Synthesis of Run-To-Completion Controllers for Discrete Event Systems

Yehia Abd Alrahman^{1}, Victor Braberman^{2,3}, Nicolás D'Ippolito^{2,3}, Nir Piterman^{1,4} and Sebastian Uchitel^{2,3,5}

Abstract— **A controller for a Discrete Event System must achieve its goals despite its environment being capable of resolving race conditions between controlled and uncontrolled events. Assuming that the controller loses all races is sometimes unrealistic. In many cases, a realistic assumption is that the controller sometimes wins races and is fast enough to perform multiple actions without being interrupted. However, in order to model this scenario using control of DES requires introducing foreign assumptions about scheduling, that are hard to figure out correctly. We propose a more balanced control problem, named run-to-completion (RTC), to alleviate this issue. RTC naturally supports an execution assumption in which both the controller and the environment are guaranteed to initiate and perform sequences of actions, without flooding or delaying each other indefinitely. We consider control of DES in the context where specifications are given in the form of linear temporal logic. We formalize the RTC control problem and show how it can be reduced to a standard control problem.**

I. Introduction

The field of controller synthesis covers a spectrum of control problems, including *Reactive Synthesis* [1] and *Supervisory Control* [2]. It targets dynamical systems whose state change is governed by the occurrence of discrete events. In these settings, system goals and the environment (or the uncontrolled plant) are specified as an accepted formal language, and the automatic synthesis procedure generates a correct-by-construction controller (or a supervisor).

The controller must achieve its goals by dynamically disabling some of the controllable events based on the events that it has observed so far. The controller must be *robust*. That is, it must be able to achieve its goals no matter what the environment does. However, the controller has no means of forcing the environment to generate an event. Thus, the environment not only identifies the possible controllable events in a given environment state, but also gets to choose the next scheduled event out of those selected by the controller and all enabled uncontrollable events.

This asymmetric interaction between the environment and the controller represents a worst-case scenario that asks for producing robust controllers, achieving their goals despite the advantage offered to the environment.

In many application domains, this asymmetric interaction is too adversarial and requires adding explicit foreign assumptions that restrict the behaviour of the environment. One such application domain is that of embedded systems design in which reactive languages (e.g., [3], [4], [5]) adopt a synchronous hypothesis where the system can react to an external stimulus with all the computation steps it needs [6]. To handle such applications, the modeller is forced to introduce assumptions about the scheduling of the environment and the controller. These assumptions are not only hard to figure out correctly, but are also far from the actual focus of the control problem under consideration. In many cases, this may lead to generating controllers that satisfy their goals trivially by cornering the environment and disrupting its behaviour dramatically. Furthermore, the written specifications become harder to read and understand, and consequently trickier to be incrementally developed due to their extensive dependencies.

In this paper, we introduce a novel control problem, named *run-to-completion* (RTC), to mitigate the shortcomings of classical control, for such applications. RTC is a more balanced control problem that supports more natural modelling of systems that can initiate and perform sequences of actions in response to external stimuli. In essence, RTC provides a natural execution assumption in which both the controller and the environment may initiate and perform sequences of computation steps, without flooding or delaying each other indefinitely. Namely, the controller has the ability to block environment actions for a finite time, this is akin to the controller stating that it still has something to do. However, when the controller yields control back to the environment, it has to yield completely, i.e, the controller must allow all uncontrollable actions enabled by the environment. Furthermore, to support environment's run-to-completion, the controller must not interrupt the environment during its turn.

RTC is suitable to control componentized systems where a response to a single external stimulus may require communication among subsystems. Due to flexible deadlines in RTC control, we are no longer required to count (or hardcode) the number of computation steps for the system before it is ready again to react to the next stimulus.

We show how to reduce RTC control to a modified control problem (i.e., with an asymmetric interaction). Furthermore, we show that RTC Control when used with GR(1) [7] goals can be reduced to Streett control of index 2 [8].

This paper is organised as follows: In Sect. II we present the necessary background and in Sect. III we present a motivating example about a UAV reconnaissance mission.

^{1}Department of Computer Science and Engineering, University of Gothenburg, Sweden.

^{2}Departamento de Computacón, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina.

^{3}Instituto de Ciencias de la Computación, CONICET, Argentina.

^{4}University of Leicester, Leicester, UK.

^{5}Imperial College London, UK.

This work is supported by the following grants: the ERC Consolidator grant D-SynMA (No. 772459), the Marie Skłodowska-Curie BeHAPI (No. 778233), the grants ANPCYT PICT 2018-3835, ANPCYT PICT 2015-1718, CONICET PIP 2014/16 N°11220130100688CO, UBACYT 20020170100419 BA, and the Swedish research council grants (No. 2020-04963) and SynTM (No. 2020-03401).

In Sect. IV and Sect. V, we formally define RTC control and use the example to show its novel features. In Sect. VI, we solve RTC control by a reduction to standard control. Finally, In Sect. VII, we conclude and discuss related work.

II. Background

A. Doubly-Labelled Transition System (DLTS)

LTSs have been widely used for modelling and analysing the behaviour of concurrent and distributed systems (e.g. [9]. An LTS is a transition system where transitions are labelled with actions or events. Here, as a part of the reasoning, we also label the states of the transition system with propositions, representing the set of events (or actions) that can be enabled from a specific state. Therefore, we use a DLTS instead. The use of DLTS is only a technicality and will not impact on the type of the generated controllers. In fact, state labels will be ignored in the generated controllers.

Definition 2.1: (DLTS) A DLTS is $T = (S, P, A, \Delta, L, s_0)$, where S is a finite set of states, P is a set of *state propositions*, A is a *transition alphabet* partitioned $A = A_T \uplus \overline{A_T}$ to actions controlled by T and actions monitored by T, $\Delta \subseteq (S \times A \times S)$ is a transition relation, $L : S \to 2^P$ is a labeling function, and $s_0 \in S$ is the initial state.

We denote $\Delta_\ell(s) = \{s' \mid (s, \ell, s') \in \Delta\}$, $\Delta_{A'}(s) = \bigcup_{\ell \in A'} \Delta_\ell(s)$, and $\Delta(s) = \Delta_A(s)$. This notation is extended to sets of states, e.g., $\Delta_\ell(S') = \bigcup_{s \in S'} \Delta_\ell(s)$. We say ℓ is enabled in state s if $\Delta_\ell(s) \neq \emptyset$.

