Submitted, 2017 Conference on Decision and Control
http://www.cds.caltech.edu/~murray/papers/fm17-cdc_s.pdf

Hiding variables when decomposing specifications into GR(1) contracts

Ioannis Filippidis and Richard M. Murray

***Abstract*— We propose a method for eliminating variables from component specifications during the decomposition of GR(1) properties into contracts. The variables that can be eliminated are identified by parameterizing the communication architecture to investigate the dependence of realizability on the availability of information. We prove that the selected variables can be hidden from other components, while still expressing the resulting specification as a game with full information with respect to the remaining variables. The values of other variables need not be known all the time, so we hide them for part of the time, thus reducing the amount of information that needs to be communicated between components. We improve on our previous results on algorithmic decomposition of GR(1) properties, and prove existence of decompositions in the full information case. We use semantic methods of computation based on binary decision diagrams. To recover the constructed specifications so that humans can read them, we implement exact symbolic minimal covering over the lattice of integer orthotopes, thus deriving minimal formulae in disjunctive normal form over integer variable intervals.**

I. INTRODUCTION

Before we build a system, usually we make a design. Specifying a system rarely happens as one step. Starting from a coarse problem description, it is iteratively divided into smaller problems that are more detailed, but also more specific and local in nature. This decomposition involves distributing functionality among components, and creating interfaces between them. Part of the reason that we decompose is to *focus* and *isolate*. We want to focus on each subsystem separately, and isolate the reasoning about one subsystem from another [1], to the extent that their dependency on each other allows. So, we want to omit details about other components. A component's behavior is represented mathematically by variables, so this means that we want to eliminate external variables from the specification of any given component. Why did we write those variables in the first place, if we were going to invest effort in omitting them? Because we can more easily express what behavior we want from a collection of components by letting ourselves refer freely to the variables that model them.

In this paper, we study the problem of eliminating variables during the decomposition of specifications into assume-guarantee contracts between components. In order to detect which variables each component needs to know about, we use parameters that prescribe whether each variable is hidden or not. This can be regarded as parameterizing the information communicated between components. This allows computing whether an assume-guarantee contract of the form we studied in [2] exists, as a function of the communication architecture. A specific architecture is then selected, and the properties decomposed. We prove that variables selected to be hidden can be eliminated from the resulting specifications, while still writing them without temporal quantifiers [3]. Thus, we hide information without creating games with partial information at lower levels. This is motivated by the computational hardness that partial information introduces [4], [5], [6], even for linear temporal logic specifications in the GR(1) fragment [7]. A control function that implements a realizable GR(1) specification can be constructed in time polynomial in the number of states (which is exponential in the number of variables). We improve on our previous assumption construction algorithm by accounting for candidate assumptions of a component during the construction of guarantess for the components that will ensure those assumptions, thus computing GR(1) decompositions in cases the previous approach resulted in nesting [2]. In addition, we prove that in presence of full information, the previous approach can be used to always find GR(1) contracts (without nesting).

The idea of using masks as parameters originates from the work of Pnueli, Klein, and Piterman on approximating asynchronous synthesis with GR(1) synthesis [5], [8] using Rosner's reduction. Parameterization of safety and reachability goals, and bounded compositional synthesis with partial information have been studied recently in [9]. The approach proposed here can be viewed as a way to avoid constructing estimators [4] at lower levels, and automatically eliminating variables from the specifications. Synthesis of almost-sure winning sensing strategies in order to avoid synthesis under partial information has been studied in [10].

The paper is organized as follows. Sec. II introduces relevant preliminaries from logic. We develop in Sec. III the parameterization of communication, and elimination of variables, in Sec. IV the computation of assumptions and guarantees for decomposition, improve in Sec. V on previous results in the full information case, and discuss an example in Sec. VI, with conclusions in Sec. VII.

II. OPEN SYSTEMS AND LOGIC

A. Logic and set theory

We use the (raw) temporal logic of actions (TLA$^+$) [3], with some minor modifications that accommodate for a smoother connection to the literature on games. TLA$^+$ is based on Zermelo-Fraenkel (ZF) set theory, which is regarded as a foundation for mathematics. Every entity in TLA$^+$ is a set (also called a "value"). A function f is a set with the property that, for every $x \in \text{DOMAIN}\, f$, we know

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

what value $f[x]$ is. Functions can be defined with the syntax $f \triangleq [x \in S \mapsto e]$, where e some suitable expression [3, p.303, p.71]. For any $x \notin \text{DOMAIN}\, f$, $f[x]$ is some value, unspecified by the axioms of TLA⁺. The collection of functions with domain S and range $R \subseteq T$ forms a set, denoted by $[S \rightarrow T]$. An operator $g(x)$ is a mapping, defined by some expression, with no domain specified. Brackets instead of parentheses distinguish a function from an operator. Unnamed operators are built with the construct LAMBDA [11], which we abbreviate by λ (as in λ-calculus). The notation $F(x, G(\_))$ denotes an operator F that takes as arguments a nullary operator x and a unary operator G [3, §17.1.1]. Nat denotes the set of natural numbers [3, §18.6, p.348], and $0..n \triangleq \{i \in Nat : 0 \leq i \wedge i \leq n\}$. A function with domain $1..n$ for some $n \in Nat$ is called a tuple and denoted with angle brackets, for example $\langle a, b \rangle$. Quantification can be bounded, as in the formula $\forall\, x \in S : P(x)$, or unbounded, as in $\forall\, x : P(x)$. The former is defined in terms of the latter as $\forall\, x \in S : P(x) \;\triangleq\; \forall\, x : (x \in S) \Rightarrow P(x)$. So the "bound" is essentially a suitable antecedent. The operator $\wedge$ denotes conjunction, $\vee$ disjunction, $\neg$ negation. Substitution of expression e_1 for (a rigid or flexible) variable x in the expression e is expressed with $\text{LET}\; x \;\triangleq\; e_1 \;\text{IN}\;\; e$.

Game solving involves reasoning about sets of states. Symbolic methods using binary decision diagrams (BDDs) [12] are used for compactly representing sets of states, instead of enumeration. To use BDDs for specifications in untyped logic we need to identify relevant (integer) values from type invariants that appear in the formulas. This information is used as type hints to automatically rewrite the problem in terms of new variables so that all relevant values be Boolean (instead of integer), thus enabling use of BDDs. This process is called bitblasting and is analogous to program compilation. More details can be found in [13].

