Time-annotated game graphs for synthesis from abstracted systems

Scott C. Livingston

***Abstract*— The construction of discrete abstractions is a crucial part of many methods for control synthesis of hybrid systems subject to formal specifications. In general, the product of discrete abstractions may not be a discrete abstraction for the product of the underlying continuously-valued systems. Addressing this, we present a control synthesis method for transition systems that are built from components with uncertain timing characteristics. The new device, called here time-annotated game graphs, is demonstrated in a variety of examples. While it is applicable generally to parity games, we consider it in the context of control subject to GR(1) specifications. We show how a nominal strategy obtained without time knowledge can be modified to recover correctness when time information becomes available. The methods are applied to a brief case study of an aircraft electric power system.**

I. Introduction

In the practical application of formal synthesis in robotics, one may first attempt to construct discrete abstractions of the various subsystems before carefully combining them all into a product transition system over which (reactive) control is sought. The combination must be made "carefully" because, while the abstractions of components preserve properties like reachability for the corresponding subsystem, these same properties may fail to hold for the product of the underlying systems, i.e., bisimilarity may be lost.

In the context of formal methods for robotics research, this problem setting was first considered by Raman et al. [11]. In that work, the authors' treatment is entirely discrete; while being motivated by differences in continuously-valued timing, none of the timing information of abstractions is manipulated. Instead, they primarily rely on introducing liveness conditions into a GR(1) specification (cf. Section II for relevant definitions), which cause the synthesized strategy to wait for expected transitions to occur. While this ensures a safe interleaving of events, it is conservative because components may not require indefinitely long amounts of time to complete.

In the present work, we propose to cross the boundary of abstraction by annotating transition systems with time intervals of possible trajectory durations from the concrete (or "underlying") system. This is done not for timed synthesis, but rather, we use this timing information in two respects, corresponding to the main contributions of the paper. First, the product of the transition systems can be modeled as a game graph (cf. Section II for relevant definitions) based on feasible interleavings of component transitions. It is a game because an adversary can decide which interleaving is actually taken. This structure, which we call *time-annotated game graphs*, is amenable to incorporation into existing motion-planning methods that rely on game graphs, e.g., [4]. Second, fragments of a time-annotated game graph can be constructed as needed to verify and patch a given control strategy to ensure robustness, as we present in Sections IV and V. Despite the availability of well-studied models like timed automata [1], incremental synthesis on compositions of discrete abstractions motivate the notation developed here.

S.C. Livingston is with the California Institute of Technology, Pasadena, CA, USA. Email address of SCL is slivingston@cds.caltech.edu.

An outline of the paper is as follows. Background concepts and notation are briefly introduced in Section II. Our proposed structure, the time-annotated game graph, is developed in Section III. While applicable more broadly, we consider specifically control subject to GR(1) specifications in Section IV and present a method for patching a given strategy to be robust provided timing annotation. Finally, application to aircraft electric power systems is discussed in Section V.

II. Preliminaries

A *parity game* is a tuple $((V_0 \cup V_1, E), v_{\text{init}}, \chi)$, where V_0 and V_1 are disjoint and are known as Player 0 and Player 1 vertices, respectively, $(V_0 \cup V_1, E)$ is a directed graph, v_{init} is the initial vertex, and $\chi : V_0 \cup V_1 \to \mathbb{Z}$. $(V_0 \cup V_1, E)$ is referred to as a *game graph*. A *play* is a (finite or infinite) sequence of vertices $\rho = \rho_0\rho_1 \cdots \rho_i \cdots$ such that $\rho_0 = v_{\text{init}}$ and each consecutive pair is an edge in E. A *strategy* for Player i is a partial function $f_i : V^*V_i \to V$, where $V = V_0 \cup V_1$. Intuitively, a strategy for Player 0 decides which vertex to select given a history of visited vertices. Plays arise from both players using particular strategies. A play ρ is winning for Player 0 if the play reaches a V_1-vertex with no outgoing edges, or if the set of infinitely occurring vertices has even maximum χ-value, i.e., $\max\{\chi(v) \mid v \in \text{Inf}(\rho)\}$ is even. ω-regular winning conditions, e.g., Büchi or Rabin, can be reduced to parity games using finite memory. Because parity games have so-called "memoryless determinacy", it follows that many game types of interest admit finite memory strategies. For an introduction including proofs, consult [3].

A discrete abstraction is a finite transition system that is bisimilar to a hybrid system [2], [12]. To be precise, a *labeled transition system* is a tuple $(\text{AP}, S, S_{\text{init}}, L, \to)$ where AP is a finite set of atomic propositions, S is a set of states (not necessarily finite), $S_{\text{init}} \subseteq S$ are the initial states, $L : S \to 2^{\text{AP}}$ is a labeling that assigns subsets of AP to states, and $\to \subseteq S \times S$ is a transition relation. An atomic proposition $p \in \text{AP}$ is said to be `True` at state $s \in S$ if and only if $p \in L(s)$. $(s, s') \in \to$ is also denoted by $s \to s'$.

The notion of transition is quite general. Any dynamical control system can be regarded as a transition system by,

for instance, taking $S = \mathbb{R}^n$ and $s \to s'$ if and only if there is some controller such that the resulting flow maps s to s' in finite time. Then, viewed as a transition system, any controllable linear system has an edge between any two states. The significance of an abstraction, as we next define, comes from a combination of the usual definition of bisimulation with a finiteness requirement.