We say a DLTS is transition-deterministic if (s, ℓ, s') and (s, ℓ, s'') are in Δ implies $s' = s''$. An execution of T is a maximal sequence of states and transition labels $\pi = s_0, a_0, s_1, \ldots$ where s_0 is the initial state and for every $i \geq 0$ we have $(s_i, a_i, s_{i+1}) \in \Delta$.

Definition 2.2: (The Parallel Composition of DLTS(s)) Let $M = (S_M, P_M, A_M, \Delta_M, L_M, s_{0_M})$ and $E = (S_E, P_E, A_E, \Delta_E, L_E, s_{0_E})$ be two DLTSs. The *parallel composition* of M and E is defined by a symmetric and a binary operator $\|$ such that $M \| E$ is also a DLTS $T = (S_M \times S_E, P, A_M \cup A_E, \Delta, L, (s_{0_M}, s_{0_E}))$, where $P = P_M \uplus P_E$, $L(m, e) = L_M(m) \uplus L_E(e)$, and Δ is the smallest relation that satisfies the rules below,

$$\frac{m \xrightarrow{\ell} m'}{(m,e) \xrightarrow{\ell} (m',e)}\ \ell \notin A_E \qquad \frac{e \xrightarrow{\ell} e'}{(m,e) \xrightarrow{\ell} (m,e')}\ \ell \notin A_M$$
$$\frac{m \xrightarrow{\ell} m',\ e \xrightarrow{\ell} e'}{(m,e) \xrightarrow{\ell} (m',e')}\ \ell \in A_M \cap A_E$$

Note that parallel composition only synchronise on actions, and thus preserves the proposition values of the two separate parts.

B. Fluent Linear Temporal Logics

Linear temporal logics are widely used to describe and analyse behaviour requirements [10], [11], [12], [13]. The *Fluent Linear Temporal logic (FLTL)* [10] replaces state propositions in traditional temporal logics with fluents. A fluent is a predicate over a set of initiating and terminating actions. Once triggered by an initiating action, a fluent continues to hold as long as no terminating action is enabled. Thus, FLTL provides a uniform framework for specifying both instantaneous actions and also actions that take time [10], [14]. To simplify notations we do not include the next operator in FLTL. All our results can be easily generalised to include the next operator.

FLTL was designed for LTS, here we adapt it for DLTS by introducing fluents to also account for the propositions that label states of a DLTS. We introduce two types of fluents: *transition fluents* and *proposition fluents*. A *transition fluent* f is defined by a pair of sets of actions and a Boolean value: $f = \langle I_f, T_f, \mathit{Init}_f \rangle$, where $I_f \subseteq \mathit{Act}$ is the set of initiating actions, $T_f \subseteq \mathit{Act}$ is the set of terminating actions and $I_f \cap T_f = \emptyset$. A transition fluent may be initially *true* or *false* as indicated by Init_f. Every action $\ell \in \mathit{Act}$ induces a *transition fluent*, namely $\dot{\ell} = \langle \ell, \mathit{Act} \setminus \{\ell\}, \mathit{false} \rangle$. Every state proposition p of a DLTS induces a *proposition fluent* p.

Let $\mathcal{F}$ be the set of all fluents over Act and P. An FLTL formula is built up from the standard Boolean connectives and the temporal operator U (strong until) as follows:

$$\varphi ::= f \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi\ U\ \psi,$$

where $f \in \mathcal{F}$. As usual we introduce $\wedge$, $\Diamond$ (eventually), $\Box$ (always), and W (weak until) as syntactic sugar. Let Π be the set of infinite executions of a DLTS T over Act and P. For an execution $\pi = s_0, \ell_0, s_1, \ell_1, \ldots$, we say it satisfies a transition fluent f at position i, denoted $\pi, i \vDash f$, if and only if one of the following conditions holds:

- $\mathit{Init}_f \wedge (\forall j \in \mathbb{N} \cdot 0 \leq j \leq i \to \ell_j \notin T_f)$
- $\exists j \in \mathbb{N} \cdot (j \leq i \wedge \ell_j \in I_f) \wedge (\forall k \in \mathbb{N} \cdot j < k \leq i\ \to \ell_k \notin T_f)$

It satisfies a proposition-fluent p at position i, denoted $\pi, i \vDash p$, if and only if $p \in L(s_i)$.

Given an infinite execution π, the satisfaction of a formula φ at position i, denoted $\pi, i \vDash \varphi$, is defined as follows:

$$\begin{array}{lll}
\pi, i \vDash f & \triangleq & \pi, i \vDash f \\
\pi, i \vDash \neg\varphi & \triangleq & \neg(\pi, i \vDash \varphi) \\
\pi, i \vDash \varphi \vee \psi & \triangleq & (\pi, i \vDash \varphi) \vee (\pi, i \vDash \psi) \\
\pi, i \vDash \varphi\ U & \triangleq & \exists j \geq i \cdot \pi, j \vDash \psi \wedge \forall\ i \leq k < j \cdot \pi, k \vDash \varphi
\end{array}$$

We say that φ holds in π, denoted $\pi \vDash \varphi$, if $\pi, 0 \vDash \varphi$. A formula $\varphi \in$ FLTL holds in an LTS T (denoted $T \vDash \varphi$) if it holds on every infinite execution produced by T.

We assume that user supplied specifications do not use proposition fluents. However, proposition fluents are required for various parts of our analysis.

C. Controller Synthesis

The standard control problem is as follows: Consider an FLTL formula φ and a DLTS model E of the environment, with the set of actions A partitioned into environment actions A_E and monitored controller actions $\overline{A_E}$. Construct a DLTS M to control $\overline{A_E}$ and to monitor A_E such that when composed with E (i.e. $E \| M$), the controller does not block environment actions (i.e. actions in A_E), $E \| M$ is deadlock-free, and every execution of $E \| M$ satisfies φ. For simplicity (and taking the controller's point of view), we uniformly denote by U (for uncontrollable) the set $A_E = \overline{A_M}$ and by C (for controllable) the set $\overline{A_E} = A_M$. That is, U is the set of actions controlled by the environment and monitored by the controller and C is the set of actions monitored by the environment and controlled by the controller.

