H. Kress-gazit, T. Wongpiromsarn, and U. Topcu. Correct, reactive robot control from abstraction and temporal logic specifications.

[10] O. Kupferman, N. Piterman, and M. Vardi. Safraless compositional synthesis. In *CAV 2006*, 2006.

[11] J. Liu, N. Ozay, U. Topcu, and R. M. Murray. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Transactions on Automatic Control*, 58(7), 2013.

[12] Y. Lustig and M. Y. Vardi. Synthesis from component libraries. In *Foundations of Software Science and Computational Structures*. Springer, 2009.

[13] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Symposium on Theoretical Aspects of Computer Science*, pages 229–242, 1995.

[14] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *IROS*, 2014.

[15] S. Sohail and F. Somenzi. Safety first: A two-stage algorithm for LTL games. In *FMCAD*, 2009.

[16] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[17] A. Vahidi. Jdd. http://javaddlib.sourceforge.net/jdd/index.html.

[18] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning. *AC*, 2012.