Let $\mathcal{T}_1 = (\mathrm{AP}, S_1, S_{\mathrm{init},1}, L_1, \to_1)$ and $\mathcal{T}_2 = (\mathrm{AP}, S_2, S_{\mathrm{init},2}, L_2, \to_2)$ be transition systems. We say that $\mathcal{T}_2$ is a *discrete abstraction* of $\mathcal{T}_1$ if S_2 is a finite set and there exists a bisimulation relation between $\mathcal{T}_1$ and $\mathcal{T}_2$, i.e., there exists $R \subseteq S_1 \times S_2$ such that for all $(s_1, s_2) \in R$,

1) $L_1(s_1) = L_2(s_2)$;
2) for all $s_1 \to s_1'$, there exists s_2' such that $s_2 \to s_2'$ and $(s_1', s_2') \in R$;
3) same as previous but swap the roles of $\mathcal{T}_1$ and $\mathcal{T}_2$;

and for all $i_1 \in S_{\mathrm{init},1}$, there exists $i_2 \in S_{\mathrm{init},2}$ such that $(i_1, i_2) \in R$, and same as previous but swap the roles of $\mathcal{T}_1$ and $\mathcal{T}_2$.

An alternative definition of abstraction is presented in [6].

The notation of linear temporal logic (LTL) is not needed until Sections IV and V, where GR(1) specifications (a fragment of LTL) are considered. A GR(1) specification is a formula,

$$\theta_{\mathrm{env}} \wedge \Box\, \rho_{\mathrm{env}} \wedge \left(\bigwedge_{j=0}^{m-1} \Box \diamondsuit\, \psi_j^{\mathrm{env}} \right) \implies \theta_{\mathrm{sys}} \wedge \Box\, \rho_{\mathrm{sys}} \wedge \left(\bigwedge_{i=0}^{n-1} \Box \diamondsuit\, \psi_i^{\mathrm{sys}} \right), \quad (1)$$

where each subformula is defined in terms of two sets of variables defined over finite domains [5]. One of these sets is uncontrolled (sometimes called "environment" variables), and the other is controlled (sometimes called "system" variables). The left-side of the implication in (1) is known as the "assumption", and the right-side as the "guarantee." Informally and in summary, the problem of reactive synthesis for a GR(1) formula is to construct a strategy for assigning a sequence of values to controlled variables given a sequence of environment-chosen values.

III. Time-annotated game graphs

A. Single transition system construction

Denote the set of nonnegative real numbers by $\mathbb{R}_+$.

Definition 1: A *time-annotated graph* is a tuple (X, δ, t), where (X, δ) is a directed graph, and $t : \delta \to 2^{\mathbb{R}_+}$ is a mapping from edges to closed intervals of the real line. An *execution* of a time-annotated graph is an infinite sequence of state-time pairs, $(s_0, \tau_0), (s_1, \tau_1), \ldots$, where for each step i, $(s_i, s_{i+1}) \in \delta$ and $\tau_i \in t((s_i, s_{i+1}))$. A finite execution is a finite sequence with the same form as (infinite) executions, except that the last entry is only a state (i.e., there is no transition time).

Let $(x, y) \in \delta$, and $[a, b] = t((x, y))$. The interval $[a, b]$ bounds the time required to complete the transition from x to y. In the terminology of timed automata [1], executions are like "timed words." Including an initial state and a labeling function, we have a *time-annotated labeled transition system*.

In the present section we focus on discrete abstractions formed from a single dynamical control system. Composition of abstractions, including a game based on possible interleavings, is introduced in Section III-B. The primary concern here is durations of trajectories that implement transitions of the abstraction. We now make this notion precise and then connect it to time-annotated graphs. Let $F : X \times U \to TX$ be a time-invariant dynamical control system, where X and U are the state and control input sets, respectively, and F defines a (control-dependent) vector field or difference equation. We do not need to fix whether time is continuous or discrete, or whether X is a smooth manifold or a finite set. Instead, the crucial piece is a set of durations that is defined with a transition system constructed from F as follows. Let $\mathcal{T}_1 = (\mathrm{AP}, S_1, S_{\mathrm{init},1}, L_1, \to_1)$ be a transition system where $S_1 \subseteq X$, and $(s, s') \in \to_1$ if and only if there exists a map $\mu : X \times I \to U$ such that the flow of F under μ is $\Phi_\mu : X \times I \to X$, where I is a closed interval in $\mathbb{R}$ containing 0, and such that $\Phi_\mu(s, 0) = s$, $\Phi_\mu(s, \max I) = s'$, and the partition of $\Phi_\mu(s, I) \cap L_1^{-1}(l)$ is nonempty for at most two labels $l \in 2^{\mathrm{AP}}$, and for each of these, $\Phi_\mu(s, I) \cap L_1^{-1}(l)$ is a simply connected component. Intuitively, the latter condition states that trajectories under the controller μ can change label at most once. Among other properties, this precludes so-called chattering at the label boundaries. We write μ as a function of state and time for generality, though of course one may restrict this depending on the application. Let $\mathrm{Controls}_{\mathcal{T}_1}(s, s')$ be the set of all such μ. Note that by definition of edges in $\mathcal{T}_1$, this set is nonempty. Finally, we define

$$\mathrm{Durations}_{\mathcal{T}_1}(s, s') = \big\{ \max I \mid \mu : X \times I \to U \in \mathrm{Controls}_{\mathcal{T}_1}(s, s') \big\}. \quad (2)$$

When we do not need to specify the dynamical control system, we will simply say that a transition system $\mathcal{T}_1$ is *equipped with duration sets* and indicate the association using a subscript; e.g., $\mathcal{T}_1$ is equipped with duration sets $\mathrm{Durations}_{\mathcal{T}_1}$. Intuitively, the transition $s \to s'$ in $\mathcal{T}_1$ is effectively implemented by some controller μ, which may be "open-loop" or state-feedback. In practical applications, the actual controller selected may be one of many possibilities.