Semantics of modal logic: Temporal logic serves for reasoning about dynamics, because it is interpreted over sequences of states. A *state* s is an assignment of values to *all* variables. A *step* is a pair of states $\langle s_1, s_2 \rangle$, and a *behavior* σ is an infinite sequence of states, i.e., a function from $\mathbb{N}$ to states. An *action* (*state predicate*) is a Boolean-valued formula over steps (states). Given a step $\langle s_1, s_2 \rangle$, the formulae x and x' denote the values $s_1[\![x]\!]$ and $s_2[\![x]\!]$, respectively. We will use the temporal operators: $\Box$ "always" and $\Diamond$ "eventually". If formula f is TRUE in every (some) state of behavior σ, then $\sigma \models \Box f$ ($\sigma \models \Diamond f$) is TRUE. Formal semantics are defined in [3, §16.2.4]. If a property φ cannot distinguish between two behaviors that differ only by repetition of states, then φ is called *stutter-invariant*. If for any behavior σ, $\sigma \not\models \varphi$ implies that $\exists\, n \in Nat$ such that the finite subsequence $front(\sigma, n) \triangleq [i \in 0..n \mapsto \sigma[i]]$ cannot be extended to satisfy φ, then φ is a *safety* property. If for any behavior $\sigma \not\models \varphi$ implies that $\forall\, n \in Nat$: $front(\sigma, n)$ can be extended to satisfy φ, then φ is a *liveness* property. A property of the form $\Box\Diamond p$ ($\Diamond\Box p$) is called *recurrence* (*persistence*). Due to TLA⁺ semantics, formulae define collections of sets that are not sets (they are proper classes). For simplicity, we still discuss about "sets" of states. Restricting attention to only variables declared in a given specification leads to this viewpoint too. Different components can observe different variables, giving rise to local state. This is simpler to discuss using suitable state predicates, instead of assignments to subsets of variables.

B. Assume-guarantee properties and contracts

An open system is one constrained with respect to variables it does not control. We use *component* and *player* interchangeably. Specifying a component should not constrain its environment. This is expressible with an *assume-guarantee* formula that spreads implication incrementally over a behavior [14], [15], [16]. We use the form [2]

$$\begin{aligned} &AsmGrnt(Init, EnvNext, SysNext, Liveness) \;\triangleq \\ &\quad Init \Rightarrow \wedge\; \Box\big(Earlier(EnvNext) \Rightarrow SysNext\big) \\ &\qquad\quad\;\; \wedge\; (\Box EnvNext) \Rightarrow Liveness \end{aligned} \tag{1}$$

which is a modification of [16] to avoid circularity. *Earlier* abbreviates the past LTL operators *WeakPrevious* and *Historically* composed (expressible in TLA⁺ using temporal quantification and history variables)–its definition can be omitted here.

Synthesis of implementations: A temporal property can be less specific than the final implementation. Synthesis is the algorithmic construction of a controller that ensures the component satifies a given temporal property. Let x be uncontrolled and y controlled variables, and

$$\begin{aligned} &Realization(f, g, mem_0) \;\triangleq \\ &\quad (mem = mem_0) \wedge \Box \wedge\; y' = f[\langle mem, x, y \rangle] \\ &\qquad\qquad\qquad\qquad\quad \wedge\; mem' = g[\langle mem, x, y \rangle] \end{aligned}$$

where *IsFiniteSet* requires finite cardinality [3, p.341]. Assume that φ is a temporal logic formula over variables x, y but not the memory variable mem. An implementation (program, circuit, etc.) of φ is a pair $\langle f, g \rangle$ of a controller function f and a memory update function g, and an initial memory value mem_0, such that $\models \wedge\; IsAFunction(f) \wedge IsFiniteSet(\text{DOMAIN}\, f)$ ($\models$ means
$\wedge\; IsAFunction(g) \wedge IsFiniteSet(\text{DOMAIN}\; g)$
$\wedge\; Realization(f, g, mem_0) \Rightarrow \varphi$
TRUE for all behaviors σ). We denote existence of an implementation with $IsRealizable(j, \varphi)$, where j identifies the component (so x, y). A formal TLA⁺ definition is possible [17], [18] but the above suffices here.

A formula of the form $\bigvee_{j \,\in\, 1..m} \Diamond\Box P_j \vee \bigwedge_{i \,\in\, 1..n} \Box\Diamond R_i$ describes a liveness property categorized as generalized Streett(1), or GR(1) [7]. Synthesis for φ as in Eq. (1) with GR(1) *Liveness* has complexity polynomial in the number of (relevant) states, thus GR(1) is preferred to GR(k) or LTL.

Contracts: We will be discussing the decomposition of a temporal property φ into multiple properties that are realizable by a collection of components. A *contract* is a partition of variables among players in $1..n$ and a collection of properties $\psi_1, .., \psi_n$ such that [2]

$$\models \bigwedge\nolimits_{j \in 1..n} IsRealizable(j, \psi_j) \wedge \big((\bigwedge\nolimits_{j \in 1..n} \psi_j) \;\Rightarrow\; \varphi\big)$$

In other words, a contract is a collection of assume-guarantee properties for each component that are realizable and conjoined imply the desired behavior for the system assembled from those components. The index j is notation in the metatheory, *not* within the object theory TLA+ [19].

The specifications we consider are interleaving, in that they allow only variables of one component to change in each step (a "move"). Components move in a fixed order that repeats (a turn-based game).

Elements of synthesis algorithms: Computing attractors is fundamental to reasoning about open systems. An attractor is the set of states from where a controller exists that can guide the system to a desired set of states. It answers the "multi-step" control problem, and can be computed as the least fixpoint of an operation that involves the controllable predecessor operator *CPre*, which answers the "one-step" control problem [20]. In a turn-based game with full information, *CPre* in the system's turn is

$$\begin{aligned} SysPr(x, y) &\triangleq \exists\, \hat{y} \,:\, SysNxt(x, y, \hat{y}) \wedge Target(x, \hat{y}) \\ EnvPr(x, y) &\triangleq \forall\, \hat{x} \,:\, EnvNxt(x, y, \hat{x}) \Rightarrow Target(\hat{x}, y) \end{aligned} \tag{2}$$

A state satisfies *CPre* if x, y take values in that state such that the system can choose a next move $\hat{y}$ allowed by $SysNext$, and any next environment move $\hat{x}$ that $EnvNext$ allows leads to the given $Target$ states.