A legal controller does not block the actions in U and enables only actions in C that are available. This notion is based on that of *legal environment* for Interface Automata [15]. Formally *legality* is defined as follows.

Definition 2.3: (Legality) Consider a DLTS model of the environment $E = (S_E, P_E, A, \Delta, L_E, s_{E_0})$ and a DLTS model of the controller $M = (S_M, P_M, A, \Gamma, L_M, s_{M_0})$, where $A = U \uplus C$. We say that M is legal for E if for every reachable state (m, e) of $M \| E$ the following holds.

- For all $\ell \in U$ such that $\Delta_\ell(e) \neq \varnothing$ we have $\Gamma_\ell(m) \neq \varnothing$.
- For all $\ell \in C$ such that $\Delta_\ell(e) = \varnothing$ we have $\Gamma_\ell(m) = \varnothing$.

Definition 2.4: (Standard Control) Given a domain model in the form of a DLTS $E = (S, P, A, \Delta, L, s_0)$, where $A = U \uplus C$, and an FLTL formula φ, a solution for the DLTS control problem $\mathcal{E} = \langle E, \varphi, C \rangle$ is a DLTS $M = (S_M, P_M, A, \Delta_M, L_M, s_{0_M})$ such that M is legal for E, $E \| M$ is deadlock free, and $E \| M \models \varphi$.

The synthesis problem for FLTL is 2EXPTIME-complete [16]. Nevertheless, restrictions on the form of the goal and assumption specifications have been studied and found to be solvable in polynomial time. For example, goal specifications consisting uniquely of safety requirements can be solved in linear time, and particular styles of liveness properties such as GR(1) [17] can be solved in quadratic time. An adaptation of GR(1) in the context of LTS has been presented in [18] and is defined as follows:

Definition 2.5: (SGR(1) DLTS Control) A DLTS control problem $\mathcal{E} = \langle E, \varphi, \overline{A_E} \rangle$ is SGR(1) if E is deterministic, and φ is of the form $\varphi = \Box\rho \wedge (\bigwedge_{i=1}^{n} \Box \Diamond a_i \rightarrow \bigwedge_{j=1}^{m} \Box \Diamond g_j)$, where ρ, a_i and g_j are Boolean combinations of fluents. Note that $\Box\rho$ is a safety condition on both the environment and the controller. Furthermore a_i and g_j are liveness assumptions and guarantees on the environment and the controller respectively.

III. Motivating Example

Consider a reconnaissance mission for a UAV, surveying a discretised area. The UAV controls the following actions: takeoff, go[x][y], takePicture[x][y], econoMode, and land. However, during surveillance, the UAV is required to monitor environment actions: arrive[x][y], lowBat, and criticalBat.

The behaviour exhibited by the UAV when not controlled is depicted on the left of Fig. 1, where after taking off it may do an arbitrary action (except for takeoff and land) in its alphabet A before it finally lands. The safety assumption on the environment, as depicted on the right of Fig. 1, ensures that arrive[x][y] may only happen as a result of a go[x][y] action. For the sake of presentation, we only consider an area, consisting of two locations: (1,1) and (1,2).

We want to synthesise a controller for the UAV, satisfying the following safety goals:

1) Landing must only occur after taking a picture for every locations or upon a critical battery alert.

$$\Box(\dot{land} \rightarrow \forall x, y \cdot Sensed[x][y] \vee CritBat)$$

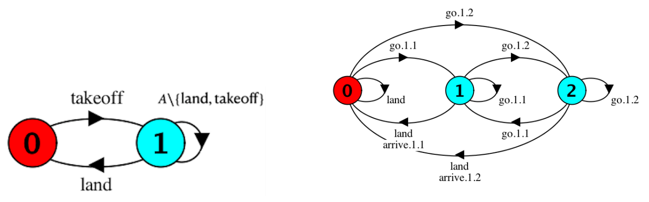

Fig. 1. (left) Model of the UAV and (right) the Environment assumption

where fluent $Sensed[x][y]$ is defined as $\langle\{\text{takePicture[x][y]}\}, \{\text{takeoff}\}, \textit{false}\rangle$ and fluent $CritBat$ as $\langle\{\text{criticalBat}\}, \{\text{takeoff}\}, \textit{false}\rangle$

2) Taking a picture for a particular location must only happen at that location:

$$\Box(\forall x, y \cdot \dot{takePicture}[x][y] \rightarrow At[x][y])$$

where fluent $At[x][y]$ is defined as $\langle\{\text{arrive[x][y]}\}, \{\text{go[x'][y']}, \text{land}\}, \textit{false}\rangle$

3) Low battery alerts must trigger economy flying mode as soon as possible:

$$\Box(\dot{lowBat} \rightarrow ((\neg \bigvee_{\ell \in C \setminus \{\dot{land}, \dot{econoMode}\}} \ell) \; W \; \dot{economode})$$

4) Critical battery alerts must trigger immediate landing:

$$\Box(\dot{criticalBat} \rightarrow ((\neg \bigvee_{\ell \in C \setminus \{\dot{land}\}} \ell) \; W \; \dot{land}))$$

Finally, the liveness goal for the UAV controller is always eventually landing: $\Box \Diamond \, land$. We stress that the safety of landing implies that this happens only after having completed the survey or in response to a critical battery alert. Furthermore, when the UAV issues a go[x][y] command, we require that the environment ensures always eventual arrival: $\Box \Diamond \neg PendingArrival$, where fluent $PendingArrival$ is defined as $\langle\{\text{go[x][y]}\}, \{\text{arrive[x][y]}, \text{land}\}, \textit{false}\rangle$.

No solution for this control problem exists because the environment can flood the controller by generating an infinite number of lowBat and criticalBat events, impeding all controlled actions and hence progress towards the liveness goal. The non existence of such a solution stems from an unrealistic assumption on the environment. Namely, that the environment may impede the progress of the controller merely by a continual notification of a drained battery.

A natural environment assumption that can be introduced to avoid this is to cap the number of lowBat and criticalBat events. In Fig. 2 we show one such constraint in which a maximum of one lowBat and one criticalBat can occur between go commands.

