Synthesizing Cooperative Reactive Mission Plans

Rüdiger Ehlers*, Robert Könighofer*, and Roderick Bloem*

Abstract— **By performing synthesis from formal high-level mission specifications, we can obtain robot controllers that are guaranteed to operate correctly under the specified environment conditions. Such conditions must be stated in the specification whenever there is no way in which the robot's task can be fulfilled without them holding, and they relate the possible behaviors of the environment with the behavior of the robot. Contemporary synthesis algorithms however frequently construct implementations that try to trivially satisfy their specifications by actively working towards the violation of the assumptions, which is undesirable behavior.**

To solve this problem, we consider the synthesis of *cooperative* implementations of high-level robot controllers. In addition to being correct, these have executions from every of their states on which the environment satisfies its assumptions. Cooperative implementations are particularly helpful in scenarios in which the behavior of humans or other robots is modeled by environment assumptions, as such implementations help them with meeting their own objectives.

We show how the generalized reactivity(1) synthesis approach, which was shown to be valuable for robotics applications many times, can be adapted to yield such implementations, and demonstrate the suitability of the resulting synthesis algorithm on a complex delivery scenario. Our results show that synthesizing cooperative implementations has the same complexity as the standard reactive synthesis problem and is a tractable problem in practice.

I. Introduction

A particularly ambitious approach to the construction of robot controllers that execute reactive high-level mission plans is to automatically synthesize them from formal specifications [1], [2], [3], [4]. Whenever a mission plan is computed by the synthesis procedure, it is correct by construction. This means that whenever the environment fulfills the *assumptions* that are stated about it in the specification and given the correct operation of the actuators and sensors, executing the plan leads to the satisfaction of the system *guarantees* that are stated in the specification.

While task planning is a classical topic in robotics, many recent works consider *reactive* missions plans, which are represented by *finite-state machines* and contain executions for all possible environment behaviors. Thus, they are able to react to changes in the environment without the need to re-plan. The applicability of synthesis for reactive mission plans has been demonstrated for a plethora of domains, such as the coordination of multiple robots working on a single task [4], and robots operating under piece-wise affine dynamics [5].

When synthesizing a reactive mission plan, the assumptions in a specification play an important role, as they define the conditions under which a robot has to perform its mission [1], [2], [3], [4]. For example, if the workspace of a robot contains doors, assumptions need to be made about when doors are open. Without such assumptions, a specification is only *realizable* if no behavior of the environment can prevent the robot from fulfilling its mission. In addition to *liveness assumptions* (such as "the door is always eventually open"), *safety assumptions* are also frequently part of the specification as they describe environment behavior that the robot can assume not to occur (e.g., "the door never opens when the robot is standing near the inside of the door").

The assumptions and guarantees in the specification are connected by an *implication*: the guarantees only have to be met if the environment satisfies the assumptions that we made about it. As the guarantees can typically only be satisfied if the assumptions are, this is the right way of connecting assumptions and guarantees. Unfortunately, connecting assumptions and guarantees by an implication leads to the controller having an incentive to actively work against the satisfaction of the assumptions. In our door example, this could happen by the robot standing still indefinitely on the inside of the door while it is closed: the door is then not allowed to open anymore, while at the same time the assumptions also state that it is open infinitely often. Since the assumptions are not fulfilled in this case, this releases the robot from the obligation to fulfill its mission. Obviously, such robot behavior is of little use in practice. Yet, it fulfills the requirement that the mission is carried out whenever the environment assumptions hold.

Changing the assumptions to allow the door to stay closed while the robot is standing on its inside would eliminate the problem. However, adding exceptions to the specification for all cases in which the robot can exploit the environment would defeat the purpose of synthesis, as such cases can be numerous and hard to find. Also, requiring the robot to fulfill its mission regardless of whether the environment assumptions hold is not feasible either, as the assumptions have been made for a reason. In our example from above, if the door connects two rooms which the robot needs to patrol through, the assumption that the door between the rooms is always eventually open is necessary for a mission plan to exist.

The observation that automatically synthesized reactive high-level mission plans can be *uncooperative* leads directly to the question if there is some way to avoid changing the specification and to synthesize a reactive mission plan that is *cooperative* with its environment, i.e., performs its mission if the environment satisfies its assumptions while refraining from working towards the violation of the assumptions.

*Rüdiger Ehlers is affiliated with the University of Bremen and the DFKI GmbH, Bremen, Germany. The other two authors are with Graz University of Technology, Austria. Emails: rehlers@uni-bremen.de, {robert.koenighofer, roderick.bloem}@iaik.tugraz.at.

In this paper, we give a positive answer to this question. We start with the generalized reactivity(1) synthesis algorithm [3], which has been proven to be particularly well-suited for robotics applications [1], [4], [5]. It is commonly known under its abbreviated name *GR(1) synthesis* and is attractive due to its comparably low singly-exponential time complexity, which is in contrast to synthesis from richer logics, such as full *linear temporal logic* (LTL), which requires doubly-exponential time for synthesis [6]. It is easy to show that controllers computed with this approach are not necessarily cooperative by itself, and we give an example in Section II.

We present a modification of the GR(1) synthesis approach to produce *cooperative* reactive mission plans whenever possible. These are not only correct in the traditional sense (i.e., fulfill the specified guarantees if the assumptions are satisfied), but at any point in time offer an execution that leads to the satisfaction of both assumptions and guarantees. Our modification does not increase the computational complexity of synthesis: it stays singly-exponential in the number of propositions used in the assumptions and the guarantees.

Our cooperative synthesis algorithm is a direct replacement for the original GR(1) synthesis algorithm, which makes it useful in contexts in which GR(1) synthesis has previously been applied. Of particular interest are applications in which a robot and a human coordinate. Here, the specifications contain assumptions about the behavior of the human as well as system guarantees that the robot should fulfill. Surely, the ultimate goal in the setting is to let both the human and the robot meet their specified goals, which makes our cooperative strategies especially useful in this context.

A. Related Work

Cooperative robot controllers that allow a group of robots to perform its joint task in collaboration are a classical topic in robotics (see, e.g., [7], [8]). The focus in this context lies on computing a joint strategy for the robots, which they never deviate from. Another research direction is the cooperation with humans or other robots in a multi-agent setting. Here, the agents and humans have their own individual goals and coordinate on achieving them and/or some joint goal. Some recent works in this area use formal methods in order to derive correct-by-construction controllers [9].

When robot controllers must react to the environment, soundness and completeness of controller generation can be assured by performing *reactive synthesis* [3], [1]. In this context, assumptions about the environment have to be stated, and the controller can assume to work in an environment that satisfies the imposed constraints. In previous work, the complex interaction with the environment was typically not considered, as the assumptions were typically simple, such as "some door has to be open infinitely often". Thus, the cooperation of the system with the environment, which can include humans or other robots, was not explicitly considered.