III. HIDING VARIABLES

A. Motivation and overview

A hierarchical approach that omits details at the high level is one way to reduce the computational cost of decomposition. After decomposition, we can introduce the omitted details at lower levels. Clearly, those details should be irrelevant to the higher level design. To prefer this approach, the subproblems must be simpler and the synthesized specifications and understandable by humans. In this section we eliminate variables that leave component realizability unaffected, to simplify the subproblems. For variables that cannot be entirely eliminated because they provide useful information at some phases of execution, we apply the same method to eliminate them locally, depending on the current objective, and so reduce how much information needs to be communicated between components at any single time. We also implement symbolic minimal covering to simplify the generated contract specifications so that humans can read them, as described later. Knowing a variable's value is information. Quantification *hides* a variable's value, destroying some information. If a component specification's implementation can work without knowing a variable's value, then that value doesn't need to be communicated at runtime, motivating the following.

Problem 1 (Hiding variables): Given a realizable assume-guarantee GR(1) formula φ with one recurrence pair (memory embedded in state space), and a finite collection of component variables, which environment variables can be universally quantified (hidden from the component) without preventing the component from realizing φ?

A temporal formula describes how its free variables behave. Hidden variables are bound by quantifiers, but they still need to be reasoned about, unless entirely eliminated from a (sub)formula. We want the decomposition step to remove irrelevant details, in order to create space for adding relevant finer details to each component's specification. In other words, simplify before adding more, motivating the following.

Problem 2 (Expressibility): Can we write the component specifications so that hidden variables not occur in the resulting formulas, and define an assume-guarantee GR(1) property without hidden variables (thus with full information wrt the variables declared in the component specification)?

B. Shared invariant as safety assumption

We will quantify variables universally to hide them. In this section x, h denote uncontrolled (environment) variables and y controlled (component) variables. Suppose $P(x, y, h)$ is a predicate where we want to hide variable h. If we use unbounded quantification, $\forall\, h \,:\, P(x, y, h)$, then in most cases the result will be impractically restrictive (too conservative), or FALSE. We should instead introduce a state predicate as antecedent to bound h. For that we use a global ("shared") invariant, as is common practice in reasoning about distributed algorithms [21]. The weakest invariant that works in the full information case is the collection of cooperatively winning states [2], [22], and is unique. Let $Inv(x, y, h)$ denote this invariant, computed as the states from where a centralized player would have a winning control strategy [2, §III-A]. Let us introduce this invariant as an antecedent, in order to bound the variable x that is a candidate for hiding.

C. Hiding variables

A state predicate that at a higher level appears as an antecedent (assumption) in an open system (assume-guarantee) property can at a lower level be integrated into the environment and system action formulas of the open system property at that lower level. The variable i tracks the player whose turn it is to move, thus expressing the modeling assumption of interleaving. For readability below i is omitted. Finer detail can be found in [13] and implementation code. The component can safely step from a state $s_1 \triangleq [x \mapsto a, y \mapsto b, h \mapsto c]$ to a state $s_2 \triangleq [x \mapsto a, y \mapsto y', h \mapsto c]$ that satisfies the predicate $Target(x, y', h)$ without observing variable h if

$$\begin{aligned} SysCPre(x, y) \;\triangleq\;& \exists\, y' \,:\, \forall\, h \,: \\ & Inv(x, y, h) \;\Rightarrow\; \wedge\; SysNext(x, y, h, y') \\ & \qquad\qquad\qquad\quad \wedge\; Target(x, y', h) \end{aligned} \tag{3}$$

Writing $\exists\, y'$ is abuse of TLA+ syntax in the metatheory, for readability. (In our case $SysNext$ implies $Inv(x, y', h)$ from taking cooperative closure.) The variable h is hidden due to the (linear in first-order logic) order of quantifiers that forces the choice of y' unknowing of h. The component can be sure that the environment is necessarily stepping from state s_1 to a state $s_3 \triangleq [x \mapsto x', y \mapsto b, h \mapsto h']$ that satisfies the predicate $\mathit{Target}(x', y, h')$ if

$$\begin{aligned}\mathit{EnvCPre}(x, y) \;\triangleq\; & \forall\, x', h' : \forall\, h : \\ & \mathit{Target}(x', y, h') \vee \neg \wedge \mathit{Inv}(x, y, h) \\ & \qquad\qquad\qquad\quad \wedge \mathit{EnvNext}(x, y, h, x', h')\end{aligned} \tag{4}$$

(In our case $\mathit{EnvNext}$ implies $\mathit{Inv}(x', y, h')$ from taking cooperative closure.) If the objective $\mathit{Target}(x, y, h)$ is independent of variable h, then we can arrange Eq. (3) as

$$\begin{aligned}\lambda x, y : \exists\, y' : \wedge\, \forall\, h : & \vee \neg\mathit{Inv}(x, y, h) \\ & \vee \mathit{SysNext}(x, y, h, y') \\ \wedge\, & \mathit{Target}(x, y')\end{aligned} \tag{5}$$

and Eq. (4) as

$$\begin{aligned}\lambda x, y : \forall\, x' : \vee \neg \exists\, h, h' : & \wedge \mathit{Inv}(x, y, h) \\ & \wedge \mathit{EnvNext}(x, y, h, x', h') \\ \vee\, \mathit{Target}(x', y) &\end{aligned} \tag{6}$$

D. Parameterizing the communication architecture

Which variables can we hide (Sec. III-C) without sacrificing realizability? Instead of enumerating the combinations of variables that can be hidden, we parameterize which variables are hidden or not. For each variable, a *mask* variable m is introduced that "routes" the variable to take a visible or hidden value $\mathit{Mask}(m, v, h) \triangleq \text{IF } (m = \text{TRUE}) \text{ THEN } h \text{ ELSE } v$ We can thus model the availability or lack of information. Substitute this selector expression for the tentatively hidden variable in Eq. (3) to obtain

$$\begin{aligned}\lambda x, y, v, m : \exists\, y' : \forall\, h : & \text{ LET } r \triangleq \mathit{Mask}(m, v, h) \\ & \text{ IN } \vee \neg\mathit{Inv}(x, y, r) \\ & \qquad \vee \wedge \mathit{SysNext}(x, y, r, y') \\ & \qquad\quad \wedge \mathit{Target}(x, y', r, m)\end{aligned} \tag{7}$$

