2016 American Control Conference (ACC)
http://www.cds.caltech.edu/~murray/preprints/fm16-acc.pdf

Symbolic construction of GR(1) contracts for systems with full information

Ioannis Filippidis and Richard M. Murray

***Abstract*— This work proposes a symbolic algorithm for the construction of assume-guarantee specifications that allow multiple agents to cooperate. Each agent is assigned goals expressed in a fragment of linear temporal logic known as generalized Streett with one pair, GR(1). These goals may be unrealizable, unless each agent makes additional assumptions, about the behavior of other agents. The algorithm constructs a contract among the agents, in that only the infinite behavior of the given goals is constrained, known as liveness, not the finite one, known as safety. This defers synthesis to a later stage of refinement, modularizing the design process. We prove that there exist GR(1) games that do not admit any refining GR(1) contract. For this reason, we formulate contracts with nested GR(1) properties and auxiliary communication variables, and prove that they always exist. The algorithm's fixpoint structure is similar to GR(1) synthesis, enjoying time complexity polynomial in the number of states, and linear in number of recurrence goals.**

I. INTRODUCTION

The design and construction of a large system relies on the ability to divide the problem into smaller ones. Each subproblem involves a subset of the system, and may itself be refined further into smaller problems. The subsystems that result from the smaller problems are considered as modules of the larger system. In many cases, the modules interact with each other, either physically, or as software, or both. For this reason, the interaction between modules needs to be constrained, to ensure that they can perform their operation as intended.

A representation with precise syntax and semantics, or *specification*, is desirable to describe each module, and its interaction with other modules. Broadly, the system designer is faced with two problems. First, to specify the modules, with detail sufficient to allow for implementation, preferably automated. Second, to look at each module's specification, construct an implementation, and assemble the results.

In this work, we are interested in automating the first step, modularization of a design that has been partially specified by a human. Note that human input is necessary, in one form or another, because the algorithm cannot know what the modules are intended for. These *primitive* specifications may not yield a coherent system. Starting from these, the proposed algorithm constructs a collection of new specifications for the modules, a *contract* [1], [2]. Each new specification is implementable, and, together, they refine the original intent.

A common refinement is *synthesis*, whereby the immediate behavior of all modules is designed at once. We want to avoid doing so, because such refinement is premature. Instead, we want to restrict only the *infinite* behavior of the system. We work under the assumption that the modules can observe all variables in the problem, known as *full information*. To this, we are motivated by the undecidability of distributed synthesis [3], [4], except for restrictions of the communication architecture [5], the desired behaviors [6], or the search space [7].

The paper is organized as follows. Sec. II reviews preliminaries on logic and games, where we formulate concepts for multi-player games, in an interleaving logic representation. Contracts are defined in Sec. III, and our purpose to find realizable module specifications high in the refinement hierarchy. We want contracts as Streett(1) specifications, but prove in Sec. III-B that these do not exist in general. For a restricted case, this limitation is lifted in Sec. IV, where we introduce nested GR(1) specifications, and an algorithm to construct them. In the presence of multiple recurrence goals, we prove in Sec. IV-B that auxiliary variables must be introduced, and thus define a distributed scheme of switching between nested GR(1) games, one for each recurrence goal. Examples are given in Sec. V, relevant work and conclusions are discussed in Sec. VI.

II. PRELIMINARIES

A. Temporal logic and μ-calculus

Linear temporal logic (LTL) [8] can be used to describe sets of infinite sequences. Let $\mathcal{V}$ be a set of variable symbols $p, q, \dots$ that take integer or Boolean values in $\mathcal{D} \triangleq \mathbb{N} \cup \{\text{TRUE}, \text{FALSE}\}$, where $\mathbb{N} \triangleq \{0, 1, \dots\}$. Denote with $[A \to B]$ the set of

The authors are with Control and Dynamical Systems, California Institute of Technology, Pasadena, CA, 91125, USA. E-mail: {ifilippi, murray}@caltech.edu

functions with domain A and range a subset of B (notation from [9, p.48]). Let $V \triangleq [\mathcal{V} \to \mathcal{D}]$ be the set of assignments of values to variables. For a sequence $w \in \Omega \triangleq [\mathbb{N} \to V]$, let $w[i \dots]$ denote the subsequence $w[i]w[i+1] \dots$. Besides the Boolean operators "and" $\wedge$, "or" $\vee$, "not" $\neg$, LTL includes the future operators "next" p' (prime), "always" $\Box p$, "eventually" $\Diamond p$, and "until" $p \mathcal{U} q$, and past operators "weak previous" $\ominus p$ and "historically" $\boxminus p$. The semantics of LTL [8], [10] defines whether a word w is a *model* of a formula φ at position i, denoted by $(w, i) \models \varphi$, as follows:

- $(w, i) \models \varphi'$ iff $\big(w, (i+1)\big) \models \varphi$
- $(w, i) \models \varphi \mathcal{U} \psi$ iff $\exists k \in \mathbb{N} : \big(w, (i+k)\big) \models \psi$ and $\forall j \in \mathbb{N}_{<k} : \big(w, (i+j)\big) \models \varphi$,

and $\Diamond \varphi \triangleq \top \mathcal{U} \varphi$ and $\Box \varphi \triangleq \neg \Diamond \neg \varphi$.

- $(w, i) \models \ominus \varphi$ iff $i = 0$ or $\big(w, (i-1)\big) \models \varphi$
- $(w, i) \models \boxminus \varphi$ iff $\forall j \in \mathbb{N}_{\leq i} : \big(w, (i-j)\big) \models \varphi$.

Given a first-order formula φ over variables in $\mathcal{V}$, define the set of models $[\![\varphi]\!]_{\mathcal{V}} \triangleq \{u \in V : u \models \varphi\}$. Given an LTL formula ψ, define the set of models $[\![\psi]\!]_{\mathcal{V}} \triangleq \{w \in \Omega : (w, 0) \models \varphi\}$. For readability, occasionally we refer to a set $[\![f]\!]$ by mentioning f. With the Cantor topology over Ω [11], if a formula describes an open (closed) set, then it is called a liveness (safety) property [12]. For example, $\Box \Diamond p$ describes liveness, and $\Box(p' = \neg p)$ safety. Properties of the form $\Box \Diamond p$ are known as *recurrence*, and $\Diamond \Box p$ as *persistence* [13].

The modal μ-calculus [14] extends modal logic with a least (greatest) fixpoint operator μ (ν). For example, if $[\![S]\!]$ is a set of system states, and $[\![f(S)]\!]$ the set of states that can be reached in one time step from some state in $[\![S]\!]$, then $[\![\mu X : I \vee f(X)]\!]$ is the set of states that are inside, or reachable in a finite number of time steps, starting from within the set $[\![I]\!]$. For comparison, $[\![\mu X : f(I) \vee f(X)]\!]$ contains states that are reachable in a finite (non-zero) number of time steps, and $[\![\mu X : f(X)]\!] = \emptyset$. For a formal definition, we refer to [15], [16].