A notable exception is the work on synthesizing robot controllers that work under certain dynamics by DeCastro et al. [10]. They compute specification revisions that require the environment not to perform adversarial actions whenever due to the robot dynamics, the robot cannot avoid to violate its guarantees if the environment performs these actions. A typical example is a robot with inertia that is moving towards a door: if it is quite close to the door already, the environment must not be allowed to close it at that point in order for the mission specification to be fulfillable.

Reactive synthesis has also been applied in multi-robot settings. In particular, Weng Wong and Kress-Gazit [11] present an approach to let a group of robots coordinate their individual tasks by exchanging information about their tasks. They then use this information as assumptions about the behavior of the other robots in order to find a controller for their own objectives.

This work is part of a larger endeavor to synthesize controllers that *cooperate*, *are not lazy*, and *never give up*. All these problems are outlined in [12]. A procedure for synthesizing *cooperative implementations* from general linear temporal logic (LTL) specifications is given in [13]. In that work, we consider multiple reasonable *levels of cooperation* between the environment and the system to be constructed, and give a synthesis procedure with a doubly-exponential time complexity that is suitable for all considered levels of cooperation. The high computational complexity of the general case makes the procedure from [13] unsuited for robotics applications, which motivated the development of a specialized procedure for GR(1) specifications and the type of cooperation between the environment and the system that we consider in this paper. In addition, the best-effort quality guarantees that GR(1) synthesis offers (e.g., that the synthesized implementations contain no unnecessary loops while moving towards the next goal) are retained in the approach in this paper. Furthermore, our specialized synthesis procedure for GR(1) specifications works without the tree automata constructions of [13], which improves the accessibility of the theory behind cooperative synthesis for researchers from the robotics domain.

II. Motivating Example

Before we give the formal preliminaries and define the problem of synthesizing a cooperative reactive mission plan, let us motivate this problem using an example.

Consider the workspace depicted in Figure 1. We want to synthesize a reactive mission plan (which is also called a *controller* in the formal methods literature) for a robot whose task is to patrol between the two green cells. It must not collide with a moving obstacle, which may be a human or another robot. Neither the moving obstacle nor the robot may collide with walls, and they can move up, down, left, or right in every step. Diagonal motion is permitted for both the robot and the moving obstacle. The robot is not allowed to move towards the obstacle if they are located adjacently, and also the obstacle may not move towards the robot when they are located adjacently. With these constraints, collisions between the robot to be controlled and the moving obstacle are not possible. As the robot and the obstacle can block each

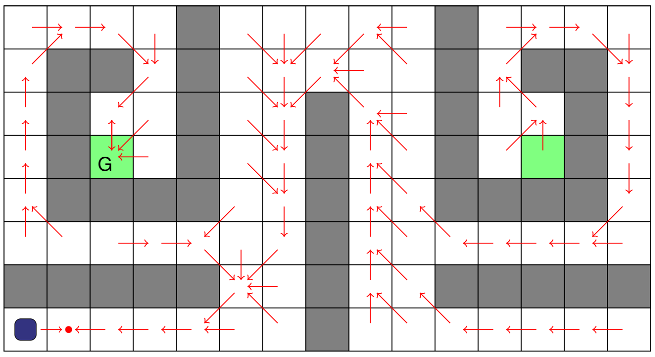

Fig. 1. An example workspace, where the robot to be controlled is not shown. The arrows represent the vector field of discrete motion actions that a robot with a non-cooperative high-level controller performs when the region marked with a G is its current goal while the moving obstacle (which is shown as a rounded rectangle) is standing still in the lower left corner.

other's paths, we add the assumption on the environment that the moving obstacle has to eventually move away if the robot and the obstacle are located in adjacent cells for a long period of time. Only with this assumption, the specification built from the scenario becomes *realizable* (i.e., there exists a controller) and we can synthesize a corresponding reactive mission plan.

The resulting robot controller is not cooperative with the environment. Figure 1 shows the possible trajectories of the robot if the moving obstacle is in the lower left corner and the robot's current goal is the region marked by a G. It can be seen that in some cases, instead of moving to this goal, the robot actually works towards blocking the obstacle's motion. In particular, the robot makes this choice whenever it is not already very close to the goal. Once the obstacle is cornered, it cannot fulfill the liveness assumption to always eventually move away any more. By the way in which the assumptions and guarantees are connected in a specification, this is allowed behavior despite the fact that the goal is never reached. However, this behavior makes little sense.

A cooperative implementation would only be allowed to corner the obstacle temporarily. Thus, it would eventually allow the obstacle to leave the lower left corner.

III. Preliminaries

a) Propositions, traces, and controllers: For all sets $\mathcal{A}$ and $\mathcal{B}$ and all $(x, y) \in 2^{\mathcal{A}} \times 2^{\mathcal{B}}$, we denote by $(x, y)|_C$ for $C \in \{\mathcal{A}, \mathcal{B}\}$ the mapping of (x, y) to x or y, respectively. Given sets $\mathcal{X}$ and $\mathcal{Y}$ of input and output propositions, a controller over $\mathcal{X}$ and $\mathcal{Y}$ is a finite-state machine $\mathcal{M} = (S, \mathcal{X}, \mathcal{Y}, \delta, s_0)$ with the set of controller states S, the partial transition function $\delta : S \times 2^{\mathcal{X}} \rightharpoonup S \times 2^{\mathcal{Y}}$, and the initial state $s_0 \in S$. Executions of the controller are called *traces* and defined as words $w = w_0 w_1 w_2 \ldots \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ such that there exists a *run* $\pi = \pi_0 \pi_1 \ldots \in S^\omega$ with $\pi_0 = s_0$ and $\delta(\pi_i, w_i|_{\mathcal{X}}) = (\pi_{i+1}, w_i|_{\mathcal{Y}})$ for all $i \in \mathbb{N}$. Traces can also be finite. In this case, π is a finite string of length $i + 1$ for some $i \in \mathbb{N}$, where $\delta(\pi_i, w_i|_{\mathcal{X}})$ is undefined while for all $j < i$, $\delta(\pi_j, w_j|_{\mathcal{X}})$ is defined.

b) Specifications: Intuitively, specifications describe a set $T \subseteq (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ of acceptable traces for a controller. We use propositional logic augmented with the unary temporal operators $\Box$ ("globally"), $\Diamond$ ("finally"), and $\bigcirc$ ("next") as a concise representation of specifications. All these temporal operators are borrowed from linear temporal logic (LTL) [14]. A trace $w = w_0 w_1 w_2 \ldots \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ can satisfy a temporal logic specification φ, written as $w \models \varphi$, or not. We define that $w \models p$ for some $p \in \mathcal{X} \cup \mathcal{Y}$ if $p \in w_0$. Boolean connectives have the usual semantics. The temporal operators allow sub-formulas to observe the elements of the trace after the first one. We write that $w \models \bigcirc\varphi$ for some sub-formula φ if $w_1 w_2 w_3 \ldots \models \varphi$. Likewise, we have that $w \models \Diamond\varphi$ if there exists some $i \in \mathbb{N}$ such that $w_i w_{i+1} w_{i+2} \ldots \models \varphi$, and declare that $w \models \Box\varphi$ if for all $i \in \mathbb{N}$, we have $w_i w_{i+1} w_{i+2} \ldots \models \varphi$.