The idea of using masks as parameters originates from work on approximating asynchronous synthesis with GR(1) synthesis [5], [8].

$$\begin{aligned}& \lambda x, y, v, m : \forall\, h, h', x' : \\ & \quad \text{LET } r \triangleq \mathit{Mask}(m, v, h) \\ & \quad \text{IN } \vee \neg \wedge \mathit{Inv}(x, y, r) \\ & \qquad\qquad \wedge \mathit{EnvNext}(x, y, r, x', h') \\ & \qquad \vee \mathit{Target}(x', y, h', m)\end{aligned} \tag{8}$$

The previous formulas are from the viewpoint of the component selected as the system. Similar formulas apply from the viewpoint of a component that is part of the environment above, but with a different mask variable in place of m. A different mask is associated with each variable communicated between each pair of components. These masks increase the number of Boolean variables in the symbolic computation, but are not quantified during controllable predecessor operations, and are Boolean-valued, whereas the variables they mask are integer-valued. With n components and k (integer-valued) variables in total (over all components), $(n-1)k$ Boolean mask variables are introduced. These are parameters, so the number of reachable states remains unchanged, and thus the same number of controllable predecessor operations will be applied, and realizability fixpoints take the same number of iterations, similarly to arguments developed for parameterized synthesis [9]. The number of components n involved in each decomposition step is expected to not be large, so that the design specification be understandable by a human.

With this formulation, we can parameterize what information is available to each player. The masks parameterize the communication architecture, and allow for computing symbolically those architectures that allow for decomposing the high-level specification into a contract. We can think of the above scheme as a sensitivity analysis of the problem with respect to the communication architecture.

Can we eliminate hidden variables from component specifications? This requires writing the system and environment actions that are computed for each component with formulas that do not contain quantification. The eliminated variables do not appear in the component specification, so further work focusing on that component can be carried out in a full information context, including GR(1) synthesis. (The usual controllable predecessor operator can be used.)

Provided that goals in component specifications are expressed without mentioning any hidden variables (as described later), Eqs. (5) and (6) imply that if the actions $\mathit{SimplerSysNext}, \mathit{SimplerEnvNext}$ satisfy the axioms

$$\begin{aligned}& \models \mathit{SimplerSysNext}(x, y, y') \equiv \\ & \quad \forall\, h : \mathit{Inv}(x, y, h) \Rightarrow \mathit{SysNext}(x, y, h, y') \\ & \models \mathit{SimplerEnvNext}(x, y, x') \equiv \\ & \quad \exists\, h, h' : \mathit{Inv}(x, y, h) \wedge \mathit{EnvNext}(x, y, h, x', h')\end{aligned}$$

then these actions can be used in the full information controllable predecessor, which in system and environment turns takes the forms

$$\begin{aligned}& \lambda x, y : \exists\, y' : \mathit{SimplerSysNext}(x, y, y') \wedge \mathit{Target}(x, y') \\ & \lambda x, y : \forall\, x' : \mathit{SimplerEnvNext}(x, y, x') \Rightarrow \mathit{Target}(x', y)\end{aligned}$$

to obtain the same realizability and synthesis results with Eqs. (5) and (6). This elimination enables further reasoning for each component to be carried out as if the component had full information about its environment. We implemented the exact symbolic minimization method using branch and bound as described in [23], but for the lattice of integer orthotopes (hypperrectangles aligned to axes), instead of the Boolean lattice. Using this approach, we express the BDDs of $\mathit{SimplerSysNext}$ and $\mathit{SimplerEnvNext}$ as formulas over integer variables. Independence from the hidden variable h implies that only variables x, y, x', y' will occur in the resulting formulas. Due to lack of space, the details of the minimization will appear elsewhere. The implementation is released in the package `omega` [24]. The Quine-McCluskey minimization method, which takes exponential amount of space and time, thus is impractical, has been used before for simplifying Boolean logic expressions in manuals [25], robot path planning among planar rectangles [26] and recently for simplifying enumerated robot controllers [27].

IV. DECOMPOSITION INTO A GR(1) CONTRACT

The decomposition algorithm works by finding recurrence assumptions ($\Box\Diamond$) for a player and corresponding guarantees for the other players, starting from a goal and reasoning backwards. It is based on our algorithm for constructing contracts in the presence of full information [2, Alg.1], with mainly the following differences. The controllable predecessor (CPre), fixpoint and other computations are parameterized with respect to the communication between the components, as described below and in Sec. III. Due to the different information available to each component, observability has to be taken into account, and an assumption (by one player) that corresponds to a guarantee (by another player) are each expressed using only variables visible to each player. Another difference is the construction of assumptions, which is described next.

The earlier algorithm [2, Alg.1] works as follows. Given a goal g for player j, the attractor $a \triangleq Attr_j(g)$ (states from where j can guide the system to g) is computed, and then the attractor $b \triangleq Attr_k(a)$, where k a player other than j. The next step is to find the control invariant subset of b from where j can ensure the behavior either remains forever in b, or eventually enters a ($(\Box b) \vee \Diamond a$). This set is called a $Trap_j(b, a)$, and can be empty, even when our intuition as specifiers would suggest otherwise, for the following reason. States from where player k can reach a may exist (b non-empty), but from all of them k may be able to exit b, reaching states from where j can prevent k from reaching a, by j going "backwards". However, this backward motion would lead to j satisfying the assumption we are about to construct, which is what the specifier's intuition suggests. The algorithm we describe next takes this into account, thus succeeding in finding assumptions in cases the earlier approach could not (earlier, those resulted in a recursion, where k became the "main" player from states not in b).

Following the transitions out of the "basin" b we can enlarge the basin, and use this as a *temporary* safety assumption made by player k about player j remaining in the enlarged basin. This enlargement is shown in Fig. 1 and is computed in procedure MAKEPINFOASSUMPTION in Algo. 3. In our case the decomposition works recursively with respect to teams of players, starting with a player j and the rest in the set $Team$ playing as if they were one player moving in multiple turns. Decomposition progresses by successively removing players from a team, creating an acyclic graph of assumption dependencies. Below we focus on the construction of each assumption. More details can be found in an appendix [13] and an implementation [28].