From the perspective of synthesis for discrete abstractions, planning motions in terms of transitions of the abstraction ensures controllers exist achieving similar motions in the actual system (by definition). Now with a set of durations, we also know how long these transitions can take to complete on the physical system. Informally, if a finite-state machine decides to take a symbolic move, then it is important to consider how long this will take to complete by the underlying system, which $\mathrm{Durations}_{\mathcal{T}_1}(s, s')$ provides. In the present work, we focus on bounds for it, allowing us to move from sets of durations to transition intervals. A treatment based on bounding intervals allows for conservative decision-making when the exact set of durations is not known.

Let $\mathcal{T}_2 = (\mathrm{AP}, S_2, S_{\mathrm{init},2}, L_2, \to_2)$ be a discrete abstraction (recall definition from Section II) for a transition system $\mathcal{T}_1$ that is equipped with duration sets. A time-annotated graph (X, δ, t) is constructed from $\mathcal{T}_2$ and $\mathrm{Durations}_{\mathcal{T}_1}$ as follows. X is the set of used labels, i.e., $X = L_2(S_2)$. Note that by definition of bisimulation, X is thus also exactly the set of labels used by $\mathcal{T}_1$. For each $l, l' \in X$, $(l, l') \in \delta$ if and only if there is a path $\rho_0\rho_1\cdots\rho_K$ in $\mathcal{T}_2$ (i.e., $\rho_k \to_2 \rho_{k+1}$ for $k \in \{0, \ldots, K-1\}$) such that for some j,

$$L_2(\rho_k) = \begin{cases} l & \text{if } k \leq j \\ l' & \text{else.} \end{cases}$$

Denote the set of all such paths by $\mathrm{Paths}_{\mathcal{T}_2}(l, l')$. Informally speaking, an edge from l to l' exists if there is a way to move from a state in $\mathcal{T}_2$ labelled as l to a state labelled as l' while visiting only l-states and then l'-states en route. Finally, the time-annotation t is defined as follows. Recall from the definition of discrete abstraction there is a bisimulation R between $\mathcal{T}_1$ and $\mathcal{T}_2$. Denote $s_1 \sim s_2$ if $(s_1, s_2) \in R$. Then, for each edge $(l, l') \in \delta$, $t((l, l')) = [a, b]$ where

$$\begin{aligned} a = \inf \big\{ \Sigma_{k=0}^{K-1} \tau_k \mid \exists \rho_0\rho_1\cdots\rho_K \in \mathrm{Paths}_{\mathcal{T}_2}(l, l'), \\ \forall k \in \{0, \ldots, K-1\} \exists s_k \to_1 s_k' : \\ s_k \sim \rho_k \wedge s_k' \sim \rho_{k+1} \wedge \tau_k \in \mathrm{Durations}_{\mathcal{T}_1}(s_1, s_1') \big\} \end{aligned} \quad (3)$$

and b is defined similarly but as the supremum.

Example 2: Let p be a Boolean variable, i.e., p can be assigned the values of `True` or `False` and is able to change value instantaneously. We can easily construct a transition system $\mathcal{T}_p$ equipped with duration sets by setting $S = S_{\mathrm{init}} = \{v_0, v_1\}$, L is injective, and the transition relation $\to$ is such that $(S, \to)$ is a complete directed graph. For all edges, $\mathrm{Durations}_{\mathcal{T}_p}$ is the singleton $\{0\}$. Here, the identity relation is a bisimulation, i.e., $\mathcal{T}_p$ is a discrete abstraction for itself. Hence the time-annotated graph is merely a two-state complete directed graph, where all edges are annotated with $[0, 0]$.

Example 3: An important case is when the labeling L_2 of the discrete abstraction $\mathcal{T}_2$ is injective. Then the labels can be regarded as the states themselves, so that $\mathrm{Paths}_{\mathcal{T}_2}(l, l')$ is a singleton set precisely for transition edges $(L_2^{-1}(l), L_2^{-1}(l'))$ and otherwise is empty. Also, the infimum computation in (3) can be substantially simplified by being rewritten not in terms of paths, eliminating in particular the summation term. Explicitly, (3) simplifies to

$$a = \inf \{\tau \in \mathrm{Durations}_{\mathcal{T}_1}(s_1, s_1') \mid l \sim s_1 \wedge l' \sim s_1'\}. \quad (4)$$

Example 4: Consider the rectangle $[0, 2] \times [0, 1] \subset \mathbb{R}^2$, and the single-integrator dynamics

$$\dot{x} = u$$

where input is bounded as $\|u(t)\|_2 \leq 1$ (a closed ball centered at the origin of the plane). If we partition this rectangle into two regions

$$\begin{aligned} z_1 &= [0, 1] \times [0, 1] \\ z_2 &=]1, 2] \times [0, 1], \end{aligned}$$