All specifications considered in this paper belong to the *generalized reactivity(1) class*, which is also abbreviated as *GR(1)*. Such specifications are of the form

$$(\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a) \rightarrow_s (\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g).$$

All *specification parts* φ_i^a, φ_s^a, φ_l^a, φ_i^g, φ_s^g, and φ_l^g are conjunctions of sub-formulas. The parts φ_i^a and φ_i^g are the *initialization assumptions* and *initialization guarantees*. These are only concerned with variables in $\mathcal{X}$ and $\mathcal{Y}$, respectively, and free of temporal operators. The parts φ_s^a and φ_s^g are the *safety assumptions* and *safety guarantees*, in which the only temporal operator that may occur is $\bigcirc$, and it may not be nested. Finally, the parts φ_l^a and φ_l^g are *liveness assumptions* and *liveness guarantees*. All conjuncts in these parts are of the form $\Box\Diamond\psi$, where the only temporal operator that can occur in ψ is $\bigcirc$, which may not be nested. We also say that ψ is an *environment goal* if $\Box\Diamond\psi$ is a liveness assumption, or a *system goal* if $\Box\Diamond\psi$ is a liveness guarantee. The $\rightarrow_s$ operator that separates the assumptions and guarantees in a GR(1) specification represents *strict implication* [3], which for the specification to be satisfied requires that in addition to the guarantees holding if all assumptions hold along a trace, no safety guarantee may be violated *before* a safety assumption is violated.

c) GR(1) games: The synthesis problem for a set of input propositions $\mathcal{X}$, a set of output propositions $\mathcal{Y}$, and a specification φ is to decide whether there exists a controller reading $\mathcal{X}$ and writing to $\mathcal{Y}$ all of whose traces satisfy φ. A specification is said to be *realizable* if there exists such a controller. Determining the existence of such a controller is often performed by solving a *game*. Generalized reactivity specifications in particular are translated to *GR(1) games* [3]. Given proposition sets $\mathcal{X}$ and $\mathcal{Y}$, these are defined as tuples $\mathcal{G} = (\mathcal{V}, E_0, E_1, v_0, \{\psi_1^a, \ldots, \psi_m^a\}, \{\psi_1^g, \ldots, \psi_n^g\})$, where $\mathcal{V} = 2^{\mathcal{X}} \times 2^{\mathcal{Y}}$ defines the set of *positions* in the game, $E_0 \subseteq \mathcal{V} \times 2^{\mathcal{X}}$ defines the transition relation for the *environment player*, and $E_1 \subseteq \mathcal{V} \times \mathcal{V}$ defines the transition relation for the *system player*. The elements of $\{\psi_1^a, \ldots, \psi_m^a\}$ and $\{\psi_1^g, \ldots, \psi_n^g\}$ are the environment player's and system player's goals, each of which are subsets of $\mathcal{V} \times \mathcal{V}$. To simplify the presentation in the following, we assume that the GR(1) specification from which $\mathcal{G}$ is built has a single $(x, y) \in 2^{\mathcal{X}} \times 2^{\mathcal{Y}}$ such that $(x, y) \models \varphi_i^a \wedge \varphi_i^g$, as this allows us to define that v_0 represents the singleton initial position of the game.

In the game, the two players interact by successively constructing a *play* $\pi = \pi_0\pi_1 \ldots \in \mathcal{V}^\omega \cup \mathcal{V}^*$. Starting from $\pi_0 = v_0$, the environment and system successively select values $x_i \subseteq \mathcal{X}$ and $y_i \subseteq \mathcal{Y}$, which together compose $\pi_i = x_i \cup y_i$. The environment player is only allowed to select values x_i such that $(\pi_{i-1}, x_i) \in E_0$ and the system player may only select values y_i such that $(\pi_{i-1}, (x_i, y_i)) \in E_1$. If one of the players does not have a valid next move, it loses the play, which is then finite. All other plays are infinite, and winning for the system player if either

- for all ψ_j^g for $(1 \leq j \leq n)$, we have that for infinitely many $i \in \mathbb{N}$, $(\pi_i, \pi_{i+1}) \in \psi_j^g$, or
- for some ψ_j^a for $(1 \leq j \leq m)$, we have that only for finitely many $i \in \mathbb{N}$, $(\pi_i, \pi_{i+1}) \in \psi_j^a$.

We say that the system player wins the game if it has a strategy to always choose the respective next move such that any resulting play is winning. Such a strategy can be represented by a finite-state machine $\mathcal{M} = (S, \mathcal{X}, \mathcal{Y}, \delta, s_0)$, which in turn is an implementation for the specification from which the GR(1) game is built. We say that a play π is in a *deadlock* if π is finite, and for π_i being the last element of π, there does not exists an $x \in \mathcal{X}$ such that $(\pi_i, x) \in E_0$. Furthermore, we say that an infinite play $\pi = \pi_0\pi_1 \ldots$ is in a *livelock* at position $i \in \mathbb{N}$ if for some environment or system goal ψ, for all $j > i$, we have $(\pi_j, \pi_{j+1}) \notin \psi$.

We will discuss in Section V how to *solve a GR(1) game* (i.e., how to determine whether the system has a winning strategy from v_0). For the description of game solving, we need to define a couple of operations over position and transition sets in games. Let $f : 2^{\mathcal{V}} \to 2^{\mathcal{V}}$ be a monotone function. We define $\mu X.f(X)$ to represent the value of evaluating $f(f(f(\ldots f(\emptyset))))$, i.e., iteratively applying f starting from $\emptyset$ until a fixed point (the *least fixed point*) of f is reached. Likewise, $\nu X.f(X)$ represents the value of evaluating $f(f(f(\ldots f(\mathcal{V}))))$, i.e., the *greatest fixed point* of f. Fixed points can also be nested, so that, e.g., for a function $f : 2^{\mathcal{V}} \times 2^{\mathcal{V}} \to \mathcal{V}$, $\mu Y.\nu X.f(X, Y)$ represents the least fixed point of function $f_Y(X)$, where for fixed Y, $f_Y(X)$ is defined to be the greatest fixed point of $f(X, Y)$ with respect to X. To reason about from which positions in the game, the system player can enforce that certain transitions are taken, we use the $\mathsf{EnfPre} : 2^{\mathcal{V} \times \mathcal{V}} \to 2^{\mathcal{V}}$ operator, defined as

$$\begin{aligned}\mathsf{EnfPre}(T) = \{v \in \mathcal{V} \mid \forall x \subseteq \mathcal{X} :\\ (v, x) \in E_0 \to \exists y \subseteq \mathcal{Y} : \big(v, (x, y)\big) \in E_1 \cap T\}\end{aligned}$$

By its definition, the operator takes into consideration that the environment and system players are only allowed to take transitions in E_0 and E_1, respectively.