The main computation of assumptions that a $Player$ can make, and guarantees for other players in a $Team$ that imply the assumptions is Algo. 3, shown schematically in Fig. 2. Spc stores information about the specification (player actions, observer used to index masks, visible players (unmasked)). This algorithm first attempts to find assumptions as described above. If a trap set does not exist, it enlarges the $basin$ by the set of states $escape$ to where the $Team$ of players can force an exit in one step from $basin$ (Fig. 1). The result is a larger $basin$ (Fig. 2). Player j attempts to keep the behavior within the observable subset $Obs_j(basin)$, unless it exists to a, from where j can ensure the behavior reaches the observable subset of the goal $Obs_j(g)$ ($Obs(Goal, Player, Spc)$ in Algo. 3). We are interested to ensure that if player j does not exit $Obs_j(basin)$, then the $Team$ can force a visit to some state in $G = ProjHiddenVrs_{team}(a) = \neg Obs_{team}(\neg a)$ where the hidden variables are with respect to the $Team$. This set is the attractor $\eta_{team} = PinfoAttr_{team}(G, TmpSpc)$ with parameterized communication and temporary safety assumption in $TmpSpc$. Note that the objective is weakened, because the $Team$ does not need to observe whether a has indeed been reached, player j is already doing so. Moreover, a depends on variables of j that need not be necessary information for $Team$ to accomplish its objective. Using $Obs_{team}(a)$ instead of $\neg Obs_{team}(\neg a)$ would require estimating the values of several player j variables (Obs is defined below).

The set $\neg\eta_{team}$ is the candidate recurrence guarantee for the $Team$. Player j will attempt to remain in $b_j = Obs_j(\eta_{team})$, which is a subset of $Obs_j(basin)$, thus the temporary safety assumption of the $Team$ will be implied if player j has a trap. The $Trap_j(b_j, a, Spc)$ contains those states from where player j can force a visit to a, or remain forever in b_j. We can prove that the latter will not be the case, provided that the $Team$ does satisfy its guarantee to reach $\neg\eta_{team}$, which is realizable by the $Team$ because η_{team} was computed as the states from where the $Team$ can force a visit to G or force an exit from $basin$ ($\models \eta_{team} \wedge G = \text{FALSE}$). We have outlined the proof of the following soundness result.

Theorem 3 (Soundness): For each assignment of values to parameters (which corresponds to a communication architecture) in the results of Algo. 3, if a recurrence goal $\Box\Diamond\neg\eta_{team}$ for the team and a persistence goal $\Diamond\Box r$ for the component have been found, then the recurrence goal is realizable by the team, the (progress) goal $(\Diamond\Box r) \vee (\Diamond a)$ realizable by the player, and $(\Box\Diamond\neg\eta_{team}) \Rightarrow \neg\Diamond\Box r$.

This theorem ensures that the player cannot prevent the team from reaching G, neither stay forever in P, but can ensure to reach S, if forced to exit P, which is what happens as the team moves towards G. Therefore, progress of the team can be utilized by the component to progress itself towards its own recurrence objectives. Conversely, if the team violates $\Box\Diamond G$, then it cannot force an exit from P in any way other than via S. The above algorithm is iterated to find a (parameterized) greatest fixpoint. Multiple recurrence goals for player j can be accounted for by conjunction of the results from applying the algorithm to each goal. This fixpoint nesting is arranged similarly to GR(1) synthesis [7], with the difference that there are three alternations of least and greatest fixpoints, so worst-case complexity of symbolic implementation is a number of controllable predecessor calls cubic in the number of states. In practice the latter number is the diameter of the state space (the farthest two states can be apart in number of transitions). It is interesting to observe

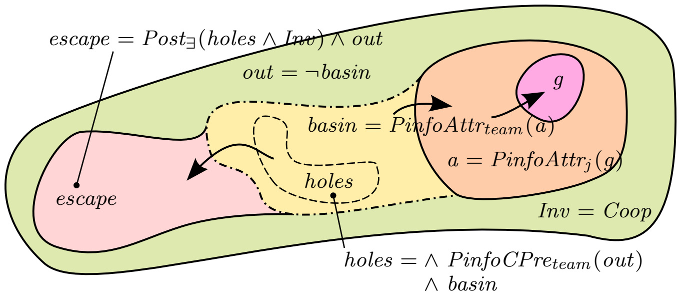
$escape = Post_\exists(holes \wedge Inv) \wedge out$
$out = \neg basin$
g
$basin = PinfoAttr_{team}(a)$
$a = PinfoAttr_j(g)$
$holes$
$escape$
$Inv = Coop$
$holes = \wedge\ PinfoCPre_{team}(out)$
$\wedge\ basin$

Fig. 1: Collecting escapes that can cause a trap set to not exist (simplified compared to Algo. 3).

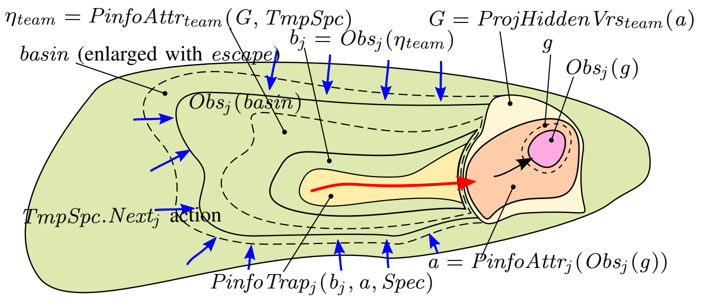
$\eta_{team} = PinfoAttr_{team}(G, TmpSpc)$
$G = ProjHiddenVrs_{team}(a)$
$b_j = Obs_j(\eta_{team})$
$basin$ (enlarged with $escape$)
g
$Obs_j(g)$
$Obs_j(basin)$
$TmpSpc.Next_j$ action
$a = PinfoAttr_j(Obs_j(g))$
$PinfoTrap_j(b_j, a, Spec)$

Fig. 2: Finding assumptions by accounting for observability, and collecting escapes to form auxiliary safety constraints that the environment assumes, which represent that player j will attempt to satisfy its persistence constraint.

that the temporary safety assumption has an objective similar to notions used in modified logic semantics and fixpoint computations of cooperative reactive synthesis [29].

As soon as a trap set is found for some parameter values, those are "frozen" in further iterations, using the variable *converged* to accumulate those values. Regarding convergence we have the following.