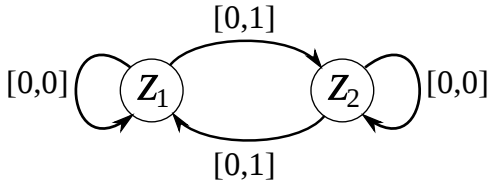

Fig. 1. Time-annotated graph for Example 4. It is constructed from an abstraction of a single-integrator dynamical system restricted to two unit squares in the plane. Intuitively, from any cell, the same cell can be reached instantaneously (using $u(t) = 0$), hence self-loops have time bounds $[0, 0]$, and going from one cell to another can take at most time 1, but possibly as little as 0 time (for points upon the boundary).

then a discrete abstraction is easily obtained. Hence, a corresponding time-annotated graph has states ζ_1 and ζ_2, edges $\{(1, 1), (1, 2), (2, 1), (2, 2)\}$, and time annotation

$$\begin{aligned} t((1, 1)) &= [0, 0], \\ t((1, 2)) &= [0, 1], \end{aligned}$$

etc. This graph is depicted in Figure 1. An example execution is $(z_1, .5), (z_2, .5), (z_1, .5), \ldots$. This execution could arise from a controller that alternates between the two cells. The discrete transition between the cells is implemented by a trajectory of length 0.5 achieved by steering the single-integrator at constant speed, i.e., $\|u(t)\|_2 = 1$ for all t.

B. Games and composition

In the previous section, a method for constructing time-annotated graphs from dynamical control systems with discrete abstractions was presented. The key item preserved is an interval of times at which the corresponding label change can occur. We now present a method for composing these and explore its consequences. The notion introduced here is called *time-annotated game graphs*, and it is exploited in Section IV for control under temporal logic specifications.

Before treating general composition (cf. Section III-C), we begin with construction from a pair. Given two time-annotated graphs (X, δ, t) and (Y, ξ, u), we may construct a game graph $(V_0 \cup V_1, E)$ (recall the definition in Section II) that indicates possible interleavings of transitions as follows. The set of Player 0 vertices is $V_0 = X \times Y$. The set of Player 1 vertices is $V_1 = (X \times Y) \times (\delta \times \xi) \times \{\mathtt{nil}, 1, 2\}$.

Edges together with a new time bound labeling τ are as follows. Let $(x_1, x_2) \in \delta$ and $(y_1, y_2) \in \xi$, and let the corresponding time bounds be $[a, b] = t((x_1, x_2))$ and $[c, d] = u((y_1, y_2))$. For brevity, let $e = ((x_1, x_2), (y_1, y_2))$. This pair of component edges may be initiated (i.e., is a control decision) and thus modeled as an outgoing edge from the Player 0 vertex (x_1, y_1) to the initiation vertex $((x_1, y_1), e)$, from which Player 1 may select the actual interleaving. Without loss of generality (due to symmetry), we show two cases of the order of a, b, c, d. For ease of reading, edges are also denoted by arrows over which the time bound is written, i.e., $u \xrightarrow{[c,d]} v$ denotes $(u, v) \in E$ and $\tau((u, v)) = [c, d]$. In all cases, there is an edge $(x_1, y_1) \to ((x_1, y_1), e, \mathtt{nil})$.

1) $b < c$. Then there are edges

$$((x_1, y_1), e, \texttt{nil}) \xrightarrow{[a,b]} ((x_2, y_1), e, 1) \xrightarrow{[c-b,d-a]} (x_2, y_2).$$

2) $a \leq c \leq b \leq d$. Then there are edges

$$((x_1, y_1), e, \texttt{nil}) \xrightarrow{[a,b]} ((x_2, y_1), e, 1) \xrightarrow{[0,d-a]} (x_2, y_2)$$
$$((x_1, y_1), e, \texttt{nil}) \xrightarrow{[c,d]} ((x_1, y_2), e, 2) \xrightarrow{[0,b-c]} (x_2, y_2).$$

Intuitively, each element of V_1 can be viewed as having three parts. First, there is the actual current state, which is an element of $X \times Y$. Second, there is a tuple of edges from the components (X, δ) and (Y, ξ). This indicates the product transition that is intended, but may not be followed depending on timing constraints. The third part is the index of the component that changed in the last step of the interleaving, or `nil` to indicate the beginning of a Player 1 round. The annotation on each outgoing edge of a Player 1 vertex indicates an interval of times that may elapse before that edge is taken, assuming Player 1 (the adversary) chooses it. In other words, for each step in a particular interleaving of (x_1, x_2) and (y_1, y_2), the time bounds of the previous steps may be accumulated.

Example 5: Recall the time-annotated graph of Example 4 (shown in Figure 1). Practically, the transition from cell z_1 to z_2 will not be instantaneous, and thus we take the lower bound time for that transition to be arbitrarily small $\epsilon > 0$. Make the same adjustment for the reverse direction (z_2 to z_1). Composing the result with Example 2, we obtain the time-annotated game graph shown in Figure 2 from Algorithm 1, having taken ϵ as 0.001. Figures 3 and 4 are detailed views.

C. Algorithm for general composition

Algorithm 1 provides composition from a set of time-annotated graphs into a time-annotated game graph. Detailed comments follow. First note that

$$V_1 \subseteq V_0 \times (\delta_1 \times \cdots \times \delta_J) \times \{\texttt{nil}, 1, 2, \ldots, J\}, \quad (5)$$

which is the straightforward generalization of the two-graph case outlined in the previous section. An intuitive motivation to keep in mind is that we are trying to model possible interleavings of component transition systems when transitions are initiated in all of them simultaneously. In Lines 5–9, the set of Player 1 is initialized using all tuples of edges in the component graphs. An edge is added to the game providing Player 0 the choice of initiating this. In Line 12, $\mathrm{Permutations}(A)$ is the set of all permutations of the set A. Here it is used to check every possible interleaving of the component transitions. In Line 16, because a particular interleaving of transitions may be infeasible, due to accumulated time bounds, the new edges and Player 1 vertices must be kept separate before committing to E and V_1. (We omit obvious optimizations to keep the algorithm here simple.)