B. Games

1) Representation: The behavior of a collection of n modules can be represented as a game with multiple players. Here, we consider turn-based games, where a single player moves at each change of state [17]. The game is synchronous, in that players move in a fixed order that repeats. Define the (finite) set $\mathcal{X}_j$ of variables controlled by player j. By x_j we will denote both the tuple of symbols in $\mathcal{X}_j$, as well as a tuple of values assigned to those symbols. Let $x \triangleq \langle x_0, x_1, \dots, x_{n-1} \rangle$ be the aggregate state. Let $\bar{x}_j$ denote (either a tuple of, or an assignment to) variables in $\bigcup_{k=0, k \neq j}^{n-1} \mathcal{X}_k$. We assume full information, meaning that each player can observe all the variables. Let i be an auxiliary variable that signifies which player should move next, $\mathcal{V} \triangleq \{i\} \cup \bigcup_{j=0}^{n-1} \mathcal{X}_j$.

We use an interleaving representation of the game in temporal logic, with one player moving in each time step. The assignments in V can be regarded as game "nodes". The partition $V_j \subseteq V$ contains nodes with $i = j$, and $\bar{V}_j \triangleq V \backslash V_j$. Player j moves only from nodes in V_j, by assigning values to variables in x_i, and incrementing i. The transition relation of player j is an action formula $\hat{\rho}_j(x, x'_j)$ (Boolean-valued formula over primed and unprimed variables) [18]. In an interleaving representation of a synchronous turn-based game, the transition relations are $\rho_j(x, x'_j, i) \triangleq \text{ite}\big(i \neq j,\ x'_j = x_j,\ \hat{\rho}_j \wedge (i' = i \oplus_n 1)\big), \bar{\rho}_j(x, \bar{x}'_j, i) \triangleq \bigwedge_{k \neq j} \rho_k(x, x'_k, i)$, where $a \oplus_c b \triangleq (a + b) \mod c$, and $\text{ite}(a, b, c)$ the ternary conditional connective (if a then b else c).

2) Winning: A *play* is an infinite sequence of nodes in V that a game visits, as players move. Given a play w, and a set W_j called *winning condition* for player j, if $w \in W_j$, then player j is a *winner* in the play w. In practice, a temporal logic formula φ_j is used to *describe* the winning condition W_j. Any LTL winning condition φ_j can be represented as a conjunction of a safety with a liveness formula [18], [12]. The complexity of solving a game depends on the liveness formula, as captured by its *Rabin index* [11], [13], [15]. Each Rabin index is associated with a syntactic fragment of LTL, with normal forms those of Streett $\bigwedge(\Box\Diamond \vee \Diamond\Box)$ (conjunctive), and Rabin $\bigvee(\Box\Diamond \wedge \Diamond\Box)$. Each formula $\Box\Diamond \vee \Diamond\Box$ ($\wedge$) is called a Streett (Rabin) *pair*. A *generalized* pair has the form $(\bigvee^N \Diamond\Box) \vee \bigwedge^M \Box\Diamond$. Games with a single generalized Streett pair (generalized reactivity(1) or GR(1)) [19] can be solved with $\mathcal{O}(NM\,|V|^2)$ controllable predecessor operations [16, Thm.3].

Let the set $M_j \subseteq \mathbb{N}$ with $|M_j| < \infty$ be a finite amount of memory. A (deterministic) *strategy* for player j is a function $T_j \in \mathcal{T}_j \triangleq [V_j \times M_j \to \bar{V}_j \times M_j]$. Let $\bar{\mathcal{T}}_j \triangleq \prod_{k \neq j} \mathcal{T}_k$. For a node $u \in V$, define the set $\text{Plays}(u, T_0, \dots, T_{n-1}) \subseteq \Omega$ of all the plays possible if, for each $j < n$, player j uses strategy T_j. For deterministic strategies, $\text{Plays}(u, T_0, \dots, T_{n-1})$ is a singleton. Let a *realization from* u be $\mathcal{L}(j, u, T_j) \triangleq \bigcup_{\bar{T}_j \in \bar{\mathcal{T}}_j} \text{Plays}(u, T_0, \dots, T_{n-1})$ [20]. Given a property φ, define the *winning set* of player j as $[\![\text{Win}(j, \varphi)]\!] \triangleq \big\{u \in V : \exists T \in \mathcal{T}_j : \mathcal{L}(j, u, T) \subseteq [\![\varphi]\!]\big\}$. For the games we consider, playing against arbitrary, but finite, strategies, and against arbitrary input sequences coincide [21, p.619]. So, the above definition is not restrictive. The notion that no player should violate

their allowed transitions before any other player can be expressed using past LTL [16]. Here, we define this precedence in an interleaving representation, as

Definition 1: Let ρ_e, ρ_s be action, W_e, W_s liveness formulae. Define the *strict implication* operator $\overset{\text{sr}}{\Rightarrow}$ as $(\square\rho_e \wedge W_e) \overset{\text{sr}}{\Rightarrow} (\square\rho_s \wedge W_s) \triangleq \square\big((\ominus\boxminus\rho_e) \Rightarrow \rho_s\big) \wedge ((\square\rho_e \wedge W_e) \Rightarrow W_s) = (\square\rho_e) \overset{\text{sr}}{\Rightarrow} \big(\square\rho_s \wedge (W_s \Rightarrow W_e)\big)$. We are interested in games with winning conditions that have favorable complexity, in particular

Definition 2 (Generalized Streett(1) [19]): Assume that, for all $k \in \mathbb{N}_{<N}, r \in \mathbb{N}_{<M}$, $P_{jk}(x,i), R_{jr}(x,i)$ are assertions (Boolean formulae over unprimed variables). Then, the LTL formula $\varphi_j \triangleq (\square\bar{\rho}_j) \overset{\text{sr}}{\Rightarrow} \big(\square\rho_j \wedge (\bigwedge_k \square\diamondsuit P_{jk} \Rightarrow \bigwedge_r \square\diamondsuit R_{jr})\big)$ describes a GR(1) property for player j.

III. CONTRACTS

We are interested in a method for constructing assume-guarantee specifications for a set of modules. The resulting specifications must be *realizable* [20], i.e., for some *common* initial condition(s), for each module, there should exist a strategy that implements its specification. In the context of full information, we can directly construct a strategy for each module, and thus solve this problem. Informally, a realization does not contain any quantification – it has all been removed by this point. At the cost of restricting the temporal property far beyond what is necessary. We do not want to do so. Instead, we want to refine the given specifications, in a way that fewer behaviors be removed.