However, this assumption yields controllers that once they have taken off they keep hovering until their batteries are drained, and consequently they land. This way they meet all of their safety goals while achieving their liveness goals by cornering the environment and restricting its set of possible actions to lowBat and criticalBat, i.e, they never issue go commands. Note that if go commands are never issued, arrive events cannot occur and thus the only environment events that can and will eventually occur are lowBat and criticalBat. Indeed, such controllers would never allow the UAV to complete the surveying mission (i.e., landing always occurs because of $criticalBat$ and never because of having achieved $Sensed[x][y]$ for all x and y).

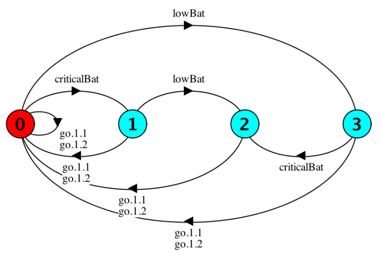

Fig. 2. Naive Environment Assumption to Avoid Flooding.

The assumption that does the trick while avoiding to synthesise trivial controllers is achieved by restricting lowBat and criticalBat to happen *once and only* after issuing a go[x][y] command (i.e. when arrived events are also enabled). This assumption is depicted in Fig. 3.

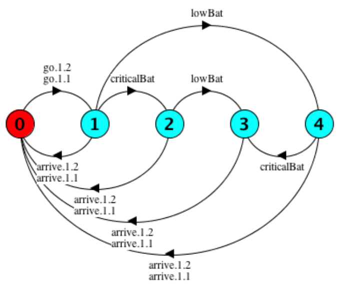

Fig. 3. Assumption to Avoid Flooding.

The example clearly shows how simplifying assumptions (e.g., Figures 2 and 3) can be tricky for the modeller to figure out, while avoiding unrealistic situations for unrealizability in control problems and also avoiding trivial solutions.

IV. Problem Statement

As exhibited in the previous section, the prevalent approach to control in Discrete Event Systems poses serious modelling problems related to the continual triggering of environment events that have to be dealt with by the controller. In many applications, the controller has to execute a sequence of steps (or a finite protocol) in response to a single event. However, this would not be possible if the environment keeps triggering events, flooding the controller with uncontrollable events, and thus impeding its progress towards completing its designated tasks.

By definition, the environment has a double role. It identifies the possible controller actions in a given environment state and it also represents the (adversarial) behaviours. The first corresponds to physical/software restrictions (e.g., go happens only after takeoff); the second corresponds to the scheduling of the next event (e.g., criticalBat can always win the race against land), because the controller cannot disable environment events and the environment always picks the next event out of those selected by the controller and all enabled uncontrollable events in an environment state.

To mitigate this problem, the modeller has to carefully consider how to restrict the environment. Essentially, the modeller is forced to introduce assumptions about the scheduling of the environment and the controller that are an artifact resulting from the definition of (an asymmetric) control problem. These assumptions are not only hard to figure out correctly, but are also far from the actual focus of the control problem under consideration. This makes written specifications harder to read and understand, and consequently trickier to be incrementally developed due to their extensive dependencies. Here, we suggest a more balanced control problem. We call this approach *Run-to-Completion (RTC)* as both the environment and the controller can perform sequences of actions. At the same time, neither can flood the other or delay it indefinitely.

In RTC control the notion of legality is more subtle. The controller has to be able to disable environment actions, this is akin to the controller stating that it still has something to do. However, when some uncontrollable action is enabled by the controller, the controller must allow all of them. Furthermore, to support environment's run-to-completion, the controller must not interrupt the environment when it is the environment that is moving. This amounts to saying that if the environment has moved, the controller must enable all uncontrollable actions. This is formalized below.

Definition 4.1: (Legality under RTC semantics) Consider the DLTSs $E = (S_E, P_E, A, \Delta, L_E, s_{E_0})$ and $M = (S_M, P_M, A, \Gamma, L_M, s_{M_0})$, where $A = U \uplus C$. We say that M is run-to-completion (RTC) legal for E if for every reachable state (e, m) of $E \| M$ the following holds.

- When allowing the environment to move, allow all its possible actions: If $\Gamma_U(m) \neq \varnothing$ then for every $\ell \in U$ such that $\Delta_\ell(e) \neq \varnothing$ we have that $\Gamma_\ell(m) \neq \varnothing$.
- After uncontrolled actions, let the environment progress towards completion: If $m \in \Gamma_U(S_M)$ then for every $\ell \in U$ such that $\Delta_\ell(e) \neq \varnothing$ we have that $\Gamma_\ell(m) \neq \varnothing$.
- For every $\ell \in C$ such that $\Delta_\ell(e) = \varnothing$ we have that $\Gamma_\ell(m) = \varnothing$.

Additionally, we have to ensure that both the environment and the controller are non-Zeno. That is, both do not take an infinite sequence of actions without giving the other opportunities for making progress. On the controller side, we require that all computations are (controller) non-Zeno. On the environment side, we consider only (environment) non-Zeno computations for the satisfaction of the goal. The latter is because it is valid for a controller to chose never to take a controlled action if this ensures its goal.

To formalize the non-Zeno assumption we first introduce four auxiliary formulas: c, u, $pass_E$, and $pass_M$. Given an environment to be controlled E, a candidate controller M, and their parallel composition $E \parallel M$, we assume that in both E and M (separately) for every $\ell \in A$ there are propositions $\ell^p_E \in P_E$ and $\ell^p_M \in P_M$ such that $(s, \ell, s') \in \Delta_E$

iff $\ell^p_E \in L(s)$ and similarly for M. Let $c = \bigvee_{\ell\in C} \dot{\ell}$, $u = \bigvee_{\ell\in U} \dot{\ell}$, $pass_M = \bigwedge_{\ell\in C} \neg\ell^p_M$, and let $pass_E = \bigwedge_{\ell\in U} \neg\ell^p_E$. That is, c and u are formulas specifying the possibility of executing some controllable and uncontrollable actions, respectively. The formulas $pass_E$ and $pass_M$ characterize states where the environment and, respectively, the controller do not enable any uncontrollable and controllable action. That is, in $pass_E$ all uncontrollable actions are *impossible* in the environment and in $pass_M$ all controllable actions (if exist) are *not enabled* by the controller. Note that by the definition of legality the controller can only enable controllable actions that are enabled in the environment.