Several basic results about Algorithm 1 follow.

Proposition 6: Let $N = \max_j |\delta_j|$. The worst-case computational complexity of Algorithm 1 is $O(J \cdot N^J \cdot J!)$.

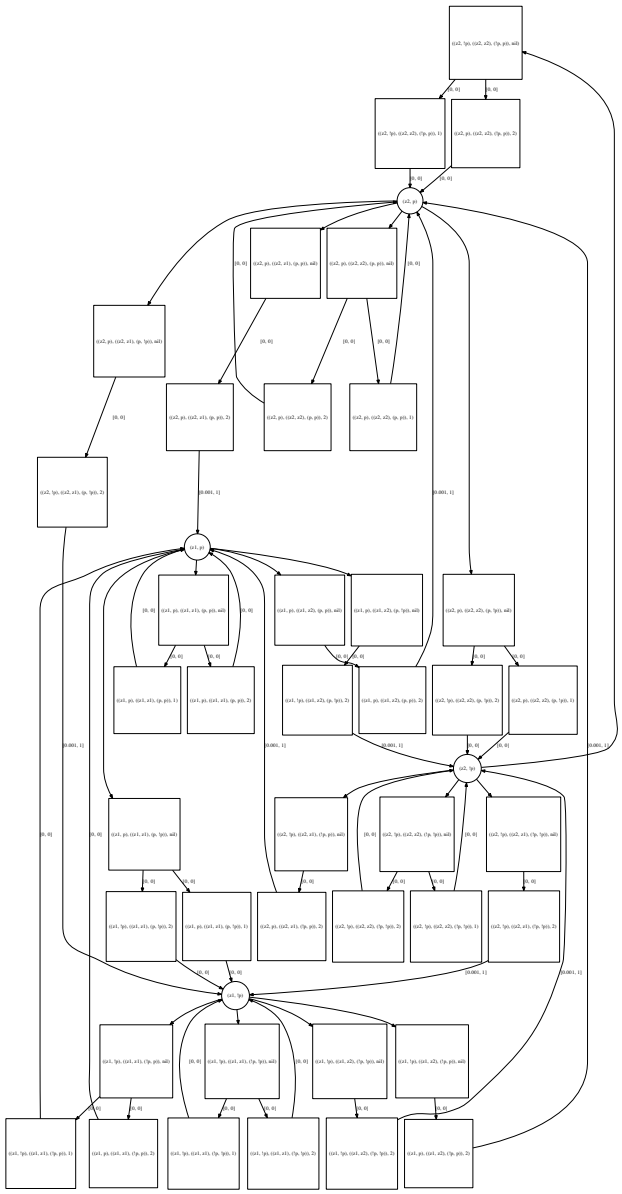

Fig. 2. Time-annotated game graph for Example 5, generated using Algorithm 1. Player 0 vertices are indicated as circles, and Player 1 vertices as squares.

Proof: The nested for-loops on lines 11 and 12 dominate. The former takes elements from $V_{1,\mathrm{orig}}$, which has size

$$|\delta_1 \times \cdots \times \delta_J| \leq N^J.$$

The number of permutations of J items is $J!$. Since the loops are nested, the total number of iterations is obtained by multiplying their respective numbers of iterations. ■

For the following, let $(V_0 \cup V_1, E, \tau)$ be as returned by Algorithm 1.

Remark 7: There is no edge with Player 0 vertices for both end-points, i.e., $V_0 \times V_0 \cap E = \emptyset$.

Proposition 8: Let $u, v \in V_0$ such that there exists a path from u to v with all intermediate vertices in V_1. The length of the path (including end-points u and v) is $J + 2$.

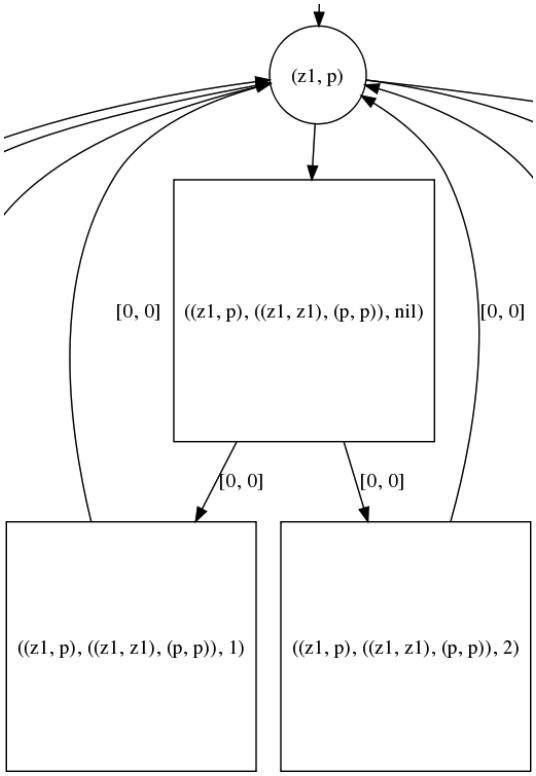

Fig. 3. Self-loop occurring in the time-annotated game graph for Example 5. (Zoom-in of part of Figure 2.)