In particular, a deterministic strategy defines a property that, for each input sequence, contains at most one play. Still, for each input sequence, there remain infinitely many plays that, by this point, have been excluded from the temporal property. We want to preserve these plays as admissible behaviors. Thus, we obtain a temporal property that contains more behaviors.

For the following discussion, we will need some definitions. As specification language, we choose LTL. We call specifications also properties, and define

Definition 3: Given n players, a *contract* is a partition of variables among players that defines which player controls each variable, together with an initial condition set $[\![I]\!] \neq \emptyset$, and a tuple of properties $\langle\varphi_0, \ldots, \varphi_{n-1}\rangle$, such that $\models I \Rightarrow \bigwedge_j \mathrm{Win}(j, \varphi_j)$.
A GR(1) contract contains GR(1) properties.

Definition 4: Let $\varphi(\mathcal{V}), \psi(\mathcal{W})$ be two properties over variables in $\mathcal{V}, \mathcal{W}$. If $[\![\varphi]\!]_{\mathcal{V}\cup\mathcal{W}} \subseteq [\![\psi]\!]_{\mathcal{V}\cup\mathcal{W}}$, then the property φ *refines* (or implements) ψ [20].
Given φ, *synthesis from* $u \in V$ is the construction of a realization ψ from u that refines φ. Synthesis is refinement. Not any refinement is synthesis.

The designer requires that the collective behavior of the players satisfy $\bigwedge_j \varphi_j(\mathcal{V})$. If, for some I, $\varphi \triangleq \langle\varphi_0, \ldots, \varphi_{n-1}\rangle$ is a contract, then we are done. Otherwise, we want to find a tuple $\psi \triangleq \langle\psi_0, \ldots, \psi_{n-1}\rangle$ that refines the given intent, defined as $[\![\bigwedge_j \psi_j]\!]_{\mathcal{V}} \subseteq [\![\bigwedge_j \varphi_j]\!]_{\mathcal{V}}$. In addition, we do not want ψ to restrict the finite behavior (safety) of the players more than φ. The limits of any finite behavior contained in a property P form its *closure* $\mathcal{C}_{\mathcal{K}}(P) \triangleq \{w \in [\mathbb{N} \to [\mathcal{K} \to \mathcal{D}]] : \forall k \in \mathbb{N} : \exists \sigma \in P : w[0 \ldots k] = \sigma[0 \ldots k]\}$ [22], [23]. Therefore, the requirement to preserve in ψ any finite behavior allowed by φ (thus avoid synthesizing) can be expressed as $\mathcal{C}([\![\bigwedge_j \psi_j]\!]_{\mathcal{V}}) = \mathcal{C}([\![\bigwedge_j \varphi_j]\!]_{\mathcal{V}})$. This extends to team games the notion of least restrictive safety assumption [23]. A contract $\psi(\mathcal{V})$ with the previous two properties *refines* the tuple $\varphi(\mathcal{V})$ (safety-preserving refinement over $\mathcal{V}$).

Problem 5: Given a GR(1) tuple $\varphi(\mathcal{V})$, does there exist a GR(1) contract $\psi(\mathcal{V})$ that refines φ?
In the following sections, we consider Problem 5 for incrementally larger classes of behaviors.

A. Safety

Assume that, for each player j, a safety property $\varphi_j = \square\rho_j$ is given as design intent. If a single player -1 controlled all variables in $\mathcal{V}$, then let the *cooperatively winning set* $C \triangleq \mathrm{Win}(-1, \bigwedge_j \varphi_j)$. Note that $[\![C]\!] = \emptyset$ implies that $[\![\bigwedge_j \mathrm{Win}(j, \varphi_j)]\!] = \emptyset$, so no contract exists in that case. A necessary condition for the existence of a contract is that $[\![C]\!] \neq \emptyset$, i.e., φ must be cooperatively winning. Define the existential predecessors of F as $\mathrm{Pre}_j(F) \triangleq \lambda x : \lambda i : (i = j) \wedge \exists x'_j : \hat{\rho}_j(x, x'_j) \wedge F|_{x'_j/x_j}(\bar{x}_j, x'_j, j \oplus_n 1)$, where x'_j/x_j signifies substitution of x'_j for x_j. Denote $\mathrm{Pre}(F) \triangleq \bigvee_j \mathrm{Pre}_j(F)$, and the iterated predecessors $\mathrm{Pre}^*(F) \triangleq \mu X : F \vee \mathrm{Pre}(X)$. Given φ, $[\![C]\!]$ can be computed as the greatest fixpoint $\nu Z : \mathrm{Pre}^*(Z)$ [19].

By extension of results for two-player games [23, Thm.7], the closure $\mathcal{C}([\![\bigwedge_j \varphi_j]\!])$ contains exactly those plays that do not exit C. This can be required from player j with the formula $\rho_{C,j} \triangleq C \wedge C|_{x'_j/x_j}$. Let $\bar{\rho}_{C,j} \triangleq \bigwedge_{k\neq j} \rho_{C,k}$. For safety formulae $\varphi_j = \square\rho_j$, the specifications $\psi_j \triangleq \big(\square(\bar{\rho}_j \wedge \bar{\rho}_{C,j})\big) \overset{\text{sr}}{\Rightarrow} \square(\rho_j \wedge \rho_{C,j})$, with $I \triangleq C$, form a contract that refines the tuple φ. An analogous result ensures that the safety component of properties with recurrence goals $\square\diamondsuit G_{jr}$ can be ensured by computing $\nu Z : \bigwedge_j \nu Z_j : Z \wedge \bigwedge_r \mathrm{Pre}^*\big(G_{jr} \wedge \mathrm{Pre}(Z_j)\big)$, as we prove in [24].

B. Recurrence

We consider now tuples φ with $\varphi_0 = \square\rho_0 \wedge \square\diamondsuit G$ and, for $j > 0$, $\varphi_j = \square\rho_j$. Unlike safety in Sec. III-A, we prove that for a single recurrence $\square\diamondsuit G$, GR(1) liveness assumptions over $\mathcal{V}$ may not exist.

Proposition 6: ASSUME: Define the transition relations ρ_0, ρ_1 by the game graph of Fig. 1, the set of nodes $V \triangleq \{s_0, \ldots, s_7\}$, and the goal $[\![G]\!] \triangleq \{s_6\}$ of player 0. PROVE: For all sets $[\![P]\!] \subseteq V$, with $\psi_1 \triangleq (\square\rho_0) \overset{\text{sr}}{\Rightarrow} (\square\rho_1 \wedge \square\diamond P)$ and $\psi_0 \triangleq (\square\rho_1 \wedge \square\diamond P) \overset{\text{sr}}{\Rightarrow} (\square\rho_0 \wedge \square\diamond G)$, it is $[\![\text{Win}(0, \psi_0)]\!] \cap [\![\text{Win}(1, \psi_1)]\!] = \emptyset$.