Theorem 4 (Termination): ASSUME : Finite number of states. PROVE : Algo. 3 terminates in a finite number of iterations.

PROOF SKETCH: CASE $\models escape = \text{FALSE}$: satisfies the loop guard. CASE $\models escape \neq \text{FALSE}$: $escape = out \wedge \ldots$ so $\models escape \Rightarrow \neg basin$. Thus $basin \vee escape$ is strictly larger than $basin$. So $basin$ increases strictly as long as $escape$ is not equiv. to FALSE. $basin$ is bounded above by the number of states, thus the loop terminates.

In order for variable elimination to leave the winning set computations unaffected, the $Target$ in Eq. (5) should not depend on h. This is ensured for the current player with a parameterized static observer (mentioned above as Obs_j) that assumes the shared invariant $Inv\ Obs(Target(\_, \_, \_, \_)) \triangleq \forall h : \text{LET } r \triangleq Mask(m, v, h) \text{ IN } Inv(r, y, i) \Rightarrow Target(r, y, m, i)$. This ensures that specifications of the current player will be expressible using only visible variables, when we choose a communication architecture, i.e., after the parameteric contract construction has been completed. Static observation is similar to positional estimation [4]. In our case the observers are parameterized by the communication architecture.

Formally, parameters are TLA$^+$ constants, also known as *rigid variables* [3]. Parameterization has a "static" effect: CPre quantifies only *flexible* variables, so the number of quantified variables remains unchanged, and the diameter of the state space in each "slice" obtained by fixing parameter values is no larger than the cooperatively reachable state space with full information. Similar observations apply to parameterized goal set computations [9].

V. EXISTENCE OF GR(1) CONTRACTS IN THE PRESENCE OF FULL INFORMATION

In previous work [2], we showed that a recurrence property of the form $\Box\rho_1 \wedge \Box\Diamond G$ for player 1 and safety $\Box\rho_i$ for other

Fig. 3: Algorithm for finding traps using temporary safety assumptions with parameterized information availability.

```
def MAKEPINFOASSUMPTION($Goal, Player, Team, Spc$) :
    $j := Turn(Player),\ jp := (j+1)\%Spc.n$
    $Spc.visible := \{Player\},\ Spc.observer := Player$
    $TeamSpc := Copy(Spc),\ TeamSpc.visible := Team$
    $TeamSpc.observer :=$ PICK $p : p \in Team$
    $a := PinfoAttr(Obs(Goal, Player, Spc), Player)$
    $basin := PinfoAttr(A, Team, TeamSpc)$
    $r, \_, basin :=$ MAKEPERSISTENCEASM(
        $a, basin, Player, Team, Spc, TeamSpc$)
    $escape :=$ TRUE, $converged :=$ FALSE
    while $(\neg \models escape = \text{FALSE})$ :
        $out := ProjHiddenVrs(\neg basin \wedge Inv, TeamSpc)$
        $holes := basin \wedge PinfoCPre(out, Team, TeamSpc)$
        $escape := out \wedge Post_\exists(holes \wedge Inv)$
        $escape := \wedge\ ProjHiddenVrs(escape, TeamSpc)$
                   $\wedge\ out \wedge \neg converged$
        (* Enlarge *)
        $basin := basin \vee escape$
        $obs\_basin := Obs(basin, Player, Spc)$
        $stay := (\text{LET } i \triangleq (j+1)\%Spc.n$
            $x_j \triangleq x'_j \text{ IN } obs\_basin)$
          $\wedge (\text{LET } i \triangleq j \text{ IN } obs\_basin)$
        $TmpSpec := Copy(Spec)$
        (* Temporary safety assumption restricts $Player$ action $Next_j$ *)
        $TmpSpec.Next_j := TmpSpec.Next_j \wedge stay$
        $r, \eta_{team}, b_{team} :=$ MAKEPERSISTENCEASM(
          $a, basin, Player, Team, Spc, TmpSpc$)
        $converged := converged \vee NonEmptySlices(r)$
    return $a, r, \eta_{team}, TmpSpc$
def MAKEPERSISTENCEASM(
        $a, basin, Player, Team, Spc, TmpSpc$) :
    $G := ProjHiddenVrs(a, TmpSpc)$
    $b_{team} := basin \wedge PinfoAttr(G, Team, TmpSpc)$
    $\eta_{team} := b_{team} \wedge \neg G$
    $b_j := Obs_j(\eta_{team}, Player, Spc)$
    $r := \neg a \wedge PinfoTrap(b_j, a, \{Player\}, Spc)$
    return $r, \eta_{team}, b_{team}$
```

players ($i \neq 1$) can be decomposed into a contract of assume-guarantee properties with nested GR(1) properties as liveness (for multiple recurrence goals memory can be embedded a priori in the state space, since anyway introduced after synthesis, because searching for memoryless controllers is NP-complete [30]). The motivation for nested GR(1) was to ensure that the conjunction of decomposed properties allows all behaviors that are safe for the original property (the one decomposed). Nesting or some other form of constraints is in general unavoidable for GR(1) decomposition [2, Prop.5].

However, by the decomposition algorithm itself, the additional behaviors preserved by using nested GR(1) properties are unrealizable by the components. This is due to a gap between what property a formula describes and the *realizable part* [31] of that property. We prove below that there are safety constraints which remove only unrealizable behaviors. This shows that the decomposition algorithm of [2] can be applied to always obtain GR(1) contracts in the presence of full information, simplifying our previous results. Thus, it can also be applied recursively, for hierarchical designs. In [2] we proved that refining GR(1) contracts may not exist. According to the definitions of [2], the GR(1) properties below yield contract, but not a refining one, because they restrict property closure. Let $d \in Nat$. We split Def.7 [2] into two, and omit component actions, because they are separate from reasoning about liveness.

Definition 5 (Nested GR(1)): Formulae described by

$$\begin{aligned} \Box \bigwedge_{m=0}^{d} \wedge\ & (P_m \wedge \bigwedge_{k \in m..d, l \in 0..H_k} \Box\Diamond\neg\eta_{kl}) \Rightarrow \Diamond Q_m \\ \wedge\ & Q_m \Rightarrow \bigwedge_{l=0}^{\Xi_m} \Box\Diamond\neg\xi_{ml} \end{aligned}$$

where $P_m, Q_m, \eta_{kl}, \xi_{ml}$ are state predicates.