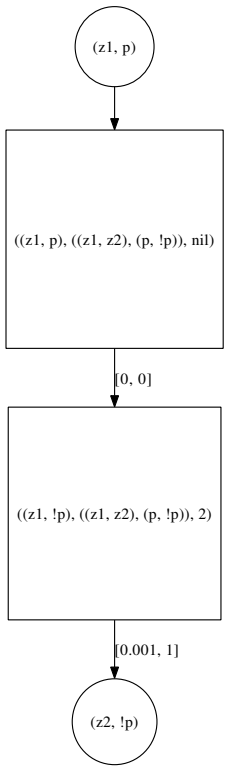

Fig. 4. Example of transition with only one feasible interleaving, given time bounds. Snippet from time-annotated game graph for Example 5. (Adjusted from the corresponding fragment of Figure 2 for clarity.)

Proof: By the previous remark, in any path from u to v there must be at least one vertex in V_1. It follows that if there is any path from V_0 to itself, then there are $u, v \in V_0$ satisfying the hypothesis of this proposition. Let $\langle u, w_1, w_2, \ldots, w_M, v\rangle$ be a path where $w_1, \ldots w_M \in V_1$. In Algorithm 1, after being initialized by $E := \emptyset$ (i.e., to empty), edges are added to E only at lines 8 and 36. The former adds edges from V_0 to V_1 exactly while V_1 is being first populated in the for-loop of lines 5–9. The latter uses E_{add}, which is built sequentially at line 29 for each permutation (for-loop on line 12). The values stored to intermediate variables prev and this in Algorithm 1 are in sequence, i.e., prev is initialized to one of the original V_1 nodes (from lines 5–9), and at the end of each iteration prev is changed to this (line 33). Thus for each feasible permutation, E_{add} is a chain of edges of length J, so it corresponds to a path of length $J + 1$. As observed earlier, each such chain begins with an edge from V_0, resulting in a path of length $J + 2$. Because no other edges are added to E, any path from u to v that does not include a vertex in V_0 besides u, v themselves must have length $J + 2$ ■

By the above results we may view time-annotated game graphs as expanded product graphs, where each edge in the product graph has been replaced by a directed acyclic subgraph of uncontrolled (Player 1) vertices. Being essentially an enumeration of interleavings, the complexity landscape is bleak, though we are able to prune interleavings that are infeasible based on duration intervals.

IV. Control synthesis for GR(1) using time-annotation

In this section we present an application of the developments of Section III to control subject to GR(1) specifications. Recall the brief introduction to the GR(1) fragment of LTL from Section II. Discrete synthesis for GR(1) has been studied fairly extensively, both in the theoretical computer science literature [5] and, more recently, in the robotics and control literature. A crucial first step in many of these methods is construction of an abstraction. As already described in the introductory Section I, difficulties arise if we are forming a product of multiple abstractions. Nonetheless, taking this product is desirable because, e.g., it may be difficult to apply existing abstraction techniques after computing a product of the concrete, potentially heterogeneous component systems. Thus we are motivated to *patch* a nominal control strategy synthesized with respect to this product of discrete abstractions. In recent work by the authors [7], algorithms have been proposed for patching strategies to recover correctness with respect to a changing specification. In Algorithm 2, we exploit these methods to incrementally patch a given strategy automaton based on time-annotated graphs that model component systems.

Algorithm 2 relies on several subroutines, which we now describe. The main work of Algorithm 1 is to construct the edges of the game graph corresponding to a nominal transition in the product of the component discrete abstrac-

Algorithm 1 Build time-annotated game graph

1: IN: time-annotated graphs $(X_1, \delta_1, t_1), \ldots, (X_J, \delta_J, t_J)$
2: OUT: game graph $(V_0 \cup V_1, E)$, time annotation τ
3: $V_0 := X_1 \times \cdots \times X_J$
4: $V_1 := E := \emptyset$
5: **for all** $((x_1, x_1'), \ldots, (x_J, x_J')) \in \delta_1 \times \cdots \times \delta_J$ **do**
6: $v := ((x_1, \ldots, x_J), ((x_1, x_1'), \ldots, (x_J, x_J')), \texttt{nil})$
7: $V_1 := V_1 \cup \{v\}$
8: $E := E \cup \{((x_1, \ldots, x_J), v)\}$
9: **end for**
10: $V_{1,\text{orig}} := V_1$
11: **for all** $((x_1, \ldots, x_J), ((x_1, x_1'), \ldots, (x_J, x_J')), \texttt{nil}) \in V_{1,\text{orig}}$ **do**
12: **for all** $\sigma \in \text{Permutations}(\{1, 2, \ldots, J\})$ **do**
13: $\text{prev} := ((x_1, \ldots, x_J), ((x_1, x_1'), \ldots, (x_J, x_J')), \texttt{nil})$
14: $v := (x_1, \ldots, x_J)$
15: $a_{\max} := b_{\max} := 0$
16: $V_{1,\text{add}} := E_{\text{add}} := \emptyset$
17: **for all** $i = 1, 2, \ldots, J$ **do**
18: $[a, b] := t_{\sigma(i)}(x_{\sigma(i)}, x'_{\sigma(i)})$
19: **if** $b < a_{\max}$ **then**
20: **skip to next permutation**
21: **end if**
22: $v_{\sigma(i)} := x'_{\sigma(i)}$ //Take step
23: **if** $i < J$ **then**
24: $\text{this} := (v, ((x_1, x_1'), \ldots, (x_J, x_J')), \sigma(i))$
25: $V_{1,\text{add}} := V_{1,\text{add}} \cup \{\text{this}\}$
26: **else**
27: $\text{this} := v$ //Last step; reach V_0 vertex
28: **end if**
29: $E_{\text{add}} := E_{\text{add}} \cup \{(\text{prev}, \text{this})\}$
30: $\tau((\text{prev}, \text{this})) := [\max\{a - b_{\max}, 0\}, \max\{b - a_{\max}, 0\}]$
31: $a_{\max} := \max\{a_{\max}, a\}$
32: $b_{\max} := \max\{b_{\max}, b\}$
33: $\text{prev} := \text{this}$
34: **end for**
35: $V_1 := V_1 \cup V_{1,\text{add}}$
36: $E := E \cup E_{\text{add}}$
37: **end for**
38: **end for**
39: **return** $(V_0 \cup V_1, E, \tau)$