PROOF SKETCH: For any P that does not intersect $\{s_0, \ldots, s_3\}$, player 1 cannot win, because player 0 can force, and keep, the play outside of P. For similar reasons, P should intersect each of $\{s_0, s_1\}$ and $\{s_2, s_3\}$. If P intersects $\{s_0, s_1\}$ and $\{s_2, s_3\}$, then player 1 can win by always moving from s_4 to s_1, when the play comes to s_4. This forces a visit to either both s_0 and s_1, or both s_2 and s_3. So, for no P do both players have a winning strategy. A formal [25] proof can be found in [24].

By adding the edge $s_5 \wedge s_2'$ to Fig. 1, lack of liveness assumptions for the goal $\square\diamond G$ can be proved for each player. In this game, if a GR(1) contract refines $\square\diamond s_6$, then it contains $\square\diamond s_6$ (or the equiv. $s_6 \vee s_7$) as goal for at least one player. Therefore, a GR(1) contract that refines $\varphi_0 \wedge \varphi_1$ does not exist.

Also, Proposition 6 implies that, even if a problem is cooperatively realizable, existing approaches for constructing GR(1) assumptions [26], [27], [28] may find only assumptions that constrain safety.

One may wonder why we didn't consider $\square\diamond s_4$ as antecedent in ψ_1, since with $[\![P]\!] = \{s_5\}$, this may seem to work. The problematic aspect in Fig. 1 is that, in GR(1), $\square\diamond s_4$ would also have to be conjoined into the consequent of ψ_0, to tell player 0 to do so. Player 0 can violate its liveness goals, thus $\square\diamond s_4$, as long as that falsifies the antecedent $\square\diamond P$ of ψ_0, in accord with Definition 2. This is known as *trivial realizability* [23], [16], [29]. In other words, there is *circularity* of dependencies for the liveness properties [30], [31], [32], [22] that each player needs to assume about the other. In Sec. IV, we propose an algorithm that constructs nested GR(1) specifications, which enjoy complexity properties similar to GR(1) games.

IV. NESTED GR(1) GAMES

A. Single recurrence

In Sec. III-B we proved that Problem 5 can fail to have a solution. To overcome this limitation, we extend the class of properties to

Definition 7: If, in syntax, $\varphi_j = \square\bar{\rho}_j \overset{\text{sr}}{\Rightarrow} \Big(\square\rho_j \wedge \bigwedge_m \square\big((P_m \wedge \bigwedge_{k\geq m,\, l} \square\diamond\neg\eta_{kl}) \Rightarrow \diamond Q_m\big) \wedge \square(Q_m \Rightarrow \bigwedge_l \square\diamond\neg\xi_{ml})\Big)$, and, in semantics, $[\![P_{m-1}]\!] \subseteq [\![Q_m]\!] \subseteq [\![P_m]\!], [\![\eta_{ml}]\!] \subseteq [\![P_m \wedge \neg Q_m]\!]$ and $[\![\xi_{ml}]\!] \subseteq [\![Q_m]\!], [\![\xi_{ml}]\!] \subseteq [\![\neg P_{m-1}]\!]$, then we call φ_j a *nested GR(1)* property for player j.

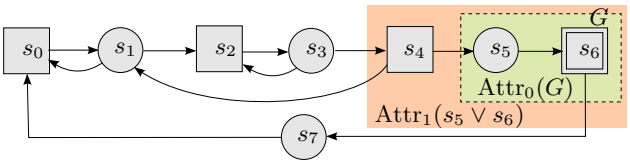


Fig. 1: Game where a liveness assumption realizable by player 1, and sufficient for player 0, does not exist. Player 0 (player 1) moves from disks (boxes).

If $a \subseteq b \subseteq \cdots \subseteq c$, then the sets $a, b, \ldots, c$ form a *chain*. A nested GR(1) property can be regarded as a request-response chain, in analogy to Rabin and Streett chains (parity) [33, p.13] [34]. Note though that synthesis for a nested GR(1) property is similar to GR(1), whereas for parity games it is unknown whether a polynomial time algorithm exists.

A nested GR(1) property enables to isolate conditional assumptions, by partitioning the game into smaller ones. Each conjunct game becomes active in P_m, it has its own assumptions $\square\diamond\neg\eta_{ml}$, and the reachability goal Q_m, tighter than the games with larger m, on which it depends. This prevents circularity of liveness dependencies. The assumptions $\square\diamond\neg\eta_{ml}$ are unconditional. Assumptions that themselves depend on other liveness assumptions become objectives in their own game. The approach of nested games is reminiscent of McNaughton's recursive algorithm for solving parity games [35].

We propose Algorithm 1, which computes a stack of nested games that cover the cooperatively winning set $[\![C]\!]$, and define a contract. So, a later visit to G is always possible, from any node in C. Define the *controllable predecessors* of F for player j as $\text{CPre}_j(F) \triangleq \lambda x : \lambda i : \bigvee_{k=0}^{n-1} \big((k = i) \wedge \neg^{k\neq j}\exists x_k' : \hat{\rho}_k(x, x_k') \wedge \neg^{k\neq j} F|_{x_k'/x_k}(\bar{x}_k, x_k', k \oplus_n 1)\big)$. The *attractor* $\text{Attr}_j(F) \triangleq \mu X : F \vee \text{CPre}_j(X)$ [36] contains those nodes from where player j is in, or can force a future visit, to F. Define the *controlled-escape* subset of S as $\text{Trap}_j(S, E) \triangleq \nu X : E \vee (\text{CPre}_j(X) \wedge S)$ that contains those nodes, from where player j can force either to remain inside $\text{Trap}_j(S, E)$, or move to E, or is already in E.

Theorem 8: ASSUME: A cooperatively realizable tuple $\varphi = \langle \varphi_0, \ldots, \varphi_{n-1} \rangle$ with $\varphi_0 = \square\rho_0 \wedge \square\diamond G$ and $\forall k > 0 : \varphi_k = \square\rho_k$. PROVE: After $\mathcal{O}(n\,|V|^2)$ calls to CPre_j, the call $\text{GAMESTACK}(0, G, C, s)$ in Algorithm 1 returns in list s a contract of nested GR(1) properties that refines φ, with $I = C$.