In the scope of the EnfPre operator, occurrences of some position set X are interpreted as the set of transitions originating from a position in X, whereas X' represents the set of transitions leading to a position in X. So for example, the formula $\mu X.\mathsf{EnfPre}(\psi_3^a \vee X \vee X')$ defines the least fixed point of positions X from which the system can enforce that either the next transition originates from a position in X, leads to a position in X, or satisfies environment goal number 3. Note that in the formula, we used the $\vee$ operator instead of $\cup$ even though ψ_3^a, X, and X' are sets and not boolean functions. We did so to remain consistent with the existing literature, where it is customary to use $\vee$ and $\wedge$ instead of $\cup$ and $\cap$ in the scope of μ and ν operators, using the duality between functions of the type $\mathcal{A} \to \mathbb{B}$ and subsets of $\mathcal{A}$ (for some set $\mathcal{A}$).

d) GR(1) synthesis for robotics: Automatically synthesized controllers are guaranteed to be able to respond to all possible environment behaviors. Thus, they are well-suited to be used in applications in which a robot must continuously react correctly to the behavior of its environment. This is especially beneficial in settings in which the workspace is shared with humans or other robots, as it makes little sense to plan a fixed trajectory in such a case. The state changes of the humans and other robots would require permanent re-planning. The upfront computation of a correct-by-construction controller is thus clearly more attractive.

In order to perform synthesis, the setting must be encoded into the specification. We give the system a set of atomic propositions for controlling the region of the workspace in which the robot is and encode the possible transitions between these regions into φ_s^g. Using a continuous motion planner in parallel to the finite-state machine then allows us to execute the controller on a physical robot [1].

In the example in Section II, we partitioned a two-dimensional workspace into 15x8 regular squares and allow the robot to move left, right, up, or down in the grid in every step of its execution. The robot has $\lceil \log_2(15) \rceil + \lceil \log_2(8) \rceil = 7$ propositions to encode its position, and so does the environment for the location of the moving obstacle. While the dynamics in the example are trivial, they allow us to show the benefits of our approach without cluttering the presentation with unnecessary details about the physical domain. For this purpose, the model is well-established in robotics (see, e.g., [15], [2]).

IV. Problem Statement

Let us now define the *cooperative* controller synthesis problem.

Definition 1: Let $\mathcal{X}$ and $\mathcal{Y}$ be sets of propositions and $\varphi = (\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a) \to_s (\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g)$ be a generalized reactivity(1) specification. The *cooperative controller synthesis* problem is to find a controller reading $\mathcal{X}$, writing to $\mathcal{Y}$, all of whose executions satisfy φ, and for which for all traces $w = w_0w_1 \ldots$ of the controller and all $i \in \mathbb{N}$ (such that w is longer than i elements), there exists some (other) infinite trace $w' = w'_0w'_1 \ldots$ of the controller such that $w' \models (\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a)$ and $w_0w_1 \ldots w_i = w'_0w'_1 \ldots w'_i$.

Informally, this definition requires the controller to only have executions such that at any point in time, the environment has some way to enforce that the execution can be completed to one that satisfies the assumptions. Since the controllers that we consider are deterministic, the execution can be enforced by the environment by simply choosing the input of trace w' in the future. As the controller is required to fulfill the guarantees if the assumptions are satisfied, fulfillment of the assumptions means that the guarantees are also satisfied along this trace. We also say that a controller can *actively work against the satisfaction of the assumptions* if it is not cooperative. As we solve the cooperative controller synthesis problem on the level of GR(1) games, we also use the following problem formulation over strategies in games:

Definition 2: Let $\mathcal{G} = (\mathcal{V}, E_0, E_1, v_0, \{\psi_1^a, \ldots, \psi_m^a\}, \{\psi_1^g, \ldots, \psi_n^g\})$ be a GR(1) game. We say that there exists a cooperative strategy to win $\mathcal{G}$ from some position $v \in \mathcal{V}$ if every prefix of a play induced by the strategy can be extended to some play on which all environment and system goals are reached on infinitely many transitions in the play.

Due to the fact that GR(1) games encode the synthesis problem for GR(1) specifications, a cooperative strategy to win $\mathcal{G}$ from v_0 is precisely a solution to the problem from Definition 1 for the specification from which $\mathcal{G}$ was built.

V. Synthesizing Cooperative Implementations

Given a generalized reactivity(1) game $\mathcal{G} = (\mathcal{V}, E_0, E_1, v_0, \{\psi_1^a, \ldots, \psi_m^a\}, \{\psi_1^g, \ldots, \psi_n^g\})$, the set of positions from which player 1 (the *system player* or *output player*) wins (in a not necessarily cooperative manner) can be determined by evaluating the following fixed point formula:

$$W = \nu Z. \bigwedge_{j \in \{1,\ldots,n\}} \mu Y. \bigvee_{i \in \{1,\ldots,m\}} \nu X. \\ \mathsf{EnfPre}\big((Z' \wedge \psi_j^g) \vee Y' \vee (\neg\psi_i^a \wedge X')\big) \quad (1)$$

The set W can be thought of as being computed step-wise by the fixed point operators. Recall that the winning condition of the game is that the system wins if either some liveness assumption is violated, or all liveness guarantees are met. The outermost least fixed point νZ successively approximates the set of winning positions from above by continuously filtering out positions from which the environment has a strategy to avoid a system goal transition to ever be taken while meeting its liveness assumptions. The conjunction over the guarantee indexes reflects the fact that no liveness guarantee must be left out, so the set of winning positions needs to be winning with respect to all of them. The least fixed point μY reflects the aim of the system to get closer to the system goal; the earlier a position is added to Y when evaluating $\mu Y.(\ldots)$, the closer it is to the system goal. The disjunction over the liveness assumptions is rooted in the fact that the system wins if *any* of the liveness assumptions is not satisfied. The νX greatest fixed point then represents the search for *waiting* positions in the game, i.e., a set of positions in which the strategy can wait for the environment goal to be met. When this goal is met, the strategy has to get closer to the goal (or at least move to waiting positions that are closer to the goal).

The EnfPre operator implements the game aspect – at every step in the play, the system needs to be able to pick a move such that the resulting transition in the game either

1) meets the goal ψ_j^g and leads to a position from which the game is still winning for the system player,
2) gets the play closer to the current goal ψ_j^g, or
3) constitutes allowed waiting for some environment goal ψ_i^a while staying in some "waiting zone" X.

These three cases are precisely the ones given as disjuncts to the EnfPre operator. To obtain a strategy for a GR(1) game (and hence, an implementation for the GR(1) specification from which we built the GR(1) game), it suffices to always pick transitions that are passed to EnfPre as early as possible in the computation of W while the greatest fixed point operators are fully evaluated. This ensures that transitions that get the system closer to the system goal receive priority, and thus prevents livelocks in the generated strategy. The strategy cycles through the system goals in a round-robin fashion and whenever its respective current goal has been reached, it switches to the next goal.