Algorithm 2 Patch strategy using timed models

1: IN: time-annotated graphs $(X_1, \delta_1, t_1), \ldots, (X_J, \delta_J, t_J)$, strategy automaton $A = (S, \delta, L)$, GR(1) specification φ, safety requirement $\square\psi$.
2: OUT: patched automaton A'
3: $\varphi' := \varphi$; $A' := A$
4: changed_spc := `false`
5: **for all** $(s_1, s_2) \in E(A')$, the edge set of A' **do**
6: $(\bar{V}_0 \cup \bar{V}_1, \bar{E}) := \texttt{SingleTransitionTAGG}(L(s_1), L(s_2))$
7: **if** there exists $v \in \bar{V}_1$ that does not satisfy ψ **then**
8: $\varphi' := \texttt{Restrict}(\varphi', (L(s_1), L(s_2)))$
9: changed_spc := `true`
10: **end if**
11: **end for**
12: **if** changed_spc **then**
13: $A' := \texttt{Patch}(A', \varphi')$
14: **if** `Patch`() succeeded **then**
15: **goto** Line 4
16: **else**
17: **abort** //Failed to patch. Try global re-synthesis.
18: **end if**
19: **end if**
20: **return** A'

tions. The subroutine `SingleTransitionTAGG`() (Line 6) constructs only the subgraph $(\bar{V}_0 \cup \bar{V}_1, \bar{E})$ that is formed to model this transition. Because the Player 1 vertices, which are uncontrolled, model possible intermediate states due to timing uncertainty of the composed systems, the safety condition ψ can be checked in Algorithm 2 merely by enumerating $\bar{V}_1$. If such a violating state is found, then the corresponding transition of the product system cannot be taken safely and thus must be effectively precluded by extending the original specification. This is straightforward but tedious for GR(1) formulae, so we summarize this step using the subroutine `Restrict`() (Line 8). Finally, the subroutine `Patch`() (Line 13) modifies the strategy automaton given a change in the specification, e.g., as described in [7].

V. Example: aircraft electric power system

An important practical application domain for modern control is aircraft electric power systems. Modern aircraft have complex electric power systems due to the increasing reliance on electric power relative to other traditional sources [9]. Future designs will need to be more sophisticated to cope with increasing complexity while satisfying formal requirements. Bringing formal verification and synthesis to bear on the design of aircraft electric power systems is the topic of current research, and approaches based on composition are especially relevant [10].

One of the basic means of control input in this setting is contactors [9]. In summary, a contactor is a switch that can be commanded to close or open, thereby making or breaking the connectivity along the corresponding branch of an electrical circuit. For aircraft electric power systems, switching contactors can be used to route power as needs change or faults occur. There are numerous questions that can be formulated in this setting, such as switching to identify faults when there are limited sensors available [8].

A crucial assumption often made in previous work is that contactors can be switched instantaneously. In this section, we use time-annotated game graphs to make a nominal strategy robust with respect to timing uncertainties of contactors. We use a simplified model of the AC side of an aircraft electric power system, as shown in Figure 5.

The specification is defined as follows. Recall the temporal logic notation from Section II. At each time step, at least one

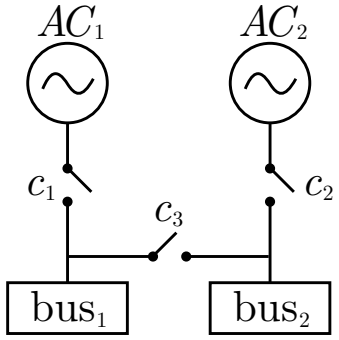

Fig. 5. The states of the contactors are modeled by c_1, c_2, c_3. The AC sources are referred to as AC_1, AC_2. The various electrical loads that require power and are critical for flight are modeled as occurring on two sides of the aircraft using bus_1 and bus_2.

of the AC sources is assumed to provide power, i.e.,

$$\square\left(\mathrm{AC}_1 \vee \mathrm{AC}_2\right). \tag{6}$$

Our controller must guarantee that, for each $j \in \{1, 2\}$,

$$\square\left(\bigvee_i \left(\mathrm{AC}_i \wedge \mathtt{Path}(\mathrm{AC}_i, \mathrm{bus}_j)\right)\right) \leftrightarrow \mathrm{bus}_j \wedge \square\diamondsuit \mathrm{bus}_j, \tag{7}$$

which consists of two conditions. The $\square$-subformula on the left includes a Boolean function $\mathtt{Path}(a, b)$ that is true if and only if there is a path between the two nodes in the graph induced by the states of contactors c_1, c_2, and c_3 in Figure 5. Thus, the $\square$-subformula enforces that the variable bus_j is true if and only if there is a route to a power source. The $\square\diamondsuit$-subformula on the right is the requirement that each bus is repeatedly powered. Note that counters can be introduced to provide time limits on buses being unpowered.