PROOF SKETCH: Function UNCONDASM finds the largest set r from where player j can keep the play inside the k-attractor of the j-attractor of goal g, as shown in Fig. 2. This yields an *unconditional* assumption that player j makes about player k. Iteration l of L3 produces an assumption $\square\diamond\neg\eta_{ml}$ of

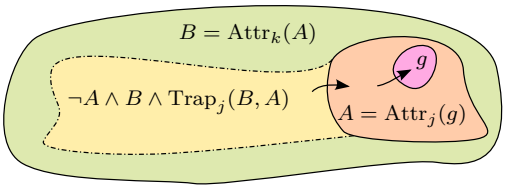


Fig. 2: Predicates computed by UncondAsm, Algorithm 1.

player j, with $\eta = \mathrm{trap} = r$, which is also a guarantee $\square\diamondsuit\neg\xi_{m'l}$ by player k, with $\xi_{m'l} = \mathrm{trap}$. The fixpoint $\nu : \mu : \mu$ for computing r in UncondAsm has alternation depth 1 [37], so it invokes $\mathrm{CPre}_?$ $\mathcal{O}(|V|)$ times. Due to determinacy [11], [21], and the definition of the cooperatively winning set C, for every n nested calls at L15, at least one call to UncondAsm at L5 removes a node from *uncovered*. So, GameStack makes $\mathcal{O}(n\,|V|^2)$ calls to $\mathrm{CPre}_?$. A formal proof can be found in [24].

Algorithm 1 can be applied also to games where the players do not move in a fixed order, because those can be described in GR(1). Concurrent [17] and asynchronous [38] games are special cases of games with partial information, which are not determined, so the inductive argument does not hold in that case.

Note that searching for fewer assumptions, inducing a smaller winning set, can be exponentially expensive, as proved for syntactic recurrence formulae in [27]. Conceptually, the nesting of games has common elements with modular game graphs [39] and open temporal logic [40].

Example 1: Let us revisit the example of Fig. 1, to observe the algorithm's execution. Player 0 wants $\square\diamondsuit G$. The first call to GameStack will call UncondAsm. Player 0 can force a visit to s_6 from the attractor $A = \mathrm{Attr}_0(s_6) = s_5 \vee s_6$. Player 1 can force A from $B = \mathrm{Attr}_1(A) = s_4 \vee s_5 \vee s_6$. But $r = \bot$, because player 1 can escape to s_1. So, a nested game is constructed over $s_0 \vee s_1 \vee s_2 \vee s_3 \vee s_4$, with player 1 wanting $\diamondsuit(s_5 \vee s_6)$. In the nested game, $A = \mathrm{Attr}_1(s_5 \vee s_6) = s_4 \vee s_5 \vee s_6$. The attractor $B = \mathrm{Attr}_0(s_4 \vee s_5 \vee s_6) = \top$, and player 0 can keep player 1 in there, until player 0 visits $s_4 \vee s_5 \vee s_6$. So, in the nested game, player 1 makes the assumption $\square\diamondsuit\neg(s_0 \vee s_1 \vee s_2 \vee s_3)$. This covers the cooperative winning set, in this example the entire game graph.

B. Conjoined recurrence

Let us consider Problem 5 again, this time for any GR(1) tuple φ. The difference with Sec. IV-A is that there may be more than one recurrence goals in φ. In this case, we prove with the counterexample of Fig. 3 that a GR(1) assumption may not exist. This includes *any* GR(1) assumption, not only those that preserve safety. In other words, no transitions can be removed, and no additional recurrence goals can produce two GR(1) properties that both players can realize from the same initial set.

Algorithm 1 Construction of nested GR(1) specification, for a single recurrence goal G.

```
1:  def GameStack(j, G, uncovered, stack)
2:      trap ← ⊤, goal ← G, stack ← set()
        ▷ Create unconditional assumptions η_ml
3:      while ⟦trap⟧ ≠ ∅ :                    ▷ until stuck
4:          for k ≠ j :
5:              attr, trap ← UncondAsm(j, k, goal)
6:              goal ← attr ∨ trap
7:              assumptions.add((k, □◇¬trap))
8:              if ⟦trap⟧ ≠ ∅ :
9:                  break
10:     game ← (j, goal, G, assumptions)
11:     stack.append(game)
12:     uncovered ← uncovered ∧ ¬goal
        ▷ Covered cooperatively winning set?
13:     if ⟦uncovered⟧ = ∅ :
14:         return
        ▷ Construct a nested game
15:     GameStack(j ⊕_n 1, goal, uncovered, stack)
16: def UncondAsm(j, k, g)
17:     A ← Attr_j(g)
18:     B ← Attr_k(A)
19:     r ← ¬A ∧ B ∧ Trap_j(B, A)
20:     return A, r
```

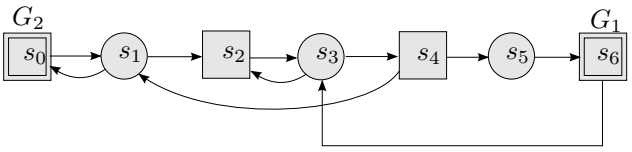


Fig. 3: No GR(1) assumption exists for this example.

Proposition 9: Assume: Define the transition relations ρ_0, ρ_1 as in Fig. 3, and goals $G_1 \triangleq \{s_6\}, G_2 \triangleq \{s_0\}$ of player 0. Prove: For all properties P such that $\psi_1 \triangleq (\square\rho_0) \overset{\mathrm{sr}}{\Rightarrow} (\square\rho_1 \wedge P)$ and $\psi_0 \triangleq (\square\rho_1 \wedge P) \overset{\mathrm{sr}}{\Rightarrow} (\square\rho_0 \wedge \square\diamondsuit G_1 \wedge \square\diamondsuit G_2)$ be GR(1) properties, it is $[\![\mathrm{Win}(0, \psi_0)]\!] \cap [\![\mathrm{Win}(1, \psi_1)]\!] = \emptyset$.

A proof can be found in [24]. The reason for this limitation is that auxiliary variables are needed for coordination between players, to ensure they play the same nested GR(1) game. This relates to the requirement for memory in strategies for GR(1) games [36], [16]. Each player can have their own memory, but if these local memories are not coordinated, the situation is similar to one player with two independent memories over a partition. In other words, additional *common* memory is required.

However, Problem 5 does not apply any more, because ψ_j will be defined over $\mathcal{W}$, which is $\mathcal{V}$ aug-

mented with auxiliary variables. Given a property P over variables $\mathcal{W} \supseteq \mathcal{K}$, define projection on $\mathcal{K}$ as $\pi_{\mathcal{K}}(P) \triangleq \{w \in [\mathbb{N} \to [\mathcal{K} \to \mathcal{D}]] : \exists \sigma \in P : \forall k \in \mathbb{N} : w[k] = \sigma[k] \cap [\mathcal{K} \to \mathcal{D}]\}$. Define closure preservation over the original variables $\mathcal{V}$ as $\pi_{\mathcal{V}}\big(\mathcal{C}_{\mathcal{W}}(\llbracket \bigwedge_j \psi_j \rrbracket_{\mathcal{W}})\big) = \mathcal{C}_{\mathcal{V}}(\llbracket \bigwedge_j \varphi_j \rrbracket_{\mathcal{V}})$. If $\mathcal{V} \subseteq \mathcal{W}$, $\psi(\mathcal{W})$ is a contract, $\llbracket \bigwedge_j \psi_j \rrbracket_{\mathcal{W}} \subseteq \llbracket \bigwedge_j \varphi_j \rrbracket_{\mathcal{W}}$ and closure is preserved over $\mathcal{V}$, then contract $\psi(\mathcal{W})$ *refines tuple* φ *with memory* $\mathcal{W} \setminus \mathcal{V}$.