We now define the formulas ψ_e and ψ_c denoting non-Zeno-ness assumptions on the environment and the controller respectively. Let $\psi_\text{e} = \Box\Diamond(c \vee pass_M)$. That is (if enabled in the environment model), the environment allows infinitely many controllable actions (by M) in the execution or there are infinitely many states visited in which M does not enable controllable actions. Let $\psi_\text{c} = \Box\Diamond(u \vee pass_E)$. That is, the controller allows infinitely many uncontrollable actions (by E) in the execution or there are infinitely many states visited in which E does not enable uncontrollable actions.

Definition 4.2: (RTC Control) Given an environment model $E = (S, P_E, A, \Delta, L_E, s_0)$ and an FLTL formula φ, where $A = U \uplus C$ is defined as before. A solution for the RTC control problem $\mathcal{E} = \langle E, \varphi, C\rangle$ is a DLTS $M = (S_M, P_M, A, \Delta_M, L_M, s_{0_M})$ such that M is RTC legal for E, $E \parallel M$ is deadlock free, and every execution π of $E\|M$ satisfies $\pi \vDash \psi_\text{c} \wedge (\psi_\text{e} \rightarrow \varphi)$.

We note that we cannot move the condition φ_c into the implication. Indeed, this would imply that by *not* fulfilling φ_c the controller is able to force the environment to violate φ_e as well. Thus, the controller would trivially fulfil the goal by blocking the environment forever.

V. Example Revisited

We revisit the example in Sect. III under RTC control. We show how RTC control relieves the modeller from dealing with intricate scheduling issues that are hard to figure out correctly. In fact, the modeller is no longer required to come up with foreign modelling artefacts (i.e., Fig. 2 and Fig. 3) to avoid unrealistic situations for unrealizability or to ensure run-to-completion. We also show how RTC control permits writing loosely-coupled specifications, and thus facilitates incremental development.

In Fig. 4, we show a snippet of the RTC controller for the surveillance mission. Due to the change of control mode we no longer need to cap the number of uncontrolled events to avoid flooding. This is naturally captured in RTC control as both the environment and the controller may perform finite sequences of actions without flooding or delaying each other indefinitely. Note the path via states 0, 1, 2, 3, 17, 16 where the UAV has arrived to location [1][1] but both the criticalBat and the lowBat alarms have been raised. In state 16, an arbitrary number of uncontrolled events (i.e., arrive[1][1], criticalBat, lowBat) can occur; however if fairness assumption ψ_e holds, then the controller will get a chance to execute and at this point it can perform all the controlled actions it needs to do in order to satisfy the *safety* requirements (3) and (4) in Sect. III (i.e., land and econoMode via state 13 to reach state 0).

The same path shows how the environment can also run to completion, raising lowBat and criticalBat (states 3, 17 and 16) should it want to. Alternatively, it may forfeit its turn and the controller may land from state 17.

Also note how in state 3, if given the chance, the controller will both take the picture it needs and go to the next location (states 6 and 7) even though there is no explicit requirement. It does so, as it attempts to do as many actions as it can while progressing towards its liveness goal (i.e., land).

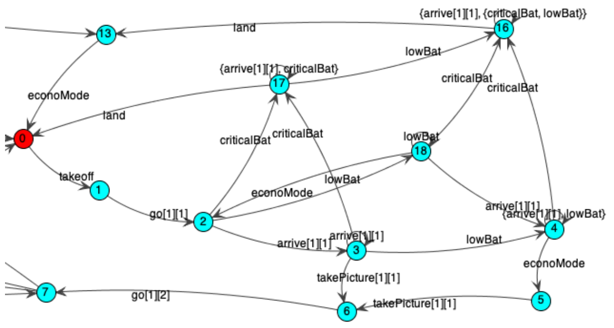

Fig. 4. RTC Controller (Snippet)

Interestingly, the fairness assumptions ψ_e and ψ_c in RTC control remove any possible clashes among subgoals in the specifications, by maintaining fair executions and flexible deadlines. Consider, for instance, the coupling between the *safety* requirements (or subgoals) (3) and (4) in Sect. III to hardcode the concept of *as soon as possible*.

In standard control, the next controllable actions after a lowBat or a criticalBat in (3) and (4) cannot be specified independently as otherwise they would contradict each other, e.g., if the next controllable action after lowBat needs to be econoMode instead of land, econoMode, then if lowBat and criticalBat happen consequtively, the controller cannot respond to either one. These subgoals require attention due to the subtle interactions between them.

In RTC control, we write specifications on top of a fair interaction model where both the controller and the environment are given the chance to run to completion, without counting or fixing deadlines. This suggests that we can write cleaner specifications that remove the explicit dependencies among subgoals, and thus enhancing the readability, and consequently facilitating incremental development.

That said, we introduce convenient schemata to naturally specify high-level concepts like *as soon as possible* and *urgent response* under RTC control. We believe these schemata are more intuitive and less error prone. The schemata relate two Boolean combinations of fluents ϕ and ψ:

$$\begin{array}{lcl} \textsc{Asap}(\psi) & = & ((\bigwedge_{\ell\in C} \neg\dot{\ell})\ W((\bigvee_{\ell\in C} \dot{\ell})\ W\psi)) \\ \textsc{urgRsp}(\phi,\psi) & = & \Box[\phi \rightarrow \textsc{Asap}(\psi)] \end{array}$$

Now, we may replace subgoals (3) and (4) in Sect. III with more natural enunciations. Namely, that econoMode and land are *urgent* response requirements to lowBat and criticalBat respectively:

$$\text{URGRSP}(lowBat, econoMode) \wedge \text{URGRSP}(criticalBat, land)$$

Clearly, these schemata removed the coupling in (3) and (4), while maintaining correctness under RTC control.

VI. Analysis

We show how to solve the RTC control problem by a reduction to a standard control problem. Recall that the environment in standard control always gets to choose the next event out of those selected by the controller and all enabled uncontrollable events in a given environment state. Therefore, we need to model the "act" of yielding control explicitly in the modified control problem. Thus, the analysis may refer directly to when each side is yielding control.