Equation 1 differs from the one originally presented in [3]. In particular, the equation is a lot simpler than the older formulation by only having a single application of the EnfPre operator in the formula and also not requiring multi-variable fixed points. On a semantic level, both the simplified and the original versions are the same, except that the one presented in Equation 1 can be used for specifications in which environment and/or system goals contain a "next" ($\bigcirc$) operator, which is useful in some cases [4]. The simplified version is also the one on which the GR(1) synthesizer `slugs` [16] is based.

Contribution: The reason why standard GR(1) synthesis may compute uncooperative strategies is two-fold. First of all, if in a position $v \in \mathcal{V}$, there is no valid next move for the environment player (according to the safety assumptions φ_s^a), then the position is automatically winning. Thus, it is added to Y in the first round of evaluating the μY fixed point of Equation 1, and hence may be found *earlier* than other transitions that actually lead closer to the system goal. As a result, the strategies found sometimes prefer the transition to a deadlock position over other transitions.

The second reason is that during the computation of X, there is no distinction between the system being able to enforce certain environment goals ψ_i^a to never be reached or waiting for an environment goal that can actually be reached (e.g., waiting for a door to open while the robot is not blocking it). Despite the fact that the waiting positions may have outgoing transitions that lead closer to the goal, the strategy extraction algorithm has no priority to take them.

While we could modify strategy extraction to work in a different manner to solve both problems, such a modification is actually an unnecessarily difficult route to take; the idea to pick transitions found early with priority is at the heart of the correctness argument of the strategy extraction routine, and thus is difficult to replace. The alternative that we propose in this paper is to prevent that the undesired transitions are ever computed in the evaluation of Equation 1. We do so in two steps: (a) by eliminating deadlocks, and (b) by eliminating livelocks.

For eliminating deadlocks, it suffices to prohibit the system from making a choice of next move that leads to a position in which the environment has to violate its assumptions to make its own next move. We can do so by modifying the set of system player transitions in $\mathcal{G}$ to:

$$\tilde{E}_1 = E_1 \cap \{(v, v') \in \mathcal{V}^2 \mid \exists x \subseteq \mathcal{X}.(v', x) \in E_0\} \tag{2}$$

This modification rules out precisely the unwanted transitions and is essentially the same as the operation used by DeCastro et al. [10] to generate revisions to an unrealizable specification. For cooperative synthesis, we however do not base the modification on a counter-strategy, but rather on the moves that the system player *could* make.

Eliminating livelocks is more intricate. The strategy that we compute should, for every of its states, always allow the environment to satisfy its liveness assumptions on a run that starts in the state. However, in Equation 1, the computation only looks at one liveness assumption at a time and derives a state set X from which the system can ensure that either (1) if X is left, then a system goal has been reached or the play comes closer to the system goal, or (2) the play stays in X and the environment goal is not reached. There is no restriction that a transition *out* of the waiting phase in X has to exist, which allows the system to prevent the environment from reaching its goal. We thus modify Equation 1 as follows:

$$\begin{aligned}\tilde{W} = \nu Z. \bigwedge_{j \in \{1,\ldots,n\}} \mu Y. \bigvee_{i \in \{1,\ldots,m\}} \nu X. \\ \mathsf{EnfPre}\big((Z' \wedge \psi^g_j) \vee Y' \vee (\neg\psi^a_i \wedge X')\big) \\ \wedge\, \mu R.\mathsf{Reach}\big((\psi^g_j \vee Y' \vee R') \wedge X\big) \\ \wedge \bigwedge_{k \in \{1,\ldots,m\}} \mu R.\mathsf{Reach}\big((\psi^a_k \vee R') \wedge Z\big)\end{aligned} \tag{3}$$

In this equation, Reach is the one-player variant of EnfPre, i.e., for some set of transitions T, $\mathsf{Reach}(T)$ computes the set of positions that have an outgoing transition in T. The first two lines of Equation 3 are an exact copy of Equation 1. The third line requires every position in X to have an outgoing sequence of transitions that ends with a transition that is a system goal or leads closer to the system goal. Essentially, adding this part to the formula removes all positions from X that are only in X because the system has a strategy to prevent progress towards reaching the environment goal. The new conjunct in the third line of Equation 3 alone does not allow the environment to reach its goals, however – if, for example, some liveness assumption is not needed to fulfill the mission, then the system may still choose its actions in a way that prevents this assumption from being satisfied. To mitigate this problem, the fourth line has been added to Equation 3: it requires that from all winning positions of the game, for all environment goals, there is always a path to the goal without leaving the winning positions.

During the computation of the strategy in the game, we have to make sure that the transitions found during the

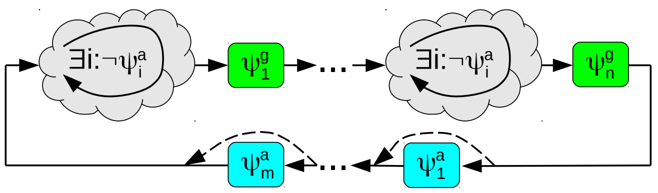

Fig. 2. Illustration of the cooperative strategy construction. Besides cycling through the system goals $\psi^g_1 \ldots \psi^g_n$, we also cycle through the environment goals $\psi^a_1 \ldots \psi^a_m$. The dashed arrows indicate that the environment goals are skipped if the system is forced to take a transition that does not bring the play closer to the next ψ^a_j. Still, there always exists a path through all ψ^a_j.

evaluation of the $\mathsf{Reach}(\ldots)$ subformulas are actually taken. The strategy extraction approach to always pick a system's choice that is found as early as possible during the computation of W (while the greatest fixed points are completely evaluated) from standard GR(1) synthesis is compatible with our addition of the subformula $\mu R.\mathsf{Reach}((\psi^g_j \vee Y' \vee R') \wedge X)$ in Equation 3. Since transitions within $\tilde{W}$ that are found to be in $(\psi^g_j \vee Y' \vee R') \wedge X$ earlier during the evaluation of R are preferred, this guarantees that the path out of the waiting phase in X that we search for in the third line of Equation 3 is actually contained in the extracted strategy. Even more, from every position in X that is not already in Y (i.e, for which the strategy part to reach the next system goal has not yet been computed), the extracted strategy always contains the *shortest* path to reach system goal ψ^g_j or a position in Y that is closer to the next system goal.

In order to integrate the fourth line of Equation 3 into the strategy extraction process, we treat it on the level of system goals. While an ordinary generalized reactivity(1) strategy cycles through all of the system goals during execution, our modified strategies cycle through the system goals *and* the environment goals. This is illustrated in Figure 2. For the environment goals, the strategy always prefers transitions that are contained in $((\psi^a_j \vee R') \wedge Z)$ early during the evaluation of the μR fixed point. The only difference to extracting a strategy for a system goal is that once the system would be forced to take a transition that does not bring the play closer to ψ^a_j, the strategy just continues with cycling through the environment and system goals, as the environment then had its chance to reach its next goal.