Problem 10: Does there exist a class of contracts that admit synthesis in polynomial time, such that, given a GR(1) tuple $\varphi(\mathcal{V})$, there always exists a contract $\psi(\mathcal{W})$ that refines φ with memory $\mathcal{W} \setminus \mathcal{V}$?

First, the transition relations should be restricted to ensure closure (safety component), as in Sec. III-A. As shown in Sec. IV-A, each recurrence goal can be reached from anywhere in the cooperatively winning set C, by using nested GR(1) properties for the players. Therefore, playing consecutively games that cycle through all the tuples of nested GR(1) properties, one for each recurrence goal, ensures that the play repeatedly visits each recurrence goal.

For this purpose, we introduce the *auxiliary* counter variables $g_j \in \mathbb{N}_{\leq N_j}$, where N_j the number of recurrence goals G_{jr} of player j. Each counter is used to indicate a recurrence goal of player j that is actively pursued by the players, each one using the strategy for the nested GR(1) game that corresponds to that goal. The behavior of the counters is described by the formulae $\bigwedge_{r<N_j} \square\big((g_j = r) \Rightarrow \text{ite}(G_{jr},\ g_j' = g_j + 1,\ g_j' = g_j)\big), \square\big((g_j = N_j) \Rightarrow \text{ite}(\ominus(g_{j\oplus_n 1} = N_{j\oplus_n 1} - 1) \wedge (g_{j\oplus_n 1} = N_{j\oplus_n 1}),\ g_j' = 0,\ g_j' = N_j)\big)$ with initial conditions $(g_0 = 0) \wedge (\forall k > 0 : g_k = N_k)$. The game with index r is activated by using $(g_j = r)$ as requests in φ_j of Definition 7.

V. EXAMPLES

The proposed algorithms have been implemented [41] in Python, using our packages `omega` [42], [43], for symbolic solution of games, and `dd`, for binary decision diagrams (BDD) [44], [43]. `dd` offers both a pure Python implementation, and an identical interface via Cython to the C library CUDD [45], for demanding applications. They are available under a BSD license. The predicates in the constructed temporal properties are represented by BDDs.

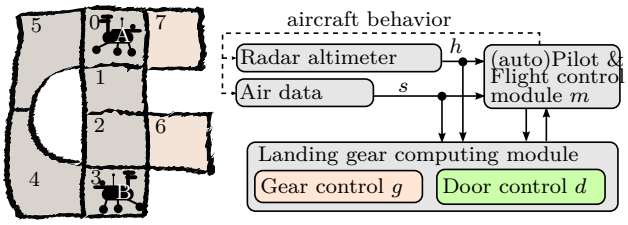
1) Grid world: The first example involves two robots A, B that are moving in a narrow passage, and need to cross each other. Their discrete world is described by cells, as in Fig. 4a, and moves are possible between adjacent cells. Define variable a (b) as the cell that robot A (B) currently is in. Each robot can remain stationary, and shall not move from, or to, a cell where another robot is, e.g., $\square(a \neq b \neq a')$ for A. Robot A (B) can move from cell $0 \mapsto 1 \mapsto 2 \mapsto 3 \mapsto 4 \mapsto 5 \mapsto 0$ ($0 \mapsto 5 \mapsto 4 \mapsto 3 \mapsto 2 \mapsto 1 \mapsto 0$, and $2 \mapsto 6 \mapsto 2$, and $0 \mapsto 7 \mapsto 0$). In other words, robot A (B) moves (counter)clockwise, and A cannot go to cells 7 and 6.

The construction of a contract for the property tuple $\varphi \triangleq (\varphi_a, \varphi_b)$ with $\varphi_a \triangleq \square\rho_a \wedge \square\diamondsuit(a = 4)$ and $\varphi_b \triangleq \square\rho_b$. starts with the closure $\mathcal{C}(\llbracket \varphi_a \wedge \varphi_b \rrbracket)$, as discussed in Sec. III-A. This removes nodes of the game graph that are unsafe for $\square\diamondsuit(a = 4)$. For example, $(a = 3) \wedge (b = 4)$ is such a node, because there is no way for A to go to $a = 4$ from there, even if both A, B were controlled by one player. The relation $\rho_{C,j}$, computed to ensure that both robots remain in the cooperatively winning set C, removes only transitions that "cross" the boundary from C to $\neg C$, as in [23]. For example, node $u \triangleq (a = 3) \wedge (b = 5) \wedge (i = 1)$ is in C, but $v \triangleq (a = 3) \wedge (b = 4) \wedge (i = 0)$ is not (recall that i signifies who should move next, with 0 for a and 1 for b). The transition from $u \mapsto v$ satisfies $\square\rho_a \wedge \square\rho_b$, but violates $\square\rho_a \wedge \square\rho_b \wedge \square\diamondsuit(a = 4)$. In the following, ρ_a, ρ_b are constrained with $\rho_{C,a}, \rho_{C,b}$.

Secondly, Algorithm 1 runs. It produces the specification ψ_a with $P_0 = C$, $Q_0 = (a = 4) \wedge (b \in \{0, \ldots 3, 6, 7\})$ and one assumption $\eta_{0,0} = \big((a = 1) \wedge \big((b \in \{2, \ldots 5\}) \vee (b = 0 \wedge i = 1) \vee (b = 6 \wedge i = 1)\big)\big) \vee \big((a = 5) \wedge \big((b \in \{0, 1\}) \vee (b = 7 \wedge i = 1)\big)\big)$. Recall from Definition 7 that ψ_a includes the conjunct $\square\big((P_0 \wedge \square\diamondsuit\neg\eta_{0,0}) \Rightarrow \diamondsuit Q_0\big)$. So, robot A has to eventually reach $\llbracket Q_0 \rrbracket \subseteq \llbracket a = 4 \rrbracket$ from any node in $\llbracket P_0 \rrbracket$. Suppose that the system is at $(a = 0) \wedge (b = 3)$. In order for A to reach 4, B must come to 6, then A cross it, and continue to reach 4. Without any liveness assumptions, A cannot do so, because B can forever stay in 3. If B continues to sit at 3, it satisfies $\rho_{C,b}$. In order to force B to move with a safety constraint, we would have to remove transitions (namely, the self-loop of B at cell 3). This would happen when applying distributed synthesis, and it would restrict the movement of B, every time it visits 3. We do not want to do so.