Definition 6: A *chain* condition is

$$\begin{aligned} \wedge \forall\, m \in 1..d\ :\ \forall\, l \in 0..\Xi_m\ :\ & \wedge\ P_{m-1} \Rightarrow Q_m \\ & \wedge\ \xi_{ml} \Rightarrow \neg P_{m-1} \end{aligned}$$

$$\begin{aligned} \wedge \forall\, m \in 0..d\ :\ & \wedge\ Q_m \Rightarrow P_m \\ & \wedge\ \forall\, l \in 0..H_m\ :\ \eta_{ml} \Rightarrow (P_m \wedge \neg Q_m) \\ & \wedge\ \forall\, l \in 0..\Xi_m\ :\ \xi_{ml} \Rightarrow Q_m \end{aligned}$$

Proposition 7 (Equirealizability): ASSUME : φ is a nested GR(1) formula (Def. 5) that results from Alg.1 of [2] (so $P_m, Q_m, \eta_{kl}, \xi_{ml}$ satisfy Def. 6). PROVE : A controller realizes a property $AsmGrnt(I, E, S, \varphi)$ if, and only if, it realizes $AsmGrnt(I, E, S \wedge Nxt, L)$ where

$$\begin{aligned} Nxt\ &\triangleq\ \bigwedge_{m \in 0..d}((P_m \wedge \neg Q_m) \Rightarrow P'_m) \\ L\ &\triangleq\ \bigwedge_{m \in 0..d} \wedge \vee \bigvee_{k \in m..d, l \in 0..H_k} \Diamond\Box\eta_{kl} \\ &\qquad\qquad \vee\ \Box\Diamond(P_m \Rightarrow Q_m) \\ &\qquad\quad \wedge\ \Box(Q_m \Rightarrow \bigwedge_{l \in 0..\Xi_m} \neg\xi_{ml}) \end{aligned}$$

PROOF SKETCH: By construction from Alg.1 of [2], P_m is the winning set for the objective $\Diamond Q_m \vee \bigvee_l \Diamond\Box\eta_{ml}$. SUFFICES : conjoining the safety constraint does not remove realizable behaviors, the rest follows by simple temporal reasoning. SUFFICES : No state $\neg P_m$ is in the winning set of $\Diamond Q_m \vee \bigvee_{k \in m..d, l \in 0..H_k} \Diamond\Box\neg\eta_{kl}$. Since P_m is the winning set for $\Diamond Q_m \vee \bigvee_{l \in 0..H_m} \Diamond\Box\neg\eta_{ml}$, it suffices to show that there are no other winning states. By construction of the η, the component cannot realize any $\Diamond\Box\eta_{kl}$. It remains to show that the component cannot force the behavior to reach P_m from $\neg P_m$. Suppose it did. The last step that enters P_m must be from a state where it is the environment's turn (otherwise P_m contains that state, because P_m itself is a winning set). All such "environment" states in $\neg P_m$ with a direct step to P_m are contained in $Q_{m-1} \wedge \neg P_{m-1}$. So at any of those states, the environment satisfies all the liveness assumptions η_{kl} (for all k, l) and can take a step that avoids P_m, a contradiction. Thus the winning set is P_m. Therefore, no controller can step from $P_m \wedge \neg Q_m$ to $\neg P_m$, because it cannot ensure a return. It would then violate the objective $\Diamond Q_m$, pending since it stepped out of $P_m \wedge \neg Q_m$.

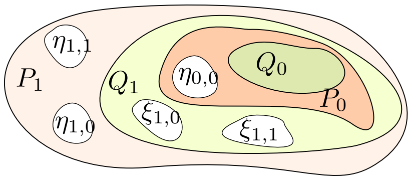

Fig. 4: An example of a chain condition, with states satisfying each predicate depicted by patches. Containment means implication, e.g. $\models Q_0 \Rightarrow P_0$.

A delicate point is that we can use specifications of *other* components to show that if all components implement the contract, then the component will return to P_m, but from a single component's viewpoint, this objective is unrealizable.

VI. Example

We show with an example that the algorithm identifies variables that we would expect to be redundant information. An experimental implementation is available [28] under a BSD-3 license, and implemented using the Python package `omega` [24]. The example concerns the interaction of subsystems in an aircraft with retractable landing gear [2]. Three components are present in this example: an autopilot, a module that controls the landing gear, and another module that controls the doors of the landing gear. Table I lists the variables that model this example's components. The modes are landing ($m = 0$), cruise ($m = 1$), and takeoff ($m = 2$), gears range from retracted ($g = 0$) to fully extended ($g = 5$), and doors from closed ($d = 0$) to fully open. The constraints are a stronger version of those in [24], requiring that the gear shall be extended when in landing mode and below a threshold altitude, retracted in cruise mode, the doors closed above a speed threshold, and doors shall be open if gear not retracted. In addition, the rates at which gear, doors, and speed change are bounded. The shared invariant is satisfied by 55576506 states (with the turn index $i \in 0..2$ included).

The two recurrence goals for the autopilot are to enter cruise and landing mode, $\Box\Diamond(m = 1) \wedge \Box\Diamond(m = 0)$. The parameterized computation for each of these goals is performed with different mask parameters, in order to allow for different communication connectivity. The objectives are chained by an outer fixpoint that ensures repetition is possible. Assumptions and guarantees are found in each case.

TABLE I: Variables in landing gear example

Var	Values	Represents	Controlled by
d	$0..5$	Doors (closed–open)	Door module
g	$0..5$	Gear position (up–down)	Gear module
H	$0..4 \times 10^3$	Altitude	Autopilot
s	$0..10^3$	Airspeed	Autopilot
m	$0..2$	Flight mode	Autopilot

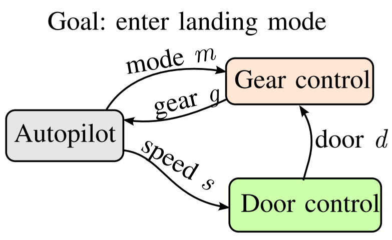

Fig. 5: Communicated variables when goal is landing mode.