As standard strategy extraction thus suffices, it remains to be proven that $\tilde{W}$ characterizes the positions in the game from which the system player has a cooperative strategy.

Theorem 1: Let $\mathcal{G}$ be a GR(1) game built from a generalized reactivity(1) specification $\varphi = (\varphi^a_i \wedge \varphi^a_s \wedge \varphi^a_l) \rightarrow_s (\varphi^g_i \wedge \varphi^g_s \wedge \varphi^g_l)$. The positions characterized by $\tilde{W}$ computed according to Equation 3 are precisely the ones from which a cooperative strategy to win $\mathcal{G}$ exists.

Proof: For the first direction of the proof, it needs to be shown that if some position v is included in $\tilde{W}$, then there exists a winning cooperative strategy from v. By the first three lines of Equation 3, positions in $\tilde{W}$ are required to be winning for the standard GR(1) winning condition. Furthermore, from every position in a maximally evaluated set X, the addition of the $\mu R.\mathsf{Reach}((\psi^g_j \vee Y' \vee R') \wedge X)$ conjunct ensures that leaving X or getting closer to a system goal does not require the falsification of some liveness assumption. Also, since Equation 3 is evaluated based on $\tilde{E}_1$, the system can achieve this without enforcing some safety assumption violation. As leaving X implies that a play gets closer to a liveness goal, which is eventually reached, we know that from any position in $\tilde{W}$, every system goal can eventually be reached in a cooperative way if the assumptions are fulfilled.

By the last line of Equation 3, we furthermore know that from every position in $\tilde{W}$, there exists a path to every environment goal that does not leave $\tilde{W}$. Both facts together imply that if we apply the strategy extraction process described above this theorem, we obtain a cooperative strategy to win $\mathcal{G}$ from v.

For the converse direction, observe that the constraint $\mu R.\mathsf{Reach}((\psi^g_j \vee Y' \vee R') \wedge X)$ that was added to the original GR(1) fixpoint equation in Equation 3 removes only positions from X from which there is no path that eventually takes a transition in $\psi^g_j \vee Y'$ without leaving X beforehand. When Z is fully evaluated, this means that any strategy that makes use of this behavior would be uncooperative (as it never gets closer to the system goal – thus, such a strategy has to rely on falsifying the liveness assumptions), so removing such states from X by the addition of $\mu R.\mathsf{Reach}((\psi^g_j \vee Y' \vee R') \wedge X)$ does not rule out any cooperative behavior. Likewise, the addition of $\mu R.\mathsf{Reach}((\psi^a_k \vee R') \wedge Z)$ does not rule out any cooperative behavior. The only case in which the conjunction with $\mu R.\mathsf{Reach}((\psi^a_k \vee R') \wedge Z)$ removes some positions is when all sequences of transitions within Z that end with a transition in ψ^a_i lead to positions that are already found to be losing for the system player. Again, cooperative strategies cannot contain such transitions, so the addition of this conjunct is justified.

Now assume that we have a cooperative strategy to win $\mathcal{G}$ that visits a set of positions $P \subseteq \mathcal{V}$ in the game. By the assumption that the strategy is correct, we have that $P \subseteq W$ for the set of winning positions W computed by the fixpoint formula from Equation 1. As in Equation 3, we only added some conjuncts that we know not to rule out cooperative behavior, we have that no position from P is removed by these conjuncts, so we know that $P \subseteq \tilde{W}$. ■

Note that the requirement for the synthesized implementation to be cooperative can sometimes be very restrictive. In scenarios in which the environment cannot be prevented from bringing itself into a deadlock situation, we cannot find a cooperative implementation. To mitigate this problem to some extent, we can compute two strategies: a cooperative one (from any starting position that admits a cooperative strategy) and a non-cooperative implementation (using standard GR(1) synthesis). Our generated overall strategy is then an *overlay* of these: If the play at some point lands in a position from which a cooperative implementation exists (but none was available from the starting position), then the resulting strategy switches to cooperative behavior. It should be noted, however, that environment deadlocks indicate a potential problem with the specification, and thus only few reasonable specifications have them.

To conclude the discussion, we would like to mention that Equation 3 can still be evaluated using *binary decision diagrams* (BDDs, [17]), which are a data structure for the symbolic manipulation of state (or position) sets. As the good efficiency of GR(1) synthesis is due to the usage of this data structure, this property of our modified synthesis algorithm is crucial for its application in practice.

VI. Experiments & Example

We implemented our cooperative strategy computation approach as a plug-in to the GR(1) synthesizer `slugs` [16]. The plug-in is included in the latest version of `slugs` and uses BDDs as data structure for symbolic reasoning. All computation times reported in the following have been obtained on an Intel Core i5-4200U CPU running an x64 Linux at a peak speed of 2.3GHz. Overall, 4 GB of main memory were available. The computed cooperative strategies are not overlayed with non-cooperative ones.

Starting with a GR(1) encoding of the motivating example from Section II, `slugs` needed 0.7 seconds to find the specification to be realizable. Synthesizing a (BDD-based) symbolic strategy took an additional 0.1 seconds. The generated implementation is non-cooperative, as depicted in Figure 1. Overall, 83.0 MB of memory were used. Checking the existence of a cooperative implementation with the new construction from Section V took 1.2 seconds, and synthesizing the cooperative (symbolic) implementation took an additional 0.1 seconds. The overall memory usage was 83.9 MB in this case.

As the example from Section II is comparably small, we also considered a larger randomly generated workspace with 32x32 cells, depicted in Figure 3. The maze-like structure of the workspace is challenging for GR(1) synthesis, as the intermediate results computed by a GR(1) synthesizer also contain the solution to the all-pairs shortest path problem in the workspace. This time, neither the moving obstacle, nor the robot can move diagonally. There is also a door (marked by a D), which is assumed to always eventually be open. The robot to be controlled cannot pass through the door while it is not open. As the position space of the game consists of (1) the position of the robot, (2) the position of the moving obstacle, and (3) the state of the door, we have 2^{21} positions in the game overall (with 5 bits per x or y coordinate of the robot or the obstacle each).

Despite this size, the standard GR(1) synthesis procedure solved the non-cooperative synthesis problem in 24.5+3.2 seconds, the latter being the time for strategy computation. The memory usage was 204.9 MB. Solving the cooperative synthesis problem took 454.5+9.6 seconds and used 590.3 MB of memory. During the execution, 23276 iterations of computing the inner-most μR fixed point were performed overall, which shows that this example is actually quite difficult to compute and explains the long computation times.