Consider an RTC control problem $\varepsilon = \langle E, \varphi, C\rangle$. We now define a DLTS that captures the transference of control between the environment and the controller:

Definition 6.1: (Yield DLTS) Let U and C be the set of actions controlled by the environment and the controller, respectively. The yield DLTS is defined as $Y = (\{c, e\}, \varnothing, \{\gamma_C, \gamma_E\} \cup C \cup U, \Delta, L, e)$. Where $\Delta = \{(c, \gamma_C, e), (e, \gamma_E, c)\} \cup \{e \xrightarrow{\ell} e \mid \ell \in U\} \cup \{c \xrightarrow{\ell} c \mid \ell \in C\}$.

Now, we define a (standard) control problem over $E \parallel Y$. Let U^+ be $U \cup \{\gamma_E\}$ and C^+ be $C \cup \{\gamma_C\}$. We use the fluents en_e and en_m, which indicate whether Y is in state e or c. Formally, $\text{en}_e = \langle\{\gamma_C\}, \{\gamma_E\}, \textit{true}\rangle$ and $\text{en}_m = \langle\{\gamma_E\}, \{\gamma_C\}, \textit{false}\rangle$. Intuitively, this corresponds to RTC legality as when the environment $E\|Y$ is in a state of the form (s, e) all uncontrollable actions are enabled. Furthermore, uncontrollable actions remain in states of this form (and thus cannot be interrupted). As for controllable actions, they lead to states of the form (s, c), where only controllable actions are enabled, allowing the controller to take a sequence of actions.

The composition $E \parallel Y$ turns all deadlocks in E, which the controller should avoid, to livelocks, where the two sides cooperate to stop time. That is, $E \parallel Y$ gets trapped in an infinite sequence of yield transitions (or Livelock cycle) $s_0, \gamma_E, s_1, \gamma_C, s_2, \gamma_E, \ldots$, namely when both $\Delta_C(s_i)$ and $\Delta_U(s_i)$ are empty.

The Livelock Removal Operator, defined below, removes livelock cycles in a DLTS by removing yield transitions by the controller (resp. environment) to states in which the environment (resp. controller) can only yield back.

Definition 6.2: (Livelock Removal Operator) Let $N = (S, P, A, \Delta, L, s_0)$ be a DLTS obtained by parallel composition with Y. The *livelock removal operator* $live(N)$ is a DLTS $(S, P, A, \Delta', L, s_0)$ where

$$\Delta' = \{(s, \ell, s') \in \Delta \mid \ell \in \{\gamma_C, \gamma_E\} \rightarrow \\ \exists \ell' \notin \{\gamma_C, \gamma_E\} \cdot \Delta_{\ell'}(s') \neq \varnothing\}$$

That is, the transitions with actions γ_C or γ_E are retained only if the environment/controller can do something other than yielding back control immediately. Following the removal of livelocks, we can reduce the RTC control to the following control problem.

Theorem 6.1: (Analysis Control) Consider an environment model $E = (S, P, A, \Delta, L, s_0)$, where $A = U \uplus C$ the set of actions controlled and monitored by E respectively, and an FLTL formula φ. A solution for the RTC control problem $\varepsilon = \langle E, \varphi, C\rangle$ exists if and only if the standard control problem $\varepsilon^+ = \langle live(E\|Y), \varphi^+, C^+\rangle$ is controllable, and φ^+ is defined below. For simplicity, we use A to denote $\bigvee_{\ell \in A} \ell$.

$$\begin{aligned} \varphi^+ = \quad & \square \Diamond (\text{en}_e \vee (\textstyle\bigwedge_{\ell\in U} \neg \ell^p_E)) \wedge \\ & (\square \Diamond (\text{en}_m \vee (\textstyle\bigwedge_{\ell \in C} \neg \ell^p_E)) \rightarrow [\square \Diamond A \rightarrow \varphi]) \end{aligned}$$

Note that φ^+ mimics formula $\psi_c \wedge (\psi_e \rightarrow \varphi)$ of the RTC control formulation with $\psi_c = \square\Diamond(u \vee pass_E)$ and $\psi_e = \square\Diamond (c \vee pass_M)$. There are two differences. First, φ^+ replaces $pass_M$ by $\bigwedge_{\ell\in C} \neg\ell^p_E$. Second, φ^+ disregards traces in ϵ^+ in which the environment and the controller collaborate to stop time. This is done by evaluating φ only on traces satisfying $\square\Diamond A$. The second is not a problem when extracting an RTC controller as the environment of the RTC controller cannot stop time.

The proof of Theorem 6.1 (reported in the full version on [19] due to space limitations) shows that given a solution M^+ of ε^+ we can construct an RTC solution M of ε. The size of M is at most twice the size of M^+.

Corollary 6.1: Given an DLTS solution M^+ for the modified control problem ε^+, we can construct a solution M to ε such that the number of states of M is at most twice the number of states of M^+. Furthermore, if M^+ is deterministic then M is also deterministic.

Proof: We only report the construction of M as its correctness is immediate from the proof of Theorem 6.1. Let $M^+ = (T, P, A^+, \Gamma^+, L^+, t^+_0)$. The components of M are defined as follows:

- $T_M = \{e, c\} \times T$ is the set of states;
- The alphabet $A_M = A^+ \setminus \{\gamma_C, \gamma_E\}$;
- The initial state $t'_0 = (e, t^+_0)$;
- The transition relation Γ is defined below:

$$\left\{((e,t),\ell,(e,t')) \,\middle|\, \begin{array}{l} \ell \in U \text{ and } \exists t_1, t_2 \text{ s.t. } (t, \gamma_E, t_1) \in \Gamma^+, \\ (t_1, \gamma_C, t_2) \in \Gamma^+ \text{ and } (t_2, \ell, t') \in \Gamma^+ \end{array}\right\} \cup$$

$$\left\{((e,t),\ell,(e,t')) \,\middle|\, \begin{array}{l} \ell \in U, (t, \ell, t') \in \Gamma^+ \text{ and} \\ \forall t_1 . (t, \gamma_E, t_1) \in \Gamma^+ \rightarrow \Gamma^+_{\gamma_C}(c, t_1) = \varnothing \end{array}\right\} \cup$$

$$\left\{((e,t),\ell,(c,t')) \,\middle|\, \begin{array}{l} (t, \gamma_E, t'') \in \Gamma^+ \text{ and} \\ (t'', \ell, t') \in \Gamma^+ \text{ and } \ell \in C \end{array}\right\} \cup$$

$$\{((c,t),\ell,(c,t')) \mid \ell \in C \text{ and } (t, \ell, t') \in \Gamma^+\} \cup$$

$$\left\{((c,t),\ell,(e,t')) \,\middle|\, \begin{array}{l} (t, \gamma_C, t'') \in \Gamma^+ \text{ and} \\ (t'', \ell, t') \in \Gamma^+ \text{ and } \ell \in U \end{array}\right\}$$