The generated assumption $\square\diamondsuit\eta_{0,0}$ makes it possible to force B to leave cell 3, and eventually come to cell 6. Robot A "enters" $\llbracket \eta_{0,0} \rrbracket$ by going to $a = 1$. It waits there, and the only way for B out of $\llbracket \eta_{0,0} \rrbracket$ is to eventually move to 6. Note that, at 6, B exits $\llbracket \eta_{0,0} \rrbracket$ as soon as A's turn comes, i.e., $(a = 1) \wedge (b = 6) \wedge (i = 1)$ is in $\eta_{0,0}$, but $(a = 1) \wedge (b = 6) \wedge (i = 0)$ is not. So, A cannot force violation by B of the ψ_b guarantee $\square\diamondsuit\neg\xi_{0,0}$ with $\xi_{0,0} = \eta_{0,0}$. B can keep staying at 6, until A decides to move to 2.

It is important to observe that the contract ψ preserves infinitely many behaviors for each robot.



(a) Grid world. (b) Landing gear avionics.

Fig. 4: Examples where liveness assumptions are constructed for cooperation of the modules.

This allows further design for each robot to continue independently, for example by different teams. If we synthesized directly controllers for each robot, then those would constrain the exact step-by-step behavior. This would require us to consider the exact details of both robots at once, which is not a modular approach. For example, robot B may be solar-powered, so it's movement depends on batteries, weather conditions, and other factors. How these factors interact and affect the choice of motors and batteries should concern the team that designs robot B. The team that designs A need not worry about those details, as long as team B can implement its side of the contract. Thus, this approach avoids to constrain the irrelevant details prematurely. It decomposes the design, so that individual components can be refined (or decomposed) further. If one of the robots fails later, then any replacement that fits the contract works. The contract ψ allows greater freedom for choosing replacements, without constraining irrelevant details about their step-by-step behavior. Similarly, a contract allows more choices of solutions from off-the-shelf components. Also, we didn't have to come up with assumptions manually, a challenging and error prone task.

2) Landing gear: We consider now a simplified modular avionics system for landing gear control [46], [47]. There are two modules, as shown in Fig. 4b, one representing the (auto)pilot and flight control module, another for landing gear control, which runs two independent control applications, one for control of the gear g (retracted: 0, moving: 1, fully extended: 2), another for control of the gear doors d (closed: 0, moving: 1, open: 2). The (auto)pilot controls airspeed $s \in \{0, \ldots, 1000\}$ km/h and absolute altitude $h \in \{0, \ldots, 100\}$ 100m, and the mode m (landing: 0, cruise: 1, takeoff: 2). The safety specification for all components requires that gear be fully extended below 300m $\square((g \neq 2) \Rightarrow (h > 3))$, doors be closed above 300 km/h $\square((s > 300) \Rightarrow (d = 0))$, doors be open if gear not retracted, $\square((g \neq 0) \Rightarrow (d = 2))$, gear be extended on ground, $\square((h = 0) \Rightarrow (g = 2))$, the flight control module cannot enter landing mode, unless the gear is fully extended $\square((m = 0) \Rightarrow (g = 2))$, and the gear must be retracted during cruise $\square((m = 1) \Rightarrow (g = 0))$. The (auto)pilot's goal is to be able to enter all modes $\bigwedge_{k=0}^{2} \square\diamondsuit(m = k)$.

For the goal $\square\diamondsuit(m = 0)$ three levels of nesting are constructed. At the bottom one, the doors assume that the pilot will eventually reduce the speed below 300 km/h. If so, the doors open, leading to $Q_{d,0}$ for the doors, which is $P_{g,0}$ for the gear. When the doors are open, the landing gear can start moving, and eventually extend fully. This leads to $Q_{g,0}$, which is $P_{a,0}$, from where the flight control module can switch to landing mode. To enter cruise mode, $\square\diamondsuit(m = 1)$, the nesting is different. At the bottom, the gear control assumes that eventually the height is above 300m, to retract. Then, the doors close, leading to the autopilot's attractor of cruise mode.

VI. Related Work and Conclusions

Synthesis has been applied widely to robotic and multiagent systems [48], [49], [50], [51]. The approach we proposed investigates construction of non-constraining properties, instead of synthesis. Most relevant to the work presented here are approaches for constructing GR(1) and LTL assumptions [26], [27], [28], [23], and for analyzing unrealizable specifications [52], [53], [54]. The syntactic methods in [26], [27] are enumerative in nature, in that a counter-strategy is iteratively synthesized, and consider only non-circular dependencies between modules. The semantic approach proposed here is symbolic (using BDDs), automated, and addresses circularity. The construction of safety assumptions before liveness is based on [23]. The assume-guarantee framework originates from [55], [56] that evolved to [57], [58], [59], [2]. Ideas about *temporal proof lattices* [31] and *temporal well-founded induction* [30] led to Algorithm 1. The nonexistence of memoryless GR(1) contracts is related to the need for auxiliary variables in constructing refinement maps [60].

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

References

[1] B. Meyer, "Applying "design by contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[2] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. Larsen, "Contracts for systems design," INRIA, Tech. Rep. 8147, 2012.

[3] A. Pnueli and R. Rosner, "Distributed reactive systems are hard to synthesize," in *FOCS*, vol. 2, 1990, pp. 746–757.

[4] S. Schewe and B. Finkbeiner, "Synthesis of asynchronous systems," in *LOPSTR*, 2007, pp. 127–142.

[5] B. Finkbeiner and S. Schewe, "Uniform distributed synthesis," in *LICS*, 2005, pp. 321–330.

[6] K. Chatterjee, T. Henzinger, J. Otop, and A. Pavlogiannis, "Distributed synthesis for LTL fragments," in *FMCAD*, 2013, pp. 18–25.

[7] B. Finkbeiner and S. Schewe, "Bounded synthesis," *STTT*, vol. 15, no. 5–6, pp. 519–539, 2013.

[8] A. Pnueli, "The temporal logic of programs," in *FOCS*, 1977, pp. 46–57.

[9] L. Lamport, *Specifying Systems*. Addison-Wesley, 2002.

[10] Z. Manna and A. Pnueli, "The anchored version of the temporal framework," in *Linear time, branching time and partial order in Logics and models for concurrency*, ser. LNCS. Springer, 1989, vol. 354, pp. 201–284.

[11] W. Thomas and H. Lescow, "Logical specifications of infinite computations," in *A decade of concurrency reflections and perspectives*, 1993, pp. 583–621.

[12] B. Alpern and F. B. Schneider, "Defining liveness," *IPL*, vol. 21, no. 4, pp. 181–185, 1985.

[13] Z. Manna and A. Pnueli, "A hierarchy of temporal properties," in *PODC*, 1990, pp. 377–410.

[14] D. Kozen, "Results on the propositional μ-calculus," *TCS*, vol. 27, no. 3, pp. 333–354, 1983.