We did not perform experiments with an abstraction of the physical workspace that is smarter than the grid-based one

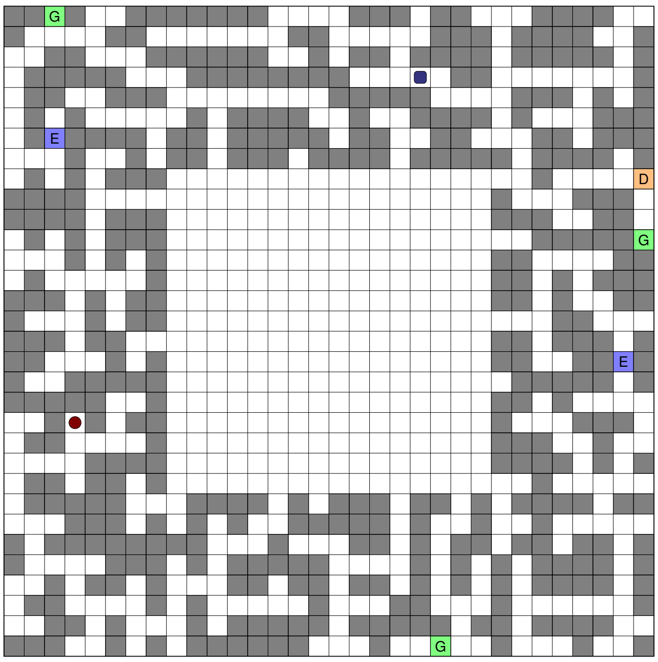

Fig. 3. A maze-like workspace, showing the initial positions of a robot to be controlled (the dot) and a moving obstacle (rounded square). Doors, system goals, and environment goals are marked with a D, a G, or an E, respectively.

(as, e.g., the abstraction based on polyhedron-shaped regions of interest performed in LTLMoP [18], [1]), but expect the approach to scale there even better, as such abstractions reduce the number of positions in the synthesis games.

VII. Conclusion

In this paper, we presented a reactive synthesis approach for obtaining *cooperative high-level robot controllers*. These allow the environment to start working towards the satisfaction of its assumptions at any point during the execution of the controller. Cooperative controllers are particularly helpful in multi-robot scenarios or settings that involve humans operating in the same space as robots, as they make sure that the controller does not actively work against the satisfaction of the assumptions. Our approach is a modification of generalized reactivity(1) synthesis and we showed that while the requirement for the controller to be cooperative makes synthesis harder, the approach can still scale to quite large scenarios. To keep the presentation simple, the robot dynamics that we used in our examples is very simple. However, our approach is suitable for all types of dynamics for which GR(1) synthesis can be applied, and thus generalizes well to more complex cases.

Acknowledgements

The authors thank Ufuk Topcu for feedback on an early version of this work.

This work was supported in part by the Austrian Science Fund (FWF) through the research network RiSE (S11406-N23), by the European Commission through the projects STANCE (317753) and IMMORTAL (644905), and by the Institutional Strategy of the University of Bremen, funded by the German Excellence Initiative.

References

[1] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[2] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010*, K. H. Johansson and W. Yi, Eds. ACM, 2010, pp. 101–110.

[3] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 911–938, 2012.

[4] V. Raman and H. Kress-Gazit, "Synthesis for multi-robot controllers with interleaved motion," in *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. IEEE, 2014, pp. 4316–4321.

[5] J. Tumova, B. Yordanov, C. Belta, I. Cerna, and J. Barnat, "A symbolic approach to controlling piecewise affine systems," in *Proceedings of the 49th IEEE Conference on Decision and Control, CDC 2010, December 15-17, 2010, Atlanta, Georgia, USA*. IEEE, 2010, pp. 4230–4235.

[6] A. Pnueli and R. Rosner, "On the synthesis of an asynchronous reactive module," in *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, ser. Lecture Notes in Computer Science, G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, Eds., vol. 372. Springer, 1989, pp. 652–671.

[7] D. Barnes, R. Aylett, A. Coddington, and R. Ghanea-Hercock, "A hybrid approach to supervising multiple co-operant autonomous mobile robots," in *Advanced Robotics, 1997. ICAR '97. Proceedings., 8th International Conference on*, Jul 1997, pp. 531–536.

[8] J. P. Desai and V. Kumar, "Motion planning for cooperating mobile manipulators," *J. Field Robotics*, vol. 16, no. 10, pp. 557–579, 1999.

[9] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *I. J. Robotic Res.*, vol. 34, no. 1, pp. 218–235, 2015.

[10] J. A. DeCastro, R. Ehlers, M. Rungger, A. Balkan, P. Tabuada, and H. Kress-Gazit, "Dynamics-based reactive synthesis and automated revisions for high-level robot control," *CoRR*, vol. abs/1410.6375, 2014.

[11] K. W. Wong and H. Kress-Gazit, "Let's talk: Autonomous conflict resolution for robots carrying out individual high-level tasks in a shared workspace," in *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May*, 2015, pp. 339–345.

[12] R. Bloem, R. Ehlers, S. Jacobs, and R. Könighofer, "How to handle assumptions in synthesis," in *Proceedings 3rd Workshop on Synthesis, SYNT 2014, Vienna, Austria, July 23-24, 2014.*, ser. EPTCS, K. Chatterjee, R. Ehlers, and S. Jha, Eds., vol. 157, 2014, pp. 34–50.

[13] R. Bloem, R. Ehlers, and R. Könighofer, "Cooperative reactive synthesis," in *13^{th} International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2015, to appear.

[14] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*. IEEE Computer Society, 1977, pp. 46–57.

[15] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*. IEEE, 2012, pp. 5163–5170.

[16] R. Ehlers, V. Raman, and C. Finucane, "Slugs GR(1) synthesizer," 2013–2015, available at https://github.com/LTLMoP/slugs.

[17] R. E. Bryant, "Symbolic manipulation of boolean functions using a graphical representation," in *Proceedings of the 22nd ACM/IEEE conference on Design automation, DAC 1985, Las Vegas, Nevada, USA, 1985.*, H. Ofek and L. A. O'Neill, Eds. ACM, 1985, pp. 688–694.

[18] C. Finucane, G. Jing, and H. Kress-Gazit, "LTLMoP: Experimenting with language, temporal logic and robot control," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*. IEEE, 2010, pp. 1988–1993.

Appendix

As a supplement to the main part of the paper, we want to shortly discuss the drawbacks of alternatives to the cooperative synthesis approach presented in the main part of the paper. The two alternatives considered in the following are:

1) Using an different parameter for strategy extraction with the GR(1) synthesis tool of `jtlv`
2) Modifying the specification to disallow uncooperative behavior by the system to be synthesized

JTLV: The `jtlv` framework for formal verification comes with a generalized reactivity(1) synthesis tool that has a couple of options for its implementation extraction part. In particular, the user can give an arbitrary prioritization to the the aims of the generated strategy to:

1) visit the current goal,
2) get closer to the current goal, and
3) take a transition that does not satisfy the current environment liveness goal.

As there are six permutations of these aims, `jtlv` has six possible settings. Using the prioritization 1-2-3 should intuitively help with extracting a strategy that is cooperative. However, this is not the case.