■

Note that the states of M retain as an extra memory the information of whether a state is on the "environment side" or the "controller side". Intuitively, for a state t of M^+ the controller M adds the memory of whether t was reached by a controllable or uncontrollable transition. If t is reached by a controllable transition, M implements all transitions possible from t and all transitions possible from the γ_C successor of t (if exists). Dually, if t is reached by an uncontrollable transition, M implements all transitions possible from t and all transitions possible from the γ_E successor of t (if exists). Furthermore, whenever possible add a detour that includes both a γ_E and a γ_C before an uncontrollable action.

Theorem 6.2: (Analysis SGR(1) Control) Let E be a DLTS $E = (S, P, A, \Delta, L, s_0)$, where $A = U \uplus C$ is the set of actions controlled and monitored by E, respectively, and let $\varphi = \Box\rho \wedge (\bigwedge_{i=1}^{n} \Box \Diamond a_i \rightarrow \bigwedge_{j=1}^{m} \Box \Diamond g_j)$ be an FLTL formula with ρ, a_i and g_j Boolean combinations of fluents.

A solution for the RTC control problem ε exists if and only if the standard control problem $\langle live(E \| Y), \varphi', C \cup \{\gamma_C\}\rangle$ is controllable, where φ' is as follows.

$$\begin{aligned}&\Box\rho \wedge \Box \Diamond (\mathrm{en}_e \vee (\textstyle\bigwedge_{\ell \in U} \neg \ell_E^p)) \wedge \\ &([\Box \Diamond (\mathrm{en}_m \vee (\textstyle\bigwedge_{\ell \in C} \neg \ell_E^p)) \wedge \textstyle\bigwedge_{i=1}^{n} \Box \Diamond a_i \wedge \Box \Diamond A] \\ &\qquad\qquad \rightarrow \textstyle\bigwedge_{j=1}^{m} \Box \Diamond g_j)\end{aligned}$$

Proof: The only difference is in the location of the safety. Every computation of $M \| live(E \| Y)$ is a computation of $M \| E$ interspersed with yield actions. It follows that $\Box\rho$ holds. ■

Corollary 6.2: The complexity of RTC control with GR(1) goals is in $O(n \times m \times |S|^3)$, where $|S|$ is the number of states of the environment.

Proof: The goal in the modified control problem ϵ^+ is of the form $\Box\rho \wedge \Box \Diamond b \wedge (\bigwedge_{i=1}^{n} \Box \Diamond a_i \rightarrow \bigwedge_{j=1}^{m} \Box \Diamond g_j)$. By adding counters that range over the number of assumptions and the number of guarantees, this kind of goal can be converted to a Streett condition of index 2 [8]. Control problems where the goal is a Streett condition of index 2 can be solved in time cubic in the number of states [20]. ■

VII. Concluding Remarks and Future Directions

We introduced a novel control problem, named *run-to-completion* (RTC), to deal with the asymmetric interaction between the controller and the environment commonly found in DES control. We showed that RTC control can be exploited to synthesise controllers for systems that can initiate and perform sequences of actions while responding correctly to external stimuli. This makes RTC suitable to control componentized systems with complex structures, and where a response to a single external stimulus may require several rounds of propagations among subsystems. Thanks to the flexible deadlines in RTC control, we are no longer required to count (or hardcode) the number of computation steps for the system before it is ready again to react to the next stimulus. Furthermore, we avoid generating trivial controllers and we simplify the specifications by removing the explicit dependencies among subgoals, and thus facilitating incremental development.

We showed that every instance of the RTC control problem can be reduced to a standard control problem, and finally we showed that when SGR(1) goals are used, RTC control can be reduced to Streett control of index 2 [8].

The notion of non-Zeno we have used is strongly related to fairness of the environment and the controller. One could consider extensions in two different directions. Our notion of non-Zeno allows the controller to force the environment to take some action. That is, in some cases where both controllable and uncontrollable actions are possible, we allow the controller to force the environment to move. One could consider a weaker notion of non-Zenoness where the environment is not forced to take actions if it does not wish to do so. Dually, we consider the controller non-Zeno if it often enough gives the environment the option to act (even if the environment cannot act). This corresponds to the notion of weak fairness. One could consider stronger restrictions on controllers in which they would have to be strongly-fair towards the environment. That is, if the environment can act infinitely often it should act infinitely often. Interestingly, one could consider even stronger notions, where strongly-fair controllers in addition completely block the environment only in cases where it is impossible to fulfil the goals when the environment acts infinitely often.

In many cases, studies of control of discrete event systems consider goals that are combinations of safety and non-blocking, while we have considered linear temporal goals. In general, the techniques required to solve the two types of problems are very similar [21]. The techniques developed in this paper can be adapted also to handle the case of non-blocking. In the case of linear temporal goals it is well known that maximally permissive controllers do not exist. It is an interesting question whether RTC-control with safety and non-blocking goals allow for maximally permissive controllers.

A. Related works

In the prominent approach to synthesis (such as *Reactive Synthesis* [1] and *Supervisory Control* [2]) the uncontrolled plant has an advantage over the controller with respect to scheduling. Although, this may seem a natural understanding of the synthesis problem, where the controller is supposed to react to every possible behaviour of the plant, it is not always an appropriate assumption. In many cases, the uncontrolled plant is not completely adversarial (see [22]), and many undesired behaviours are practically infeasible and should be ruled out by definition, i.e., due to physical and/or software restrictions. For instance, a robot moving in an arena is restricted by its fixed structure [23], and thus it does not make sense to consider all possible paths between two points.