[15] K. Schneider, *Verification of reactive systems: formal methods and algorithms*. Springer, 2004.

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

[17] R. Alur, T. Henzinger, and O. Kupferman, "Alternating-time temporal logic," in *JACM*, 2002, pp. 672–713.

[18] L. Lamport, "The temporal logic of actions," *TOPLAS*, vol. 16, no. 3, pp. 872–923, 1994.

[19] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *VMCAI*, 2006, pp. 364–380.

[20] M. Abadi, L. Lamport, and P. Wolper, "Realizable and unrealizable specifications of reactive systems," in *ICALP*, 1989, pp. 1–17.

[21] O. Kupferman, Y. Lustig, M. Vardi, and M. Yannakakis, "Temporal synthesis for bounded systems and environments," in *STACS*, 2011, pp. 615–626.

[22] M. Abadi and L. Lamport, "Composing specifications," *TOPLAS*, vol. 15, no. 1, pp. 73–132, 1993.

[23] K. Chatterjee, T. Henzinger, and B. Jobstmann, "Environment assumptions for synthesis," in *CONCUR*, 2008, pp. 147–161.

[24] I. Filippidis and R. M. Murray, "Symbolic construction of GR(1) contracts for synchronous systems with full information," Caltech, Tech. Rep. arXiv:1508.02705, 2015.

[25] L. Lamport, "How to write a 21st century proof," *Journal of fixed point theory and applications*, vol. 11, no. 1, pp. 43–63, 2012.

[26] R. Alur, S. Moarref, and U. Topcu, "Counter-strategy guided refinement of GR(1) temporal logic specifications," in *FMCAD*, 2013, pp. 26–33.

[27] ——, "Pattern-based refinement of assume-guarantee specifications in reactive synthesis," in *TACAS*, 2015, pp. 501–516.

[28] W. Li, L. Dworkin, and S. A. Seshia, "Mining assumptions for synthesis," in *MEMOCODE*, 2011, pp. 43–50.

[29] N. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel, "Synthesizing nonanomalous event-based controllers for liveness goals," *ACM TSEM*, vol. 22, no. 1, pp. 9:1–9:36, 2013.

[30] A. Pnueli, "In transition from global to modular temporal reasoning about programs," in *Logics and models of concurrent systems*, 1985, pp. 123–144.

[31] S. Owicki and L. Lamport, "Proving liveness properties of concurrent programs," *TOPLAS*, vol. 4, no. 3, pp. 455–495, 1982.

[32] K. L. McMillan, "Circular compositional reasoning about liveness," in *CHARME*, 1999, pp. 342–346.

[33] P. T. M. Varghese, "Parity and generalized Büchi automata," Ph.D. dissertation, Univ. of Liverpool, 2014.

[34] W. Thomas, "On the synthesis of strategies in infinite games," in *STACS*, 1995, pp. 1–13.

[35] R. McNaughton, "Infinite games played on finite graphs," *Ann. Pure & Appl. Logic*, vol. 65, no. 2, pp. 149–184, 1993.

[36] W. Thomas, "Solution of Church's problem: A tutorial," *New Perspectives on Games and interaction*, vol. 5, 2008.

[37] E. A. Emerson and C.-L. Lei, "Model checking in the propositional μ-calculus," University of Texas at Austin, Tech. Rep. 6, 1986.

[38] A. Pnueli and U. Klein, "Synthesis of programs from temporal property specifications," in *MEMOCODE*, 2009, pp. 1–7.

[39] R. Alur, S. L. Torre, and P. Madhusudan, "Modular strategies for recursive game graphs," *TCS*, vol. 354, no. 2, pp. 230–249, 2006.

[40] A. Banerjee and P. Dasgupta, "The open family of temporal logics: Annotating temporal operators with input constraints," *TODAES*, vol. 10, no. 3, pp. 492–522, 2005.

[41] "Contract construction implementation," 2015. [Online]. Available: https://github.com/johnyf/contract_maker

[42] "`omega`: Symbolic algorithms for solving games of infinite duration (PYTHON package)." [Online]. Available: https://github.com/johnyf/omega

[43] I. Filippidis, R. M. Murray, and G. J. Holzmann, "A multi-paradigm language for reactive synthesis," in *(SYNT'15)*, ser. EPTCS, vol. 202, 2016, pp. 73–97.

[44] "`dd`: Decision diagrams (PYTHON package)." [Online]. Available: https://github.com/johnyf/dd

[45] F. Somenzi, "CUDD: CU Decision Diagram package - v2.5.0," *Colorado University at Boulder*, 2012.

[46] I. Moir, A. Seabridge, and M. Jukes, *Civil avionics systems*, 2nd ed. Wiley, 2013.

[47] F. Boniol, V. Wiels, Y. Ait Ameur, and K.-D. Schewe, Eds., *ABZ 2014: The landing gear case study*, 2014.

[48] M. Kloetzer, S. Itani, S. Birch, and C. Belta, "On the need for communication in distributed implementations of LTL motion specifications," in *ICRA*, 2010, pp. 4451–4456.

[49] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-UAV mission planning," *IJRNC*, vol. 21, no. 12, pp. 1372–1395, 2011.

[50] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *TAC*, vol. 57, no. 11, pp. 2817–2830, 2012.

[51] N. Özay, U. Topcu, and R. M. Murray, "Distributed power allocation for vehicle management systems," in *CDC*, 2011, pp. 4841–4848.

[52] R. Könighofer, G. Hofferek, and R. Bloem, "Debugging formal specifications: A practical approach using model-based diagnosis and counterstrategies," *STTT*, vol. 15, no. 5–6, pp. 563–583, 2013.

[53] V. Raman and H. Kress-Gazit, "Explaining impossible high-level robot behaviors," *TRO*, vol. 29, no. 1, pp. 94–104, 2013.

[54] G. E. Fainekos, "Revising temporal logic specifications for motion planning," in *ICRA*, 2011.

[55] R. W. Floyd, "Assigning meanings to programs," in *Symposia in Applied Mathematics*, vol. 19, 1967, pp. 19–32.

[56] C. Hoare, "An axiomatic basis for computer programming," *CACM*, vol. 12, no. 10, pp. 576–580, 1969.

[57] L. Lamport, "The "Hoare logic" of concurrent programs," *Acta Informatica*, vol. 14, pp. 21–37, 1980.

[58] N. Francez and A. Pnueli, "A proof method for cyclic programs," *Acta Informatica*, vol. 9, pp. 133–157, 1978.

[59] J. Misra and K. Chandy, "Proofs of networks of processes," *TSE*, vol. 7, no. 4, pp. 417–426, 1981.

[60] M. Abadi and L. Lamport, "The existence of refinement mappings," *TCS*, vol. 82, no. 2, pp. 253–284, 1991.