Consider the following GR(1) specification over $\mathcal{X} = \{d\}$ and $\mathcal{Y} = \{b_0, b_1\}$:

$$\begin{aligned}
&\big(d \wedge \Box((\neg b_0 \wedge \neg b_1 \wedge \neg d) \rightarrow \bigcirc d) \\
&\quad \wedge \Box(\neg b_0 \wedge \neg b_1 \rightarrow \neg \bigcirc d)\big) \\
\rightarrow_s &\big((\neg b_0 \wedge \neg b_1) \\
&\quad \wedge \Box((\neg b_0 \wedge \neg b_1) \rightarrow \neg \bigcirc b_1) \\
&\quad \wedge \Box((b_0 \wedge \neg b_1) \rightarrow (\neg \bigcirc b_0 \vee \neg \bigcirc b_1)) \\
&\quad \wedge \Box((b_0 \wedge b_1) \rightarrow (\neg \bigcirc b_1)) \\
&\quad \wedge \Box\Diamond(b_0 \wedge b_1)\big)
\end{aligned}$$

The specification allows the system to enforce that the safety assumptions are violated by choosing $\{b_0 \mapsto \mathbf{false}, b_1 \mapsto \mathbf{false}\}$ two times in a row. Alternatively, the system can choose the output sequence $(\{b_0 \mapsto \mathbf{false}, b_1 \mapsto \mathbf{false}\}\{b_0 \mapsto \mathbf{true}, b_1 \mapsto \mathbf{false}\}\{b_0 \mapsto \mathbf{false}, b_1 \mapsto \mathbf{true}\}\{b_0 \mapsto \mathbf{true}, b_1 \mapsto \mathbf{true}\})^\omega$, which allows the environment to play $(\{d \mapsto \mathbf{false}\})^\omega$, satisfying the assumptions.

Starting from the position $\{d \mapsto \mathbf{true}, b_0 \mapsto \mathbf{false}, b_1 \mapsto \mathbf{false}\}$, `jtlv` computes an implementation that chooses $\{b_0 \mapsto \mathbf{false}, b_1 \mapsto \mathbf{false}\}$ twice, which is uncooperative behavior. This implementation is computed for all possible parameter settings for the strategy extraction routine of `jtlv`.

By analyzing the scenario by hand, we can see why this is the case. The synthesized implementation cannot wait for an environment goal to be reached as there is only the implicit **true** environment goal in the specification that is always reached. Thus, at any step of its execution, it can only get closer to the goal or reach the goal in the next step. When moving closer to the goal, from every position p, the strategy extraction routine takes outgoing transitions to positions p' that were found earlier in the middle fixed point than p. This is necessary for soundness of the strategy extraction routine. From position $p = \{d \mapsto \mathbf{true}, b_0 \mapsto \mathbf{false}, b_1 \mapsto \mathbf{false}\}$, the number of steps needed to falsify the environment assumptions is lower than the number of steps needed to reach a system goal. Thus, p is found in the middle fixed point before a transition from p towards the system goal is found, and the only transitions that are available to the strategy extraction routine in such a case are the ones that work towards the violation of the assumptions. Thus, the generated implementation is uncooperative.

While the specification in this example is quite trivial, it is well-suited to show our main point, namely that cooperation with the environment needs to be considered explicitly in the synthesis approach.

Specification modification:

A straightforward alternative to cooperative synthesis is to modify the specification to rule out uncooperative behavior. In the main part of the paper, we argued that doing so manually defeats the purpose of synthesis, as it would require the user to enumerate situations in which the controller to be synthesized could behave in an uncooperative way. After all such situations have been identified, safety guarantees could be added to the specification by the user to prevent unwanted uncooperative behavior by the system.

There are however also specifications for which uncooperative behavior cannot be ruled out by adding safety guarantees. As an example, consider the following GR(1) specification over $\mathcal{X} = \emptyset$ and $\mathcal{Y} = \{x_0, x_1\}$:

$$\begin{aligned}
&\Box\Diamond(x_0 \wedge x_1) \\
\rightarrow_s &(\neg x_0 \wedge x_1) \\
&\quad \wedge \Box\Diamond(\neg x_0 \wedge \neg x_1) \\
&\quad \wedge \Box((\neg x_0 \wedge \neg x_1) \rightarrow \neg \bigcirc x_1) \\
&\quad \wedge \Box((x_0 \wedge \neg x_1) \rightarrow \neg \bigcirc x_0 \vee \neg \bigcirc x_1) \\
&\quad \wedge \Box((\neg x_0 \wedge x_1) \rightarrow \bigcirc x_0 \vee \bigcirc x_1) \\
&\quad \wedge \Box((x_0 \wedge x_1) \rightarrow \bigcirc x_1)
\end{aligned}$$

The specification essentially states that x_0 and x_1 together implement a binary counter with range $\{0, 1, 2, 3\}$ that the system can increase or decrease by (at most) 1 in every step. The proposition x_0 represents the least significant bit of the counter, while x_1 is the most significant bit. The counter starts with a value of 2. The system must set this counter to 0 infinitely often, while we have an environment liveness assumption that states that the value is 3 infinitely often.

An uncooperative system can simply always set the value to 2 all of the time. When adding safety assumptions to rule out this behavior, we have many options. One option is to enforce that the value of the counter cannot stay at 2 during a step of the system's execution. This however still allows the system to oscillate between counter values of 1 and 2, which does not help. We now have a choice of either ruling out a transition from counter value 1 to 2 or from 2 to 1 by adding a corresponding safety guarantee. In the former case, we would prevent the system from satisfying the liveness assumption, whereas in the second case, we would prevent it from satisfying the liveness guarantee. So both of the two options are not suitable.

Now we could also try to add liveness guarantees to force the system to give the environment some leeway for satisfying its assumptions. However, adding a guarantee that infinitely often, uncooperative transitions are not taken does not help. Also, adding the liveness assumptions as liveness guarantee does not help either, as that still allows the system to wait for progress towards satisfying a liveness assumption, which is undesired behavior.

In addition to the observation that cooperative behavior cannot be easily enforced in this scenario by modifying the specification, we can see that even in this very small example, there can be multiple ways in which the system "may get stuck" (at values 0, 1, and 2, or oscillating between 0 and 1 or 1 and 2). Thus, asking the user to modify the specification to exclude all such cases can lead to a substantial blow-up of the specification. But also the automatic adaptation of the specification is difficult as our example shows that additional system propositions may be needed.

Furthermore, automatic modification of the system specification has the problem that we need to encode into the game that from every state of the controller to be synthesized, there should always *exist* the possibility for the environment to satisfy its assumptions. As this existence requirement is incompatible with the winning condition of the game (which states that for all environment behaviours, there should exist a winning trace), we could only modify the specification such that from every state in the controller, there exists a fixed predetermined way to reach any environment goal.

The cooperative synthesis approach now computes precisely such paths. The only difference to modifying the specification is that the resulting paths are not added to the specification. As they may need to be changed whenever the specification is changed, this is however no drawback, as the obtained information cannot be reused after a specification change anyway.