These restrictions are usually dealt with by introducing domain specific assumptions over the plant (see [24], [7]). However, these assumptions are not usually obvious and in many cases lead to spurious solutions (see [18]).

A classic domain in which the uncontrolled plant is not completely adversarial is that of embedded systems where reactive languages (e.g., [3], [4], [5]) adopt a synchronous hypothesis where the system can react to an external stimulus with all the computation steps it needs [6]. To the best of our knowledge, RTC control is the first to automatically handle such assumptions.

There have been many studies that focus on relating supervisory control and reactive synthesis, see [25], [21].

However, some aspects are still not considered and that become more apparent with the approach presented herein. RTC control introduces a turn-based interaction between the controller and the plant that is similar to that of Reactive Synthesis [1] for state-based models (i.e., no transition labels, only state propositions). Furthermore, in Reactive Synthesis both the controller and its adversary may perform in their own turn multiple actions concurrently. Yet in Reactive Synthesis the upper bound on actions per turn is determined by the number of state propositions, which is defined manually by the specifier before synthesis and it is not obvious how to reason about the order of concurrent events in state-based modelling. This becomes very important when dealing with systems that are required to do several rounds of data or control propagations among their subparts in response to external stimuli. Indeed, we may not know a-priory how many rounds of propagations are required or the order of events happening during the propagation. Furthermore, restricting the order (or the interleaving) of concurrent events manually might largely impact on the performance of the system under consideration. This is because hardcoded-orderings may easily sequentialise concurrent events that can safely be executed in parallel. Clearly, the last scenario poses a problem for both state-based models and event-based ones. Namely, once the events that can be executed simultaneously are explicitly identified, the flooding of adversarial events from the environment becomes as problematic as the one of uncontrollable events in discrete event systems.

References

[1] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ser. POPL '89. New York, NY, USA: ACM, 1989, pp. 179–190. [Online]. Available: http://doi.acm.org/10.1145/75277.75293

[2] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[3] N. Halbwachs, *Synchronous Programming of Reactive Systems*. Berlin, Heidelberg: Springer-Verlag, 2010.

[4] G. Berry, "Real time programming: Special purpose or general purpose languages," in *Information Processing 89, Proceedings of the IFIP 11th World Computer Congress, San Francisco, USA, Aug. 28 - Sep. 1, 1989*, G. Ritter, Ed. North-Holland/IFIP, 1989, pp. 11–17.

[5] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[6] R. de Simone, J. Talpin, and D. Potop-Butucaru, "The synchronous hypothesis and synchronous languages," in *Embedded Systems Handbook*, R. Zurawski, Ed. CRC Press, 2005. [Online]. Available: https://doi.org/10.1201/9781420038163.ch8

[7] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," *Lecture notes in computer science*, vol. 3855, pp. 364–380, 2006.

[8] R. S. Streett, "Propositional dynamic logic of looping and converse is elementarily decidable," *Inf. Control.*, vol. 54, no. 1/2, pp. 121–141, 1982.

[9] J. Magee and J. Kramer, *Concurrency: state models & Java programs*. Wiley New York, 2006.

[10] D. Giannakopoulou and J. Magee, "Fluent model checking for event-based systems," in *Proceedings of the 9th European software engineering and 11th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2003, pp. 257–266.

[11] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *IEEE Transactions on Software Engineering*, vol. 26, pp. 978–1005, October 2000. [Online]. Available: http://portal.acm.org/citation.cfm?id=357525.357521

[12] R. Kazhamiakin, M. Pistore, and M. Roveri, "Formal verification of requirements using spin: A case study on web services," in *Proceedings of the Software Engineering and Formal Methods, Second International Conference*, ser. SEFM '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 406–415. [Online]. Available: http://dx.doi.org/10.1109/SEFM.2004.19

[13] S. Uchitel, R. Chatley, J. Kramer, and J. Magee, "Fluent-based animation: exploiting the relation between goals and scenarios for requirements validation," in *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, 2004, pp. 208–217.

[14] E. Letier, J. Kramer, J. Magee, and S. Uchitel, "Fluent temporal logic for discrete-time event-based models," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 5, p. 70–79, Sept. 2005. [Online]. Available: https://doi.org/10.1145/1095430.1081719

[15] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, ser. ESEC/FSE-9. New York, NY, USA: ACM, 2001, pp. 109–120. [Online]. Available: http://doi.acm.org/10.1145/503209.503226

[16] N. D'Ippolito, "Synthesis of event-based controllers for software engineering," Ph.D. dissertation, Imperial College London, The address of the publisher, 3 2013, http://cor.to/8UmX.

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

[18] N. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel, "Synthesising non-anomalous event-based controllers for liveness goals," *ACM Tran. Softw. Eng. Methodol.*, vol. 22, 2013.

[19] Y. Abd Alrahman, V. Braberman, N. D'Ippolito, N. Piterman, and S. Uchitel, "Synthesis of run-to-completion controllers for discrete event systems." [Online]. Available: https://arxiv.org/abs/2009.05554

[20] N. Piterman and A. Pnueli, "Faster solutions of Rabin and Streett games," in *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*. IEEE, 2006, pp. 275–284.

[21] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi, "Supervisory control and reactive synthesis: a comparative introduction," *Discret. Event Dyn. Syst.*, vol. 27, no. 2, pp. 209–260, 2017. [Online]. Available: https://doi.org/10.1007/s10626-015-0223-0

[22] R. Ehlers, R. Könighofer, and R. Bloem, "Synthesizing cooperative reactive mission plans," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, 2015, pp. 3478–3485. [Online]. Available: https://doi.org/10.1109/IROS.2015.7353862

[23] K. W. Wong and H. Kress-Gazit, "Let's talk: Autonomous conflict resolution for robots carrying out individual high-level tasks in a shared workspace," in *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, 2015, pp. 339–345. [Online]. Available: https://doi.org/10.1109/ICRA.2015.7139021

[24] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009. [Online]. Available: https://doi.org/10.1109/TRO.2009.2030225

[25] A. Schmuck, T. Moor, and R. Majumdar, "On the relation between reactive synthesis and supervisory control of non-terminating processes," *Discret. Event Dyn. Syst.*, vol. 30, no. 1, pp. 81–124, 2020. [Online]. Available: https://doi.org/10.1007/s10626-019-00299-5