To enter landing mode from cruise mode, the autopilot must lower the speed and wait for the doors to open and gear to exted (generated assumption). For the opposite change of mode, the autopilot must first reach above the threshold altitude, where it awaits the gear to retract and doors to close. The resulting communication connectivity when the goal is landing mode is shown in Fig. 5. The gear and doors only need to observe the mode m and speed s, and the autopilot only the gear state g. Compared to a full communication graph (10 signals), only 4 suffice for this goal (and no more for the other goal). Moreover, some variables are entirely eliminated from component specifications. In particular, the door module never needs to receive height or mode information, and the gear never needs speed information. The autopilot does not need to observe the door state.

The states that satisfy the shared invariant Inv and from where the generated assumptions and guarantees ensure that the autopilot can guide the entire system to repeatedly enter both landing mode and cruise mode are

$$\begin{aligned}
&\vee \wedge (door = 0) \wedge (gear = 0) \wedge (height \in 3001..4000)\\
&\quad \wedge (mode \in 1..2)\\
&\vee \wedge (door = 5) \wedge (gear = 5) \wedge (mode = 0)\\
&\quad \wedge (speed \in 0..750)\\
&\vee \wedge (door = 5) \wedge (gear = 5) \wedge (mode = 2)\\
&\quad \wedge (speed \in 0..750)\\
&\vee \wedge (door = 5) \wedge (height \in 3001..4000)\\
&\quad \wedge (mode = 2) \wedge (speed \in 0..750)\\
&\vee \wedge (gear = 0) \wedge (height \in 3001..4000)\\
&\quad \wedge (mode \in 1..2) \wedge (speed \in 0..750)
\end{aligned}$$

These were computed as a minimal cover [23] using our implementation for finding minimal disjunctive normal form formulae with inequalities over integer variables as conjuncts (available in `omega` v0.1.0).

VII. CONCLUSIONS

We parameterized GR(1) contract construction with respect to communication architectures, and showed how variables can be eliminated from component specifications.

Acknowledgments: In distinguishing between syntax and semantics of assume-guarantee properties we benefited from discussions with Necmiye Özay and Scott Livingston. This work was supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA.

REFERENCES

[1] L. Lamport, "Composition: A way to make proofs harder," in *COMPOS*, 1997, pp. 402–423.
[2] I. Filippidis and R. M. Murray, "Symbolic construction of GR(1) contracts for systems with full information," in *ACC*, 2016, pp. 782–789.
[3] L. Lamport, *Specifying systems: The TLA+ language and tools for hardware and software engineers*. Addison-Wesley, 2002.
[4] R. Ehlers and U. Topcu, "Estimator-based reactive synthesis under incomplete information," in *HSCC*, 2015, pp. 249–258.
[5] A. Pnueli and U. Klein, "Synthesis of programs from temporal property specifications," in *MEMOCODE*, 2009, pp. 1–7.
[6] A. Pnueli and R. Rosner, "Distributed reactive systems are hard to synthesize," in *FOCS*, vol. 2, 1990, pp. 746–757.
[7] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *VMCAI*, 2006, pp. 364–380.
[8] U. Klein, N. Piterman, and A. Pnueli, "Effective synthesis of asynchronous systems from GR (1) specifications," in *VMCAI*, 2012, pp. 283–298.
[9] S. Moarref, "Compositional reactive synthesis for multi-agent systems," Ph.D. dissertation, University of Pennsylvania, 2016.
[10] J. Fu and U. Topcu, "Integrating active sensing into reactive synthesis with temporal logic constraints under partial observations," in *ACC*, 2015, pp. 2408–2413.
[11] L. Lamport, "TLA^{+2}: A preliminary guide," Tech. Rep., 15 Jan 2014.
[12] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *TC*, vol. 35, no. 8, pp. 677–691, 1986.
[13] Appendix to this document. [Online]. Available: https://github.com/johnyf/binaries/blob/master/appendix.pdf
[14] M. Abadi and L. Lamport, "Conjoining specifications," *TOPLAS*, vol. 17, no. 3, pp. 507–535, 1995.
[15] B. Jonsson and Y.-K. Tsay, "Assumption/guarantee specifications in linear-time temporal logic," *TCS*, vol. 167, no. 1–2, pp. 47–72, 1996.
[16] U. Klein and A. Pnueli, "Revisiting synthesis of GR(1) specifications," in *HVC*, 2010, pp. 161–181.
[17] I. Filippidis and R. M. Murray, "Formalizing synthesis in TLA$^+$," Caltech, Tech. Rep. CaltechCDSTR:2016.004, 2016.
[18] L. Lamport, "Miscellany," 21 April 1991, sent to TLA mailing list.
[19] S. C. Kleene, *Introduction to metamathematics*. North-Holland, 1971.
[20] L. de Alfaro, T. Henzinger, and F. Y. Mang, "The control of synchronous systems," in *CONCUR*, 2000, pp. 458–473.
[21] A. Cohen and K. S. Namjoshi, "Local proofs for global safety properties," *FMSD*, vol. 34, no. 2, pp. 104–125, 2009.
[22] K. Chatterjee, T. Henzinger, and B. Jobstmann, "Environment assumptions for synthesis," in *CONCUR*, 2008, pp. 147–161.
[23] O. Coudert, "Two-level logic minimization: An overview," *Integration, the VLSI Journal*, vol. 17, no. 2, pp. 97–140, 1994.
[24] I. Filippidis, R. M. Murray, and G. J. Holzmann, "A multi-paradigm language for reactive synthesis," in *SYNT*, 2015.
[25] H. Thimbleby and P. Ladkin, "From logic to manuals again," *IEE Proc.-Soft. Eng.*, vol. 144, no. 3, 1997.
[26] S. Singh and M. D. Wagh, "Robot path planning using intersecting convex shapes: Analysis and simulation," *TRO*, vol. RA-3, no. 2, 1987.
[27] B. Hayes and J. A. Shah, "Improving robot controller transparency through autonomous policy explanation," in *Human-Robot Interaction*, 2017, pp. 303–312.
[28] "Contract construction implementation," 2015. [Online]. Available: https://github.com/johnyf/contract_maker
[29] R. Ehlers, R. Könighofer, and R. Bloem, "Synthesizing cooperative reactive mission plans," in *IROS*, 2015, pp. 3478–3485.
[30] B. Jobstmann, A. Griesmayer, and R. Bloem, "Program repair as a game," in *CAV*, 2005, pp. 226–238.
[31] M. Abadi and L. Lamport, "Composing specifications," *TOPLAS*, vol. 15, no. 1, pp. 73–132, 1993.