To prevent damage, it is necessary to avoid connecting two active AC sources together. As a safety requirement for the network shown in Figure 5, this can be written

$$\square \neg \left(c_1 \wedge c_2 \wedge c_3\right). \tag{8}$$

The inner conjunction corresponds to the state of all contactors being closed, hence a path connecting both AC sources.

Consider the nominal strategy shown in Figure 6 obtained from the above specification. It is "nominal" because it relies on the assumption that contactors can switch instantaneously, which is a consequence of abstracting them as Boolean variables c_1, c_2, and c_3. Now suppose we obtain a time-annotated graph for each contactor based on device performance statistics. Roughly, one would expect that if commanded to switch, a contactor could do so but possibly after some delay, whereas it can remain in its current configuration (a "no-op" command) without delay. An appropriate time-annotated graph for this is given in Example 4.

In Figure 7, the result of synthesis is shown for a specification that is modified to prevent the transition $(c_1, c_3) \to (c_2, c_3)$ that is detected as unsafe in the manner of Lines 6–8 of Algorithm 2. The time-annotated game graph reveals that, due to the possibility of contactors switching after some delay, it is possible that both AC sources would be transiently connected to each other.

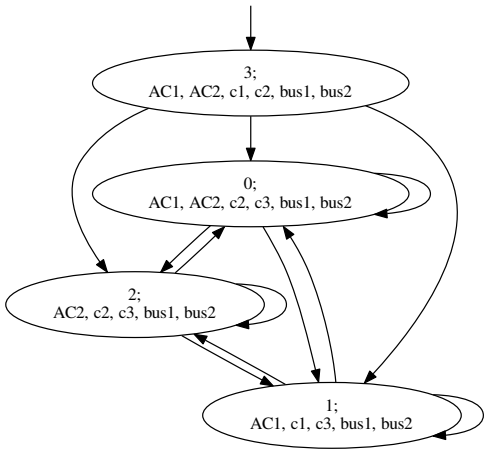

Fig. 6. Nominal protocol, without considering time delays that occur during contactor switches.

VI. Future work

While the previous two sections focus on GR(1) synthesis, the approach is useful for other problems that can be expressed in terms of game graphs. An interesting direction for future work is studying a time-annotated form of the game arena that is explicitly constructed in a sampling-based motion planning method proposed by Karaman and Frazzoli [4], which may allow for extending their method to stochastic systems.

Acknowledgments

The author thanks Richard M. Murray for motivating discussions. This work was partially supported by United Technologies Corporation and IBM, through the industrial cyber-physical systems (iCyPhy) consortium.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, July 2000.

[3] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[4] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *Proc. of the American Control Conf. (ACC)*, pages 735–742, Montréal, Canada, June 2012.

[5] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace inclusion. *Information and Computation*, 200:35–61, 2005.

[6] J. Liu and N. Ozay. Abstraction, discretization, and robustness in temporal logic control of dynamical systems. In *Proc. of Hybrid Systems: Computation and Control*, pages 293–302, April 2014.

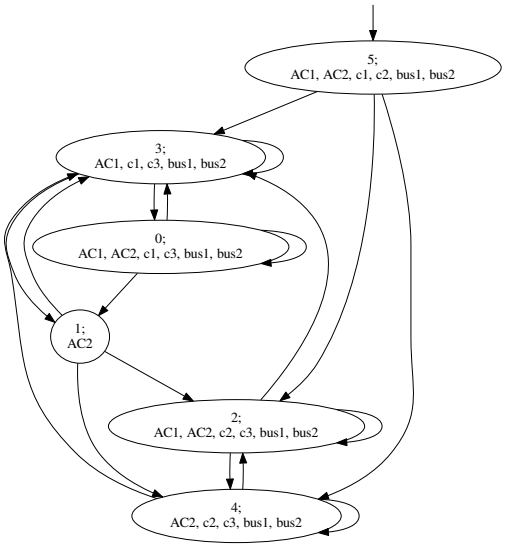

Fig. 7. Protocol obtained after explicitly blocking the unsafe component transition $(c_1, c_3) \rightarrow (c_2, c_3)$ in Figure 6. The corresponding two steps above are from node 3 to node 1 and then to node 2. Node 1 is labeled with a new intermediate state, AC_2, wherein no contactors are closed.

[7] S. C. Livingston and R. M. Murray. Hot-swapping robot task goals in reactive formal synthesis. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 101–107, Los Angeles, CA, USA, December 2014.

[8] Q. Maillet, H. Xu, N. Ozay, and R. M. Murray. Dynamic state estimation in distributed aircraft electric control systems via adaptive submodularity. In *Proc. of CDC*, pages 5497–5503, 2013.

[9] I. Moir and A. Seabridge. *Aircraft Systems: Mechanical, Electrical and Avionics Subsystems Integration*. Aerospace Series. Wiley, 2008.

[10] P. Nuzzo, H. Xu, N. Ozay, J. Finn, A. Sangiovanni-Vincentelli, R. Murray, A. Donzé, and S. Seshia. A contract-based methodology for aircraft electric power system design. *IEEE Access*, 2:1–25, 2014.

[11] V. Raman, N. Piterman, C. Finucane, and H. Kress-Gazit. Timing semantics for abstraction and execution of synthesized high-level robot control. *IEEE Transactions on Robotics*, 31(3):591–604, June 2015.

[